DELIGHT.MIMO: AN INTERACTIVE SYSTEM FOR OPTIMIZATION-BASED MULTIVARIABLE CONTROL SYSTEM DESIGN

by

Tzyh-Lih Wuu

Memorandum No. UCB/ERL M86/90

8 December 1986

DELIGHT.MEMO: AN INTERACTIVE SYSTEM FOR OPTIMIZATION-BASED MULTIVARIABLE CONTROL SYSTEM DESIGN

by

Tzyh-Lih Wuu

Memorandum No. UCB/ERL M86/90

8 December 1986

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

DELIGHT.MIMO: An Interactive System for Optimization-Based Multivariable Control System Design

Tzyh-Lih Wuu

Ph.D.

Dept. of Electrical Engineering
and Computer Sciences

Signature: _____
Committee Chairman

ABSTRACT

DELIGHT.MIMO is an interactive system for applying optimization techniques to multivariable control system design. The system provides a highly interactive language, an optimization library, a control system time and frequency response simulator, a database, a symbolic differentiator and a user interface.

DELIGHT.MIMO consists of two portable modules: DELIGHT, which was completed by Dr. William Nye [Nye.3] in 1983, and MIMO, which is currently completed. A set of semi-infinite nondifferentiable optimization algorithms (such as the Polak-Wardi algorithm) which have been found to be effective in multivariable control systems design, is included in the DELIGHT optimization library. A *multiphase* feasible directions algorithm is presented for problems containing a sequence of subproblems. The MIMO system provides all control system related simulations, as well as the sensitivity function of these responses which are required by DELIGHT optimization algorithms. An interactive database has been implemented so that users can define a hierarchical parameterized interconnected system associated with a set of design specifications. Users may interactively tune a system in order to view simulation outputs at various parameter values. To compute the sensitivity functions efficiently, a symbolic differentiator is available, which differentiates system state space matrices with respect to the design parameters.

A user interface is constructed to allow both novice and experienced users to make efficient use of MIMO's facilities. Both alphanumeric and graphical/mouse

inputs are available in the MIMO system so that it can match the capabilities of various terminals. The graphical library of the package is implemented based on the Graphical Kernel System (GKS) standard which improves the portability of the system. The user interface provides both command driven and menu driven interactive environments. A graphics editor for defining linear subsystems, input signal generators and interconnected systems is constructed, which allows both top-down and bottom-up problem formulations. To display multiple outputs from various systems, a window manager is developed to direct the outputs to a set of user defined windows. A manual design tuning capability, based on sensitivity calculations, is developed. A set of windows can be activated for use during the tuning process. An hierarchical on-line "Help" facility is also included.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Professor Elijah Polak for generously spending his time supervising my work. He has supported me with enthusiasm and was available for numerous discussions throughout the course of my research.

Professor Charles Desoer took interest in my studies. His encouragement and advice helped me make many good decisions. I have enjoyed and benefited from the many discussions I had with him.

I wish to thank Professor David Mayne for making it possible for me to visit Imperial College and for his encouragement. I am indebted to Professor Karl Pister for serving on my dissertation committee.

I would like to acknowledge the many stimulating discussions with my fellow graduate students. Ming-Jeh Chen, Septimiu E. Salcudean, Joe Higgins, Bill Nye, Mark Austin, Andrew Heunis, Ted Baker, Ywh-Pyng Harn, Joe Landers, Ren-Song Tsay, Jyuo-Min Shyu and Michael Fan have given me many suggestions during the course of my studies at Berkeley. Septimiu E. Salcudean and Joe Higgins, in particular, had a major influence on my thesis.

I am grateful to my parents for their constant, loving support during my graduate years. My wife, Yu-Fuen, has stood by me during the most difficult moments with patience and encouragement. Her love and moral support have kept me going, and I wish to thank her for that.

The research was supported in part by the National Science Foundation Grant ECS-8121149, Air Force Office of Scientific Research Grant 83-0361, Office of Naval Research Contract N00014-83-K-0602, The Semiconductor Research Corp. Contract SRC-82-11-008, The State of California MICRO Program and the General Electric Company. I would like to acknowledge these organizations for their assistance.

Table of Contents

CHAPTER 1

INTRODUCTION

1.1 Optimization-Based, Computer-Aided Control System Design

The growing importance of computer-based tools for the analysis and design of engineering systems has resulted in a relatively new engineering specialty which is commonly referred to as computer-aided design (CAD). Over the past twenty-five years, control theory has evolved to a state where the digital computer has become an indispensable design tool and has led to the development of a new approach to control system design, i.e., *Computer-Aided Control System Design* (CACSD).

The design of a good CACSD system usually requires a team effort involving control engineering specialists, computer scientists, optimization experts, and numerical analysts. The need for such a breadth is one of the reasons why there are so few high quality CACSD systems today. A survey of existing CACSD systems is given in [Jam.1]. Most of these systems provide control engineers with various tools for analysis and synthesis of control systems. However, none of these systems enables the designer to make use of *parametric optimization.*

Parametric optimization has been used in CAD of engineering systems for a long time. It is a powerful tool for the selection of favorable values for design variables. Presently, its use in many areas of engineering design is expanding rapidly (see for example, [Pol.11]). It is generally true that design complexity is increasing more rapidly than the number of design parameters. As design specifications become more complex, the size of design problems becomes larger and computing tools become more advanced, the use of parametric optimization in control system design becomes inevitable. Early interpretations of control system design as an optimization problem, can be found in [Dav.1, Kar.1, Pol.13, Zak.1]. In fact, even the linear-quadratic regulator problem [Ath.1, Kwa.1] can be viewed as a parametric optimization problem [Pol.10].

The parametric optimization algorithms of the sixties and early seventies (see, e.g., [Lue.1], [Pol.14]) were only able to deal with optimization problems of the form

$$\min\{ f(\mathbf{x}) \mid g^j(\mathbf{x}) \leq 0,\ j = 1,2,\cdots,m\ ;\ h^k(\mathbf{x}) = 0,\ k = 1,2,\cdots,l \}, \quad (1.1.1)$$

in which the cost function $f : \mathbb{R}^n \to \mathbb{R}$, and the constraint functions $g^j, h^k : \mathbb{R}^n \to \mathbb{R}$, are continuously differentiable and in which there is only a *finite number* of equality and inequality constraints. In the design of single-input single-output (SISO) control systems, specifications on quantities such as phase and gain margins, in the frequency domain, and step response rise time, settling time and overshoot, in the time domain, are commonplace (see, e.g., [Hor.2]). With the development of modern multi-input multi-output (MIMO) control system design theory, these SISO frequency domain specifications have been generalized in terms of norm (e.g., largest singular value, or H^∞ norm) bounds on various transfer function matrices (see, e.g., [Doy.1, Cru.1, Saf.1, Leh.1, Zam.1, Fra.1]). It takes very little time to come to the conclusion that all these requirements do not lead to a classical, differentiable, finitely constrained optimization problem of the form (1.1.1), but rather to a *nondifferentiable, infinitely constrained* optimization problem in the *finite dimensional* design parameter $\mathbf{x}$ which represents the free compensator coefficients. Problems of this kind are referred to as *semi-infinite* optimization problems (or semi-infinite optimization programs, i.e., SIP's). Thus, the appropriate canonical optimization problem into which control system design problems must be transcribed has the form

$$\min\{ f(\mathbf{x}) \mid g^j(\mathbf{x}) \leq 0,\ j = 1,2,\cdots,m\ ;$$
$$\phi^k(\mathbf{x},\mathbf{y}_k) \leq 0,\ \mathbf{y}_k \in Y_k,\ k = 1,2,\cdots,l \}, \quad (1.1.2)$$

in which the functions $g^j : \mathbb{R}^n \to \mathbb{R}$ are continuously differentiable and the functions $f : \mathbb{R}^n \to \mathbb{R}$ and $\phi^k : \mathbb{R}^n \times \mathbb{R}^{p_k} \to \mathbb{R}$ are *locally Lipschitz continuous* [Cla.1, Pol.3, Pol.4]. The sets $Y_k \subset \mathbb{R}^{p_k}$ are assumed to be compact. In control system design, the sets Y_k may be finite time or frequency intervals.

The realization that existing optimization algorithms were inadequate for solving many engineering design problems has stimulated much research. Over the last decade, a number of optimization algorithms which were developed specifically for the solution of optimization problems of the form (1.1.2) have been introduced (see for

example [Gon.1, Gon.2, Oet.1, Pol.2, Pol.3, Pol.4, Pol.12, Tra.1]). To date the design experience using these algorithms is still limited, though quite promising.

1.2 Common Impediments to the Use of Optimization in Engineering Design

There are three reasons why optimization has not been used much for design. First, as Brayton and Spence emphasize in [Bra.1], it is difficult for designers to formulate their design problems as well-posed optimization problems. Second, it is difficult for designers to understand how to successfully use the algorithms which have been developed or implemented. Third, it is difficult to couple existing simulation programs to optimization algorithms.

Parametric optimization is a sophisticated tool. Therefore, it requires a designer to have a certain amount of knowledge of optimization theory in order to use it successfully. For instance, after a design problem is formulated, the designer must (1) choose an appropriate optimization algorithm for the problem, (2) translate the mathematical formulations of the problem into a database which can be recognized by both the optimization algorithm and simulation programs, and (3) initialize parameter values of the algorithm or reinitialize those parameters during the design process in order to solve the problem properly and efficiently. The details of each step are often not only difficult and tedious, but also error prone. A designer can easily get frustrated after a few attempts to use an optimization algorithm which does not perform well.

There are two main limitations of both past and present simulation programs which cause their coupling to optimization algorithms to be difficult. The first limitation is that most simulators have not been designed to perform as function evaluation routines for an optimization process. Even when they have rerun capabilities for modified input parameter values, it is either through interaction or through additional input files. The second limitation of most simulators is that they do not compute sensitivity functions with respect to design parameters. These functions are essential for the *efficient* use of parametric optimization.

DELIGHT, [Nye.3] which was completed in 1983, is one of the pioneering software systems in optimization-based computer aided design. It supports an environment which simplifies the process of combining simulation and optimization. The DELIGHT system provides a high level interactive language, RATTLE, which has *define* and *macro* language extension capabilities. An optimization algorithm library was constructed in a modular fashion so that sub-blocks of an optimization procedure could be conveniently selected and substituted. However in some situations, a major limitation of the DELIGHT system is the run time speed of the RATTLE language, which is about 10 to 100 times slower than FORTRAN or C on a VAX 11/780 machine. Thus, we use DELIGHT for the optimization algorithm portion of DELIGHT.MIMO only.

1.3 Design Objectives for DELIGHT.MIMO

In this dissertation, an optimization-based computer-aided control system design package, DELIGHT.MIMO, is designed and implemented. In view of the difficulties of using optimization in design as reviewed in Section 1.3, it was felt that DELIGHT.MIMO should have the following features:

(i) An efficient simulator which evaluates not only time and frequency responses of a control system, but also their sensitivities with respect to design parameters of the system.

(ii) A database which contains descriptions of control systems to be designed and design specifications and objectives to be achieved.

(iii) A set of semi-infinite nondifferentiable optimization algorithms (such as the Polak-Wardi algorithm) which have been found to be effective in control systems design, should be included in the DELIGHT optimization library.

(iv) A friendly user interface to enable both novice and experienced users to set up their design problems and make efficient use of DELIGHT.MIMO's facilities.

(v) An I/O handler to enable DELIGHT.MIMO to communicate with other existing CACSD packages easily.

To summarize the proposed design objectives, the configuration of the DELIGHT.MIMO system is illustrated in Fig. 1.1. It consists of two portable, independent modules: DELIGHT and MIMO. DELIGHT contains the optimization algorithm library and various interaction aspects required for efficient use of optimization algorithms, while MIMO provides all of the five items discussed above.

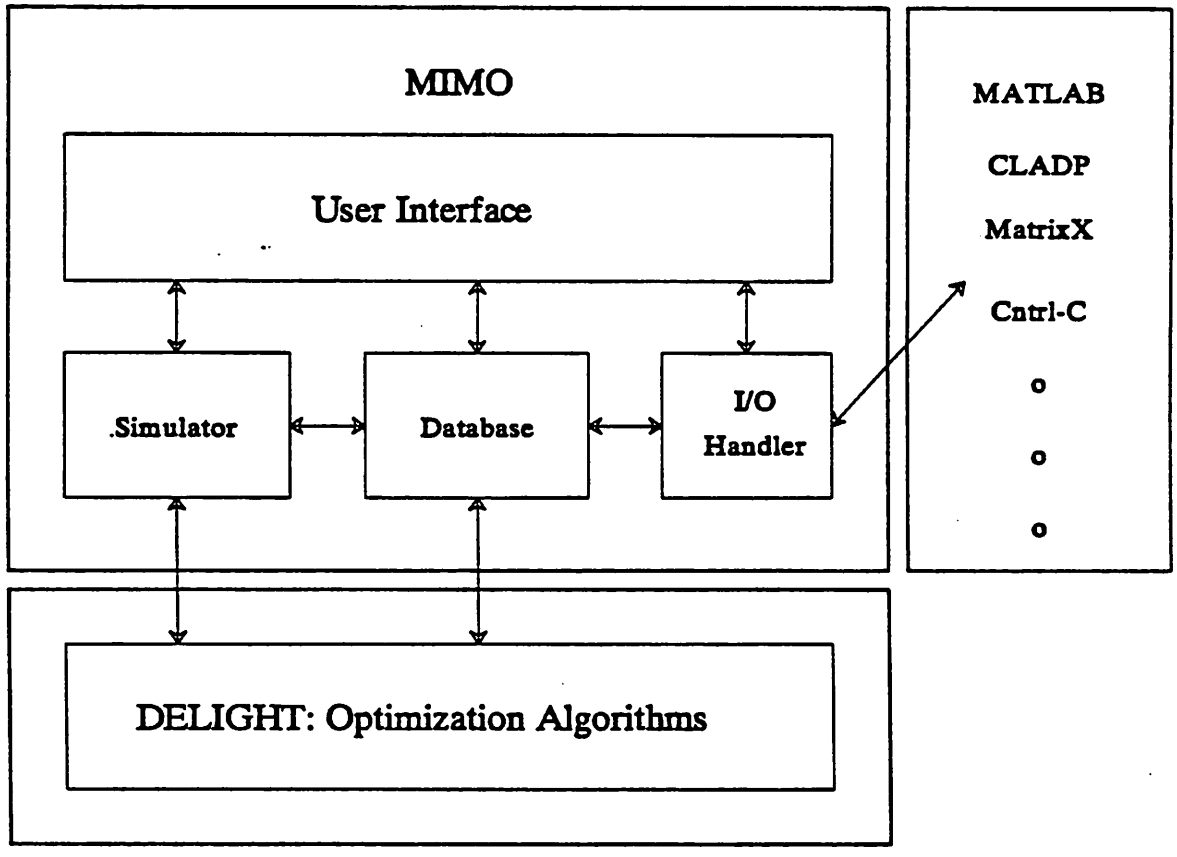


Fig. 1.1. The DELIGHT.MIMO system

1.4 Dissertation Outline

This dissertation is in two parts. The first part, Chapters 2 through 5, presents the mathematical theory needed for the construction of the DELIGHT.MIMO system. The dynamic systems which are being considered in this thesis are linear, time-invariant, finite dimensional, multivariable control systems. In Chapter 2, we begin with a description and formulation of this class of multivariable control systems. Then, various design specifications are transcribed into semi-infinite costs and constraints in Chapter 3. In formulating design specifications as inequalities or semi-infinite inequalities one must take care to ensure that the resulting functions satisfy certain minimum continuity requirements and that the implied computations are reasonably well conditioned. In particular, in order to apply current algorithms or develop rigorously algorithms for optimization problems involving nondifferentiable functions, it is necessary that these functions be at least locally Lipschitz continuous [Cla.1]. After examing the needs of various classes of responses to be computed by the MIMO system, Chapter 4 provides numerical algorithms which meet these needs for optimization-based design. In design applications, one typically considers the influences of multiple design criteria by dividing the design process into segments and sequentially working through modifications of the design with different specifications or objectives. In Chapter 5, we introduce a *multi-phase* method of feasible directions which has the capability of simultaneously considering multiple objectives with this sequential design process.

The second part of the dissertation, Chapters 6 through 8, describes the basic design ideas behind the DELIGHT.MIMO system. We start with a description of the MIMO database system, which manages a set of files and internal main frame storage. Each file stored in ASCII format contains the definitions of various parts of the control system problem description. These files can be read and transformed into data structures. Both the files and the data structures support a hierarchical description of control problems. These hierarchical relations are described in detail in Chapter 6. The user interface to the MIMO system has two major tasks. The first includes database access, query, update and screen I/O constructs. The second task is to control

the MIMO computing environment. Chapter 7 describes the design and implementation of this user interface. Finally, in Chapter 8, two design examples are given. We then conclude with the main contributions of this work and suggested directions for future research in Chapter 9. Appendices are included which explain several MIMO implementation issues.

CHAPTER 2

INTERCONNECTED CONFIGURATION AND SYSTEM DESCRIPTION

2.1 Introduction

Many configurations are used in control systems design for problems involving complex systems such as power systems, circuit systems, robot control systems, defense systems, large space structures. One of the simplest configurations used is the two degrees of freedom configuration shown in Fig. 2.1.

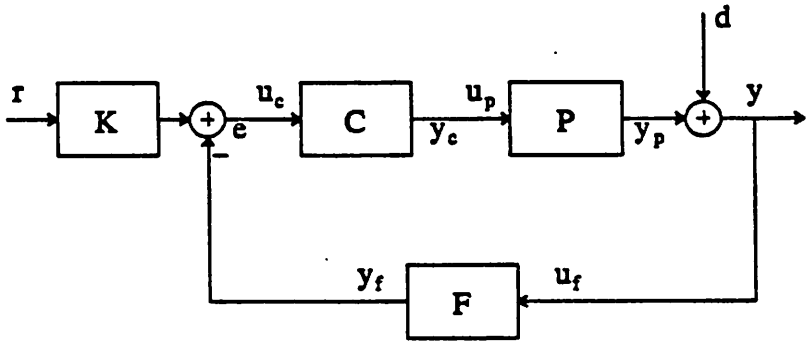


Fig. 2.1. Two degrees of freedom control system

In this chapter we define a general configuration for linear time-invariant multivariable interconnected systems. Section 2.2 describes the interconnected system Σ consisting of a number of linear subsystems and input signals. Sections 2.3 and 2.4 give models of each subsystem and input signal, which can be described by algebraic and differential equations. The models of subsystems and input signals, and the assumption of well posedness, lead to a systematic development of the algebraic and differential equations governing the interconnected systems. The resulting equations are used in the subsequent analysis, simulation and design of control systems.

2.2 Interconnection Structure of Control Systems

Consider an interconnected system Σ of μ given multivariable subsystems, each denoted by S_k with input and output dimensions n_{ik} and n_{ok}, where $k \in \underline{\mu} \triangleq \{1,2,\cdots,\mu\}$. Let each subsystem have the associated interconnection structure [Cal.1] shown in Fig. 2.2:

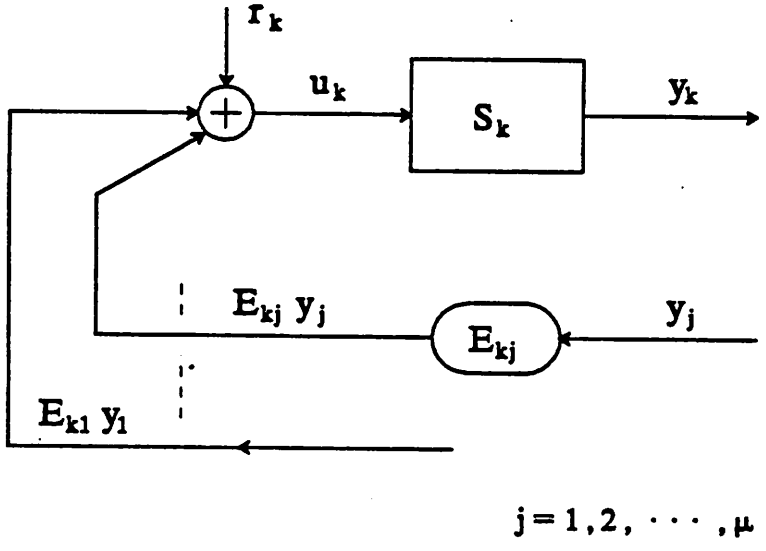


Fig. 2.2. Associated interconnected structure for subsystem S_k.

To each subsystem S_k with input $\mathbf{u}_k$ and output $\mathbf{y}_k$, we associate a summing node with the following characteristics:

(i) Its output is $\mathbf{u}_k$.

(ii) Its inputs are

(a) an exogenous input $\mathbf{r}_k$ (always assigned) with dimension n_{ik} (in time domain analysis, some of these inputs may be zero).

(b) feedback input $\mathbf{E}_{kj}\mathbf{y}_j$, $j = 1,2,\cdots,\mu$, where $\mathbf{E}_{kj} \in \mathbb{R}^{n_{ik} \times n_{oj}}$ denotes the constant gain matrix from $\mathbf{y}_j$ to the kth summing node (some of these

matrices may be zero).

An example of an interconnection is shown in Fig. 2.3.

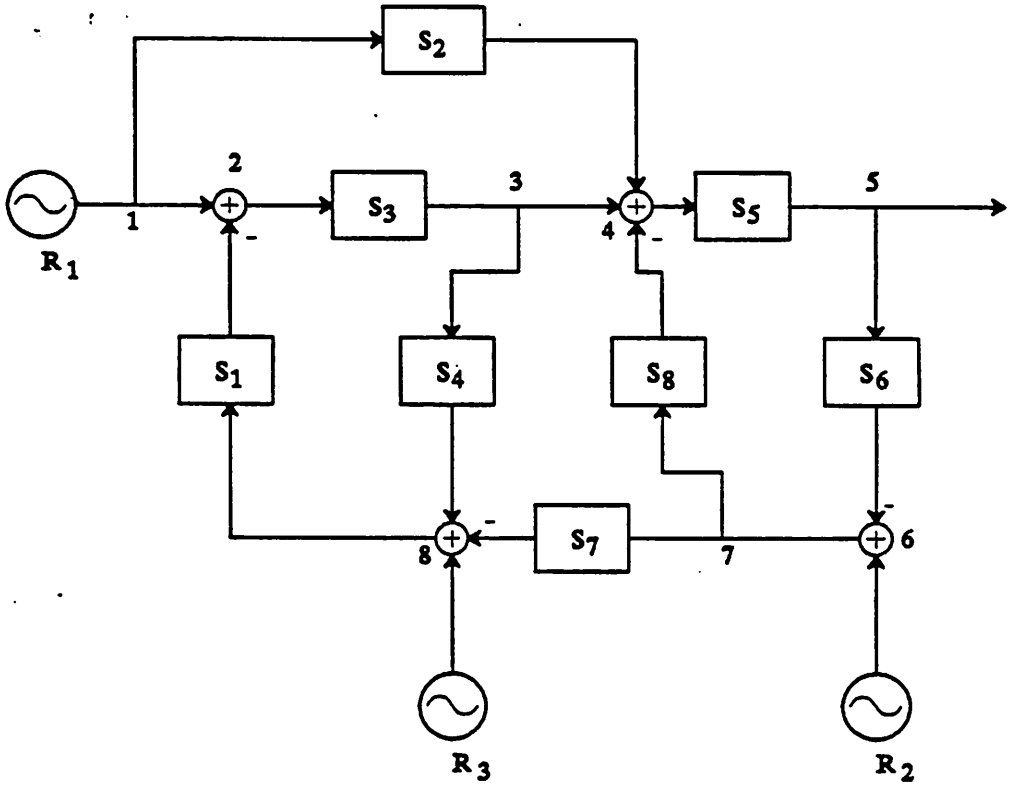


Fig. 2.3 An interconnected system Σ.

Hence the subsystems S_k are interconnected according to the equations:

$$\mathbf{u}_k = \mathbf{r}_k + \sum_{j=1}^{\mu} \mathbf{E}_{kj}\mathbf{y}_j, \qquad \forall\, k \in \underline{\mu}. \tag{2.2.1}$$

2.3 Description of Parameterized Linear Subsystems

The class of dynamic systems that we consider in this dissertation are described by parameterized linear time-invariant ordinary differential equations and algebraic equations. Suppose that the dynamics of the kth subsystem S_k, $k \in \underline{\mu}$ are given by

$$\dot{\mathbf{x}}_k = \mathbf{A}_k(\mathbf{p}_k)\,\mathbf{x}_k + \mathbf{B}_k(\mathbf{p}_k)\,\mathbf{u}_k\,, \tag{2.3.1a}$$

$$\mathbf{y}_k = \mathbf{C}_k(\mathbf{p}_k)\,\mathbf{x}_k + \mathbf{D}_k(\mathbf{p}_k)\,\mathbf{u}_k\,, \tag{2.3.1b}$$

where $\mathbf{x}_k(t,\mathbf{p}_k) \in \mathbb{R}^{n_{sk}}$ is the state, $\mathbf{p}_k \in \mathbb{R}^{n_{pk}}$ is the design parameter vector, $\mathbf{A}_k{:}\mathbb{R}^{n_{pk}} \to \mathbb{R}^{n_{sk}\times n_{sk}}$, $\mathbf{B}_k{:}\mathbb{R}^{n_{pk}} \to \mathbb{R}^{n_{sk}\times n_{ik}}$, $\mathbf{C}_k{:}\mathbb{R}^{n_{pk}} \to \mathbb{R}^{n_{ok}\times n_{sk}}$, and $\mathbf{D}_k{:}\mathbb{R}^{n_{pk}} \to \mathbb{R}^{n_{ok}\times n_{ik}}$ are continuously differentiable matrices, and $\mathbf{u}_k(t) \in \mathbb{R}^{n_{ik}}$ is the input. The transfer function $\hat{\mathbf{G}}_k(s,\mathbf{p}_k)$ from $\mathbf{u}_k$ to $\mathbf{y}_k$ can be obtained by taking the Laplace transform of the equations (2.3.1a,b). Hence

$$\hat{\mathbf{G}}_k(s,\mathbf{p}_k) = \mathbf{D}_k(\mathbf{p}_k) + \mathbf{C}_k(\mathbf{p}_k)[s\mathbf{I}-\mathbf{A}_k(\mathbf{p}_k)]^{-1}\mathbf{B}_k(\mathbf{p}_k)\,. \tag{2.3.2}$$

2.4 System Equations of Interconnected Systems

According to the interconnection structure described in Section 2.2, it is convenient to lump the state equations of the μ subsystems (2.3.1a,b) and the interconnection of the subsystems and the input signals (2.2.1) into the following block diagonal form:

$$\dot{\mathbf{x}}_c = \mathbf{A}_d(\mathbf{p}_c)\,\mathbf{x}_c + \mathbf{B}_d(\mathbf{p}_c)\,\mathbf{u}_c\,, \tag{2.4.1a}$$

$$\mathbf{y}_c = \mathbf{C}_d(\mathbf{p}_c)\,\mathbf{x}_c + \mathbf{D}_d(\mathbf{p}_c)\,\mathbf{u}_c\,, \tag{2.4.1b}$$

$$\mathbf{u}_c = \mathbf{r}_c + \mathbf{E}\,\mathbf{y}_c\,, \tag{2.4.1c}$$

where

$$\mathbf{p}_c \triangleq [\mathbf{p}_1^T,\mathbf{p}_2^T,\cdots,\mathbf{p}_\mu^T]^T,\ \mathbf{x}_c \triangleq [\mathbf{x}_1^T,\mathbf{x}_2^T,\cdots,\mathbf{x}_\mu^T]^T,\ \mathbf{y}_c \triangleq [\mathbf{y}_1^T,\mathbf{y}_2^T,\cdots,\mathbf{y}_\mu^T]^T,$$

$$\mathbf{u}_c \triangleq [\mathbf{u}_1^T,\mathbf{u}_2^T,\cdots,\mathbf{u}_\mu^T]^T,\ \mathbf{r}_c \triangleq [\mathbf{r}_1^T,\mathbf{r}_2^T,\cdots,\mathbf{r}_\mu^T]^T,\ \mathbf{E} \triangleq [\,\mathbf{E}_{kj}]_{k,j\in\mu},$$

$$\mathbf{A}_d(\mathbf{p}_c) \triangleq \mathrm{diag}[\mathbf{A}_1(\mathbf{p}_1),\,\mathbf{A}_2(\mathbf{p}_2),\cdots,\mathbf{A}_\mu(\mathbf{p}_\mu)]\,,$$

$$\mathbf{B}_d(\mathbf{p}_c) \triangleq \mathrm{diag}[\mathbf{B}_1(\mathbf{p}_1),\,\mathbf{B}_2(\mathbf{p}_2),\cdots,\mathbf{B}_\mu(\mathbf{p}_\mu)]\,,$$

$$\mathbf{C}_d(\mathbf{p}_c) \triangleq \mathrm{diag}[\mathbf{C}_1(\mathbf{p}_1),\,\mathbf{C}_2(\mathbf{p}_2),\cdots,\mathbf{C}_\mu(\mathbf{p}_\mu)]\,,$$

$$\mathbf{D}_d(\mathbf{p}_c) \triangleq \mathrm{diag}[\mathbf{D}_1(\mathbf{p}_1),\,\mathbf{D}_2(\mathbf{p}_2),\cdots,\mathbf{D}_\mu(\mathbf{p}_\mu)]\,.$$

Let $n_{s,c} \triangleq \sum_{l=1}^{\mu} n_{sl}$, $n_{i,c} \triangleq \sum_{l=1}^{\mu} n_{il}$, $n_{o,c} \triangleq \sum_{l=1}^{\mu} n_{ol}$ and $n_{p,c} \triangleq \sum_{l=1}^{\mu} n_{pl}$. A block diagonal

transfer function for (2.4.1a,b) is given by

$$\hat{\mathbf{G}}_d(s,\mathbf{p}_c) \triangleq \text{diag}[\hat{\mathbf{G}}_1(s,\mathbf{p}_1),\hat{\mathbf{G}}_2(s,\mathbf{p}_2),\cdots,\hat{\mathbf{G}}_\mu(s,\mathbf{p}_\mu)]\,. \tag{2.4.2}$$

Therefore, the interconnected system Σ can be shown in Fig. 2.4.

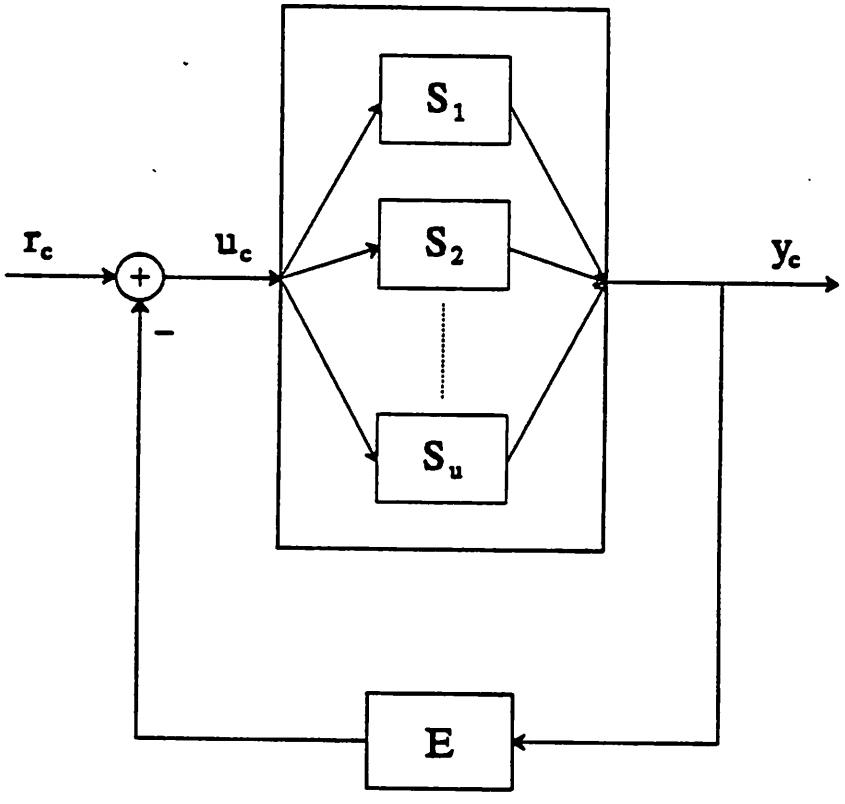


Fig. 2.4. Block Diagonal Form of Interconnected System Σ.

Assumption 2.4.1 : (Well Posedness) With S_k defined by (2.3.1a,b), and equations (2.4.1a,b,c) and (2.4.2), we assume that

$$\det[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)] = \det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(\infty,\mathbf{p}_c)] \neq 0\,, \tag{2.4.3}$$

for all $\mathbf{p}_c \in \mathbb{R}^{n_{p,c}}$ (or at least in a range of interest).

Assumption 2.4.1 is a necessary and sufficient condition for all closed-loop transfer functions $\hat{\mathbf{H}}_{y_j r_k}(s,\mathbf{p}_c)$ and $\hat{\mathbf{H}}_{u_i r_k}(s,\mathbf{p}_c)$, $i,j,k \in \mu$, to be *proper*, for all $\mathbf{p}_c \in \mathbb{R}^{n_{p,c}}$.

Under Assumption 2.4.1, the system equations (2.4.1a,b,c) can be reduced to the following closed loop state equations:

$$\dot{\mathbf{x}}_c(t,\mathbf{p}_c) = \mathbf{A}_c(\mathbf{p}_c)\,\mathbf{x}_c(t,\mathbf{p}_c) + \mathbf{B}_c(\mathbf{p}_c)\,\mathbf{r}_c(t), \tag{2.4.4a}$$

$$\mathbf{y}_c(t,\mathbf{p}_c) = \mathbf{C}_c(\mathbf{p}_c)\,\mathbf{x}_c(t,\mathbf{p}_c) + \mathbf{D}_c(\mathbf{p}_c)\,\mathbf{r}_c(t), \tag{2.4.4b}$$

$$\mathbf{u}_c = \mathbf{r}_c + \mathbf{E}\,\mathbf{y}_c, \tag{2.4.4c}$$

where

$$\mathbf{A}_c(\mathbf{p}_c) = \mathbf{A}_d(\mathbf{p}_c) + \mathbf{B}_d(\mathbf{p}_c)[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}\mathbf{E}\mathbf{C}_d(\mathbf{p}_c), \tag{2.4.5a}$$

$$\mathbf{B}_c(\mathbf{p}_c) = \mathbf{B}_d(\mathbf{p}_c)[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}, \tag{2.4.5b}$$

$$\begin{aligned}\mathbf{C}_c(\mathbf{p}_c) &= \mathbf{C}_d(\mathbf{p}_c) + \mathbf{D}_d(\mathbf{p}_c)[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}\mathbf{E}\mathbf{C}_d(\mathbf{p}_c)\\ &= [\mathbf{I}-\mathbf{D}_d(\mathbf{p}_c)\mathbf{E}]^{-1}\mathbf{C}_d(\mathbf{p}_c),\end{aligned} \tag{2.4.5c}$$

$$\mathbf{D}_c(\mathbf{p}_c) = \mathbf{D}_d(\mathbf{p}_c)[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}. \tag{2.4.5d}$$

Because of Assumption 2.4.1, the matrix $[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}$ is continuously differentiable and

$$\frac{\partial[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}}{\partial \mathbf{p}^i} = [\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}\mathbf{E}\frac{\partial \mathbf{D}_d(\mathbf{p}_c)}{\partial \mathbf{p}^i}[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}. \tag{2.4.6}$$

Therefore, from (2.4.5a,b,c,d), we conclude that the matrices $\mathbf{A}_c(\cdot)$, $\mathbf{B}_c(\cdot)$, $\mathbf{C}_c(\cdot)$ and $\mathbf{D}_c(\cdot)$ are continuously differentiable. The derivatives of $\mathbf{A}_c(\mathbf{p}_c)$, $\mathbf{B}_c(\mathbf{p}_c)$, $\mathbf{C}_c(\mathbf{p}_c)$ and $\mathbf{D}_c(\mathbf{p}_c)$ follow from (2.4.5,a,b,c,d) and (2.4.6) directly.

2.5 A Class of Input Signals

For the purpose of design in the time domain, a set of input signals must be assigned for the components of vector $\mathbf{r}_c$ in (2.4.4a,b,c). It is clear that a wide range of choices of input signals is desirable. However, having a large class of admissible inputs makes the design of an efficient time domain simulator more difficult.

A class of input signals that we will allow in the MIMO system is generated by the following state equations:

$$\dot{\mathbf{x}}_s(t,\mathbf{p}_s) = \mathbf{A}_s(\mathbf{p}_s)\,\mathbf{x}_s(t,\mathbf{p}_s) + \mathbf{B}_s(\mathbf{p}_s)\,\mathbf{r}_s(t,\mathbf{p}_s), \tag{2.5.1a}$$

$$\mathbf{y}_s(t,\mathbf{p}_s) = \mathbf{C}_s(\mathbf{p}_s)\,\mathbf{x}_s(t,\mathbf{p}_s) + \mathbf{D}_s(\mathbf{p}_s)\,\mathbf{r}_s(t,\mathbf{p}_s), \tag{2.5.1b}$$

where the input $\mathbf{r}_s$ is of the form

$$\mathbf{r}_s(t,\mathbf{p}_s) \triangleq \sum_{i=0}^{m} \mathbf{c}_i(\mathbf{p}_s)t^i, \tag{2.5.2}$$

and $\mathbf{p}_s \in \mathbb{R}^{n_{p,s}}$ is a tuning parameter vector or even a design parameter vector if input signals are part of our design requirements, $\mathbf{x}_s(t,\mathbf{p}_s) \in \mathbb{R}^{n_{s,s}}$ is the state. The coefficients in (2.5.2), $\mathbf{c}_i : \mathbb{R}^{n_{p,s}} \to \mathbb{R}^{n_{i,s}}$ are continuously differentiable vectors for all $i \in \{0\} \cup \underline{m}$, $\mathbf{A}_s : \mathbb{R}^{n_{p,s}} \to \mathbb{R}^{n_{s,s} \times n_{s,s}}$, $\mathbf{B}_s : \mathbb{R}^{n_{p,s}} \to \mathbb{R}^{n_{s,s} \times n_{i,s}}$, $\mathbf{C}_s : \mathbb{R}^{n_{p,s}} \to \mathbb{R}^{n_{o,s} \times n_{s,s}}$ and $\mathbf{D}_s : \mathbb{R}^{n_{p,s}} \to \mathbb{R}^{n_{o,s} \times n_{i,s}}$ are continuously differentiable matrices.

For example, step and ramp signals can be generated by assigning $\mathbf{A}_s=0$, $\mathbf{B}_s=0$, $\mathbf{C}_s=0$, $\mathbf{D}_s=1$, and assigning $r_s(t) = 1$ for the step input and $r_s(t) = t$ for the ramp input. A damped sine input signal of the form

$$y_s(t) = g \cdot e^{\sigma t} \cdot \sin(\omega t + \phi), \tag{2.5.3}$$

can be generated by the following state space realization:

$$\dot{\mathbf{x}}_s(t) = \begin{bmatrix} \sigma & \omega \\ -\omega & \sigma \end{bmatrix} \mathbf{x}_s(t), \tag{2.5.4a}$$

$$y_s(t) = [\,0\ -1\,]\,\mathbf{x}_s(t), \tag{2.5.4b}$$

subject to $\mathbf{x}_{s,1}(0) = g \cdot cos(\phi)$ and $\mathbf{x}_{s,2}(0) = -g \cdot \sin(\phi)$. Although the components of the inputs are restricted to polynomial functions only, it is not a great loss since all

continuous functions on a compact interval in t can be uniformly approximated by a suitable order of polynomial function.

Suppose that the matrix $\mathbf{J} \in \mathbb{R}^{n_{i,c} \times n_{s,s}}$ contains information about the connection between the input $\mathbf{r}_c$ of equation (2.4.4c) and the output $\mathbf{y}_s$ of equation (2.5.1b) such that

$$\mathbf{r}_c = \mathbf{J}\,\mathbf{y}_s\,. \tag{2.5.5}$$

Let $\mathbf{r} \triangleq \mathbf{r}_s$. Then, combining equation (2.4.4a,b,c), (2.5.5) and (2.5.1a,b) gives the following state equations for time domain simulation:

$$\dot{\mathbf{x}}(t,\mathbf{p}) = \mathbf{A}(\mathbf{p})\mathbf{x}(t,\mathbf{p}) + \mathbf{B}(\mathbf{p})\mathbf{r}(t,\mathbf{p}), \tag{2.5.6a}$$

$$\mathbf{y}(t,\mathbf{p}) = \mathbf{C}(\mathbf{p})\mathbf{x}(t,\mathbf{p}) + \mathbf{D}(\mathbf{p})\mathbf{r}(t,\mathbf{p}), \tag{2.5.6b}$$

where $\mathbf{p} \triangleq [\mathbf{p}_c^T, \mathbf{p}_s^T]^T$, $\mathbf{x} \triangleq [\mathbf{x}_c^T, \mathbf{x}_s^T]^T$, $\mathbf{y} \triangleq [\mathbf{y}_c^T, \mathbf{y}_s^T]^T$ and state space matrices are

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_c & \mathbf{B}_c\mathbf{J}\mathbf{C}_s \\ 0 & \mathbf{A}_s \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} \mathbf{B}_c\mathbf{J}\mathbf{D}_s \\ \mathbf{B}_s \end{bmatrix}, \tag{2.5.7}$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_c & \mathbf{D}_c\mathbf{J}\mathbf{C}_s \\ 0 & \mathbf{C}_s \end{bmatrix}, \qquad \mathbf{D} = \begin{bmatrix} \mathbf{D}_c\mathbf{J}\mathbf{D}_s \\ \mathbf{D}_s \end{bmatrix}.$$

Let $n \triangleq n_s \triangleq n_{s,c} + n_{s,s}$, $n_i \triangleq n_{i,s}$, $n_o \triangleq n_{o,c} + n_{o,s}$, and $n_p \triangleq n_{p,c} + n_{p,s}$. Now, the interconnected system Σ with polynomial inputs can be shown in Fig. 2.5.

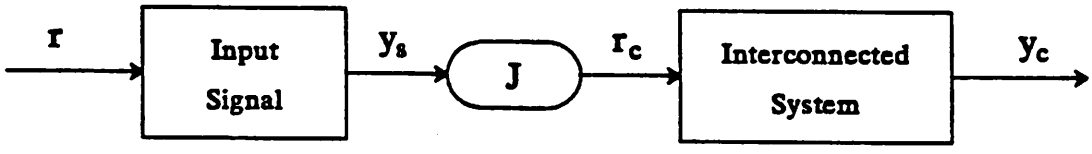


Fig. 2.5. Interconnected System Σ with polynomial inputs $\mathbf{r}$.

Since the matrices $\mathbf{A}_c(\cdot)$, $\mathbf{B}_c(\cdot)$, $\mathbf{C}_c(\cdot)$, $\mathbf{D}_c(\cdot)$, $\mathbf{A}_s(\cdot)$, $\mathbf{B}_s(\cdot)$, $\mathbf{C}_s(\cdot)$ and $\mathbf{D}_s(\cdot)$ are continuously differentiable, it follows from equation (2.5.7) that $\mathbf{A}(\cdot)$, $\mathbf{B}(\cdot)$, $\mathbf{C}(\cdot)$ and $\mathbf{D}(\cdot)$ are continuously differentiable, and hence that $\mathbf{x}(\cdot,\cdot)$ is a C^1 function [Hal.1]. Since $\frac{\partial}{\partial \mathbf{p}^i}\left[\frac{d\mathbf{x}(t,\mathbf{p})}{dt}\right]$ exists and is continuous, the state equations for sensitivity functions can be obtained by differentiating the equations (2.5.6a,b) [Mar.1]:

$$\frac{d}{dt}\left[\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}\right] = \mathbf{A}(\mathbf{p})\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i} + \frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i}\mathbf{x}(t,\mathbf{p}) + \frac{\partial \mathbf{B}(\mathbf{p})}{\partial \mathbf{p}^i}\mathbf{r}(t,\mathbf{p}) \qquad (2.5.8a)$$

$$+ \mathbf{B}(\mathbf{p})\frac{\partial \mathbf{r}(t,\mathbf{p})}{\partial \mathbf{p}^i},$$

$$\frac{\partial \mathbf{y}(t,\mathbf{p})}{\partial \mathbf{p}^i} = \mathbf{C}(\mathbf{p})\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i} + \frac{\partial \mathbf{C}(\mathbf{p})}{\partial \mathbf{p}^i}\mathbf{x}(t,\mathbf{p}) + \frac{\partial \mathbf{D}(\mathbf{p})}{\partial \mathbf{p}^i}\mathbf{r}(t,\mathbf{p}) \qquad (2.5.8b)$$

$$+ \mathbf{D}(\mathbf{p})\frac{\partial \mathbf{r}(t,\mathbf{p})}{\partial \mathbf{p}^i}.$$

Combining equations (2.5.6a,b) and (2.5.8a,b), we obtain the following state equations:

$$\frac{d}{dt}\begin{bmatrix}\mathbf{x}(t,\mathbf{p})\\ \frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}\end{bmatrix} = \begin{bmatrix}\mathbf{A}(\mathbf{p}) & 0\\ \frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{A}(\mathbf{p})\end{bmatrix}\begin{bmatrix}\mathbf{x}(t,\mathbf{p})\\ \frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}\end{bmatrix} + \begin{bmatrix}\mathbf{B}(\mathbf{p}) & 0\\ \frac{\partial \mathbf{B}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{B}(\mathbf{p})\end{bmatrix}\begin{bmatrix}\mathbf{r}(t,\mathbf{p})\\ \frac{\partial \mathbf{r}(t,\mathbf{p})}{\partial \mathbf{p}^i}\end{bmatrix}, \qquad (2.5.9a)$$

$$\begin{bmatrix}\mathbf{y}(t,\mathbf{p})\\ \frac{\partial \mathbf{y}(t,\mathbf{p})}{\partial \mathbf{p}^i}\end{bmatrix} = \begin{bmatrix}\mathbf{C}(\mathbf{p}) & 0\\ \frac{\partial \mathbf{C}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{C}(\mathbf{p})\end{bmatrix}\begin{bmatrix}\mathbf{x}(t,\mathbf{p})\\ \frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}\end{bmatrix} + \begin{bmatrix}\mathbf{D}(\mathbf{p}) & 0\\ \frac{\partial \mathbf{D}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{D}(\mathbf{p})\end{bmatrix}\begin{bmatrix}\mathbf{r}(t,\mathbf{p})\\ \frac{\partial \mathbf{r}(t,\mathbf{p})}{\partial \mathbf{p}^i}\end{bmatrix}. \qquad (2.5.9b)$$

2.6 Concluding Remarks

In this chapter we have defined a general configuration for linear time-invariant multivariable interconnected systems. A set of system equations such as (2.4.4a,b,c) (2.5.5), (2.5.6a,b) and (2.5.9a,b) have been systematically developed. With assumption 2.4.1 and the assumptions that all parameterized entries are C^1 functions, we conclude that all the solutions of the state equations and their sensitivities exist. These system equations will be applied to analysis, simulation and design of the control systems in this thesis.

CHAPTER 3

CONTROL SYSTEM DESIGN VIA SEMI-INFINITE OPTIMIZATION

3.1 Introduction

This chapter discusses design techniques involved in control system design via semi-infinite optimization. In Section 2, we will show how a typical stability requirement for a control system is transcribed into a semi-infinite optimization problem. Next, in Section 3, we will show how frequency domain and time domain specifications of control system designs can be formulated as semi-infinite optimization problems. Finally, in Section 4 we present a discussion of a number of computational issues involved in optimization-based computer-aided design. This leads to the design of the new simulator, *MIMO*, for the class of interconnected control systems $\Sigma(\mathbf{p})$ described in (2.4.4a,b,c).

3.2 Design of Stabilizing Compensators

The most basic requirement in control system design is exponential stability of the closed loop system. The manner in which this requirement is fulfilled depends on the synthesis techniques adopted by the designer. The techniques introduced in this section are based on parametric optimization.

3.2.1 S-Stability

Consider a parameterized, linear, time-invariant, interconnected, finite dimensional dynamical system $\Sigma(\mathbf{p}_c)$, described by the equations (2.4.4a,b,c). We shall denote the characteristic polynomial of $\Sigma(\mathbf{p}_c)$ by $\chi(s,\mathbf{p}_c)$.

When it is desired to ensure not only exponential stability of a closed loop system, but also to exercise some control over the location of its poles, it is convenient to make use of the following definition of S-stability.

Definition 3.2.1 *(S-stability):* Consider the linear, time-invariant, finite dimensional dynamical system Σ of the form (2.4.4a, b, c). Let $\mathbf{S}$ be an open unbounded subset of $\mathbb{C}$ (e.g., as shown in Fig. 3.1) which is symmetrical with respect to the real axis, and such that $\mathbf{S}^c \supset \mathbb{C}_+$, where $\mathbf{S}^c$ is the complement of $\mathbf{S}$ and $\mathbb{C}_+$ is the closed right half of the complex plane.

We say that the system Σ is **S–stable** if all the zeros of its characteristic polynomial are in $\mathbf{S}$.

∎

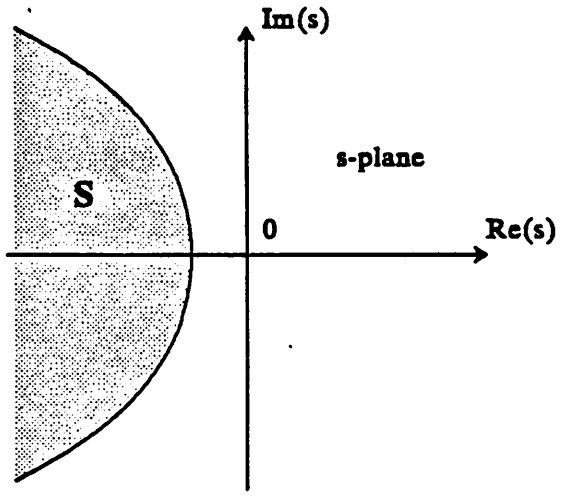


Fig. 3.1. S-stability region

In practice, S-stability can only be ensured with respect to a set $\mathbf{S}$ having a reasonably simple characterization. We shall assume that the set $\mathbf{S}$ has a right boundary $\partial\mathbf{S}$ which is given by an expression of the form

$$\partial\mathbf{S} = \{\, s \in \mathbb{C} \mid s = \sigma + j\omega, \sigma = f(\omega), -\infty < \omega < \infty \,\}, \tag{3.2.1a}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a negative valued, piecewise continuously differentiable function. In this case, the set $\mathbf{S}$ can be described by a simple inequality:

$$\mathbf{S} = \{\, s \in \mathbb{C} \mid s = \sigma + j\omega, \sigma - f(\omega) < 0, -\infty < \omega < \infty \,\}. \tag{3.2.1b}$$

Some typical S regions are shown in Fig. 3.2a,b.

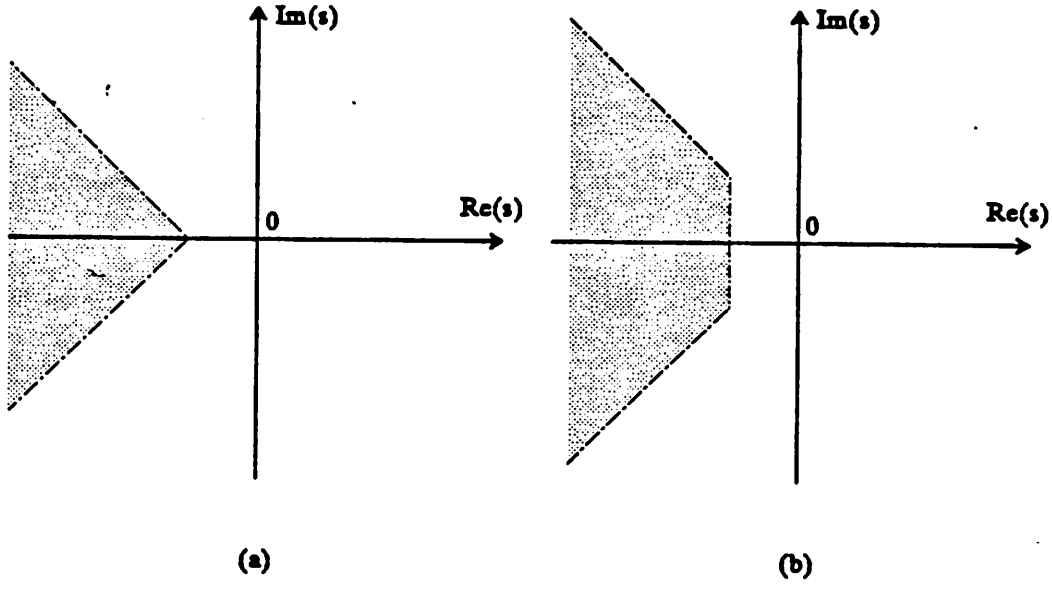


Fig. 3.2. Examples of S-stability region

Let the eigenvalues of $\mathbf{A}_c(\mathbf{p}_c)$ be denoted by $\lambda^j[\mathbf{A}_c(\mathbf{p}_c)]$. Then the simplest way to ensure S-stability for the system $\Sigma(\mathbf{p}_c)$ is to require that

$$\lambda^j[\mathbf{A}_c(\mathbf{p}_c)] \in \mathbf{S}\,, \qquad \forall\ j = 1,2,\cdots,n_{s,c}\,. \tag{3.2.2}$$

where $n_{s,c}$ is the dimension of the closed loop state vector. For example, one may wish to design a control system with a minimum bandwidth, while at the same time ensuring that all closed-loop complex poles have a minimum damping ratio. This can be accomplished by letting the region S be as shown in Fig. 3.2a, i.e., to the left of the boundary defined by

$$\mathrm{Re}(s) = -\kappa\,|\,\mathrm{Im}(s)\,| - \gamma\,. \tag{3.2.3}$$

where Re denotes the real part, Im denotes the imaginary part, and $\kappa, \gamma > 0$. Then the equation (3.2.2) can be transcribed to the following inequalities

$$\mathrm{Re}[\lambda^j[\mathbf{A}_c(\mathbf{p}_c)]] + \kappa\mathrm{Im}[\lambda^j[\mathbf{A}_c(\mathbf{p}_c)]] + \gamma \le 0\,, \quad \forall\ j = 1,2,\cdots,n_{s,c}\,. \tag{3.2.4}$$

Note that in (3.2.4) we are exploiting the fact that complex eigenvalues must come in conjugate pairs.

There are two objections to this approach. The first is that eigenvalues can be extremely sensitive to parameter variations, resulting in severe computational ill conditioning. The second is that eigenvalues are differentiable only when they are distinct [Kat.1, Lan.1]. Now, while one can generally expect closed-loop eigenvalues to be distinct, it is possible in the course of an optimization computation to approach a point of eigenvalue multiplicity with dire effects on the behavior of an optimization algorithm.

3.2.2 Generalized Nyquist Criterion

For very good reasons, the Nyquist stability criterion [Nyq.1], has served for many years as a principal "manual" tool for ensuring stability in linear time-invariant systems. Unfortunately, as was pointed out in [Pol.6], the Nyquist stability criterion cannot be used in conjunction with computer-aided design techniques which make use of semi-infinite optimization. The reason for this is easy to see. In this section, we first generalize the *Nyquist Criterion* so that it can be applied to the interconnected system $\Sigma(\mathbf{p}_c)$, described by the equations (2.4.4a,b,c). Then we explain the reason why the *Generalized Nyquist Criterion* can not be applied.

Lemma 3.2.1 : Suppose that Assumption 2.4.1 holds, then the characteristic polynomial of the interconnected system $\Sigma(\mathbf{p}_c)$ described by (2.4.4a,b,c) is given by:

$$\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)] = \prod_{i=1}^{\mu} \det[s\mathbf{I}-\mathbf{A}_i(p_i)] \cdot \frac{\det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(s,\mathbf{p}_c)]}{\det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(\infty,\mathbf{p}_c)]}, \tag{3.2.5a}$$

or

$$\frac{\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]}{\prod_{i=1}^{\mu} \det[s\mathbf{I}-\mathbf{A}_i(\mathbf{p}_i)]} = \frac{\det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(s,\mathbf{p}_c)]}{\det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(\infty,\mathbf{p}_c)]}, \tag{3.2.5b}$$

where $\hat{\mathbf{G}}_d(\cdot,\cdot)$ is defined in (2.4.2) and (2.3.2).

Proof: From (2.4.5a), we have that

$$\begin{aligned}
\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)] &= \det\{s\mathbf{I}-\mathbf{A}_d(\mathbf{p}_c)-\mathbf{B}_d(\mathbf{p}_c)[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}\mathbf{E}\mathbf{C}_d(\mathbf{p}_c)\} \\
&= \det[s\mathbf{I}-\mathbf{A}_d(\mathbf{p}_c)]\cdot\det\{I-[s\mathbf{I}-\mathbf{A}_d(\mathbf{p}_c)]^{-1}\mathbf{B}_d(\mathbf{p}_c)[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}\mathbf{E}\mathbf{C}_d(\mathbf{p}_c)\} \\
&= \det[s\mathbf{I}-\mathbf{A}_d(\mathbf{p}_c)]\cdot\det\{\mathbf{I}-\mathbf{E}\mathbf{C}_d(\mathbf{p}_c)[s\mathbf{I}-\mathbf{A}_d(\mathbf{p}_c)]^{-1}\mathbf{B}_d(\mathbf{p}_c)[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]^{-1}\} \\
&= \prod_{i=1}^{\mu}\det[s\mathbf{I}-\mathbf{A}_i(\mathbf{p}_i)]\cdot\frac{\det\{\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)-\mathbf{E}\mathbf{C}_d(\mathbf{p}_c)[s\mathbf{I}-\mathbf{A}_d(\mathbf{p}_c)]^{-1}\mathbf{B}_d(\mathbf{p}_c)\}}{\det[\mathbf{I}-\mathbf{E}\mathbf{D}_d(\mathbf{p}_c)]} \\
&= \prod_{i=1}^{\mu}\det[s\mathbf{I}-\mathbf{A}_i(\mathbf{p}_i)]\cdot\frac{\det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(s,\mathbf{p}_c)]}{\det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(\infty,\mathbf{p}_c)]} \qquad (3.2.6)
\end{aligned}$$

∎

Suppose that the stability region S is defined as in (3.2.1a,b). An indented boundary of S, $\partial\widetilde{\mathbf{S}}$, is needed for the S-stability Nyquist criterion. If $\det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(s,\mathbf{p}_c)]$ has poles on the boundary of S, then the boundary ∂S is modified by making a semicircular "detour" of a very small radius to the left of each such pole, as shown in Fig. 3.3.

The following result is obvious in view of the ordinary Nyquist stability criterion [Nyq.1, Cal.1, Des.1].

Theorem 3.2.2 (*Generalized Nyquist Criterion*): Consider the interconnected system $\Sigma(\mathbf{p}_c)$ described in the equations (2.4.4a,b,c), and suppose that Assumption 2.4.1 holds. Suppose that the polynomial $s \to \prod_{i=1}^{\mu}\det[s\mathbf{I}-\mathbf{A}_i(\mathbf{p}_i)]$, counting multiplicities, has m zeros in $\mathbf{S}^e$. Then, the system Σ is S-stable if and only if the locus of $\det[\mathbf{I}-\mathbf{E}\hat{\mathbf{G}}_d(s,\mathbf{p}_c)]$, traced out for s $(=\sigma+j\omega)$ on the indented boundary $\partial\widetilde{\mathbf{S}}$ and ω taking the values from $-\infty$ to ∞, does not go through the origin but encircles the origin m times in the counterclockwise sense.

Proof : The proof follows directly from (3.2.5a,b).

∎

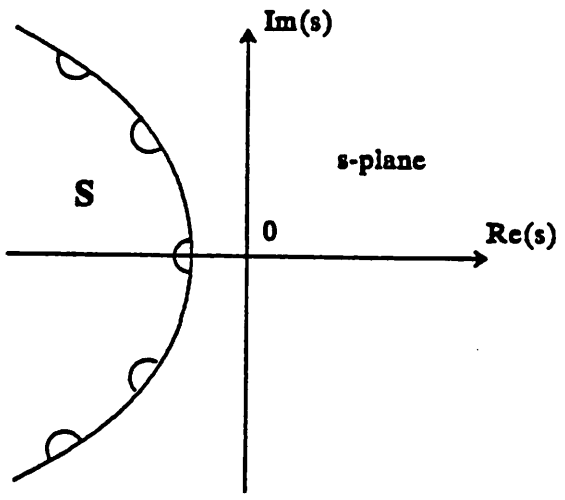


Fig. 3.3. Indented boundary of S region

Corollary 3.2.3 : Let $\chi(s,\mathbf{p}_c)=\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ be the characteristic polynomial of the interconnected system $\Sigma(\mathbf{p}_c)$ and let $d(\mathbf{p}_c,s) \triangleq \prod_{i=1}^{\mu}\det[s\mathbf{I}-\mathbf{A}_i(\mathbf{p}_i)]$. Let $m(\mathbf{p}_c)$ be the number of $\mathbf{S}^c$ zeros of $d(s,\mathbf{p}_c)$ and let

$$N(\mathbf{p}_c) \triangleq \{\lim_{\omega\to\infty}\arg[\chi(j\omega,\mathbf{p}_c)/d(j\omega,\mathbf{p}_c)]-\arg[\chi(0,\mathbf{p}_c)/d(0,\mathbf{p}_c)]\}/2\pi-m(\mathbf{p}_c) \quad (3.2.7)$$

Then, $\chi(s,\mathbf{p}_c)$ has no zeros in $\mathbf{S}^c$ if and only if $N(\mathbf{p}_c)=0$.

∎

It has been pointed out in [Pol.6] that one major objection to using the *Generalized Nyquist Criterion* on a digital computer stems from the fact that the function $N(\mathbf{p}_c)$ is integer valued. Therefore, equation (3.2.7) cannot be used as a constraint in an optimization problem solvable either by nonlinear programming or semi-infinite optimization algorithms.

A first attempt to modify the Nyquist stability criterion into a form compatible with optimization-based control system design was proposed by Polak in [Pol.6]. In particular, it was shown in [Pol.6] that if $\mathcal{A}(s)$ is a polynomial of the same degree as

$\chi(s,\mathbf{p}_c)$, whose zeros are all in $\mathring{\mathbb{C}}_-$, then $\chi(s,\mathbf{p}_c)$ has no $\mathbb{C}_+$ zeros if and only if the Nyquist locus of $T(j\omega,\mathbf{p}_c) \triangleq \chi(j\omega,\mathbf{p}_c)/\bar{d}(j\omega)$, traced out for $\omega \in \partial S$ does not encircle the origin. It was shown in [Pol.6] that a *sufficient condition* for this to hold can be expressed in the form (by no means unique)

$$\mathrm{Im}[T(j\omega,\mathbf{p}_c)] - c_1 \mathrm{Re}[T(j\omega,\mathbf{p}_c)]^2 + c_2 \leq 0, \quad \forall \ \omega \in (-\infty,\infty), \tag{3.2.8}$$

where $c_1, c_2 > 0$.

The modified Nyquist stability criterion in [Pol.6] satisfies the requirements of semi-infinite optimization. However, it does suffer from four drawbacks: (i) it is only a sufficient condition, which at times can be very conservative, (ii) it involves an arbitrary polynomial $\bar{d}(s)$ whose definition requires judgement, (iii) the selection of a form such (3.2.8) involves some skill, e.g., in some cases it may be advantageous to use the inequality

$$-c_1 \mathrm{Im}[T(j\omega,\mathbf{p}_c)]^2 - \mathrm{Re}[T(j\omega,\mathbf{p}_c)] + c_2 \leq 0, \quad \forall \ \omega \in [0,\infty), \tag{3.2.9}$$

where $c_1, c_2 > 0$, instead of (3.2.8), and (iv) it proved to be sensitive to the frequency discretization step size.

3.2.3 A Modified Nyquist Criterion

In this subsection, we present a new computational stability criterion which can be viewed as a *modified* Nyquist stability criterion and which is both a *necessary and sufficient* condition of stability. The new stability criterion is compatible with semi-infinite optimization requirements and does not suffer from any of the drawbacks of the computational stability criterion presented in [Pol.6].

We shall denote by $\mathbb{C}[s]$ the ring of polynomials in $s \in \mathbb{C}$ with coefficients in $\mathbb{C}$, by $\mathbb{R}[s]$ the ring of polynomials in $s \in \mathbb{C}$ with coefficients in $\mathbb{R}$, and by P_N the set of monic polynomials of degree N contained in $\mathbb{R}[s]$. Furthermore, for any polynomial $\chi(s)$ in $\mathbb{C}[s]$ or in $\mathbb{R}[s]$, we shall denote by $Z[\chi(s)]$ the set of zeros of $\chi(s)$. Now, the new modified Nyquist stability criterion can be stated as follows:

Theorem 3.2.4 *(Modified Nyquist Stability Criterion):* Let $\mathbf{S} \subset \mathbb{C}$ be as specified in Definition 3.2.1 and let $\mathbf{B} \subset \mathbb{C}$ be any simply connected set satisfying $0+0j \notin \mathbf{B}$. Suppose that $D(s,\mathbf{q}) \in \mathbb{C}[s]$ is a parameterized polynomial of degree N, whose coefficients depend on the parameter vector $\mathbf{q} \in \mathbb{R}^{n_D}$ in such a way that for every $\chi(s) \in P_N$ satisfying $Z[\chi(s)] \subset \mathbf{S}$, there exists a $\mathbf{q}_\chi \in \mathbb{R}^{n_D}$ such that

$$(i) \quad Z[D(s,\mathbf{q}_\chi)] \subset \mathbf{S}, \tag{3.2.10a}$$

$$(ii) \quad \chi(s)/D(s,\mathbf{q}_\chi) \in \mathbf{B}, \ \forall \ s \in \partial\mathbf{S}. \tag{3.2.10b}$$

Then, given a polynomial $\chi(s) \in P_N$, $Z[\chi(s)] \subset \mathbf{S}$ if and only if there exists a $\mathbf{q}_\chi \in \mathbb{R}^{n_D}$ such that (3.2.10a,b) hold.

Proof : (=>) Suppose that $Z[\chi(s)] \subset \mathbf{S}$. Then, by assumption, there exists a $\mathbf{q}_\chi \in \mathbb{R}^{n_D}$ such that (3.2.10a), (3.2.10b) hold.

(<=) Next, suppose that (3.2.10a), (3.2.10b) hold. Then, because $\mathbf{B}$ is a simply connected set which does not contain the origin, the locus traced out in the complex plane by $\chi(s)/D(s,\mathbf{q}_\chi)$, for $s \in \partial\mathbf{S}$, does not encircle the origin. It now follows from (3.2.10a) and the Argument Principle [Mar.2] that $Z[\chi(s)] \subset \mathbf{S}$.

∎

It is clear from Theorem 3.2.4 that an acceptable parameterization of the polynomial $D(s,\mathbf{q})$ depends on the shape of the set $\mathbf{S}$ and the choice of the set $\mathbf{B}$. A further requirement is imposed by semi-infinite optimization: the parameterization must be such that it is easy to ensure that the zeros of $D(s,\mathbf{q})$ are in $\mathbf{S}$. We shall see in the next section that the selection of a parameterization of $D(s,\mathbf{q})$ and of the set $\mathbf{B}$ can be fairly easy.

3.2.4 Parameterization of the Normalizing Polynomial $D(s,\mathbf{q})$

Because it leads to the simplest semi-infinite inequalities, we shall set $\mathbf{B} = \mathring{\mathbb{C}}_+$ (the open right half plane), and we shall give a few examples of parameterizations of the normalizing polynomial $D(s,\mathbf{q})$. The simplest parameterization of $D(s,\mathbf{q})$, which can be used with *any* set $\mathbf{S} \subset \mathring{\mathbb{C}}_-$, is

$$D(s,\mathbf{q}) = a_0 + a_1 s^1 + \cdots + a_{N-1}s^{N-1} + s^N, \tag{3.2.11}$$

where N is the degree of $D(s,\mathbf{q})$, and $\mathbf{q} \triangleq [a_0, a_1, \cdots, a_{N-1}]^T \in \mathbb{R}^N$. Since for any characteristic polynomial $\chi(s) \in P_N$ such that $Z[\chi(s)] \subset \mathbf{S}$, we can choose a coefficients vector $\mathbf{q}_\chi$ for $D(s,\mathbf{q})$ in (3.2.11), such that $D(s,\mathbf{q}_\chi) \equiv \chi(s)$ and since $1+0j \in \mathring{\mathbb{C}}_+$, it is clear that (3.2.10a), (3.2.10b) are satisfied by this parameterization for any set $\mathbf{S} \subset \mathring{\mathbb{C}}_-$ and $\mathbf{B} = \mathring{\mathbb{C}}_+$.

Unfortunately, from an optimization point of view, the parameterization (3.2.11) is not at all satisfactory for two reasons. The first is that the zeros of $D(s,\mathbf{q})$ may turn out to be unacceptably sensitive to variations in the parameter $\mathbf{q}$. The second one is that, for a given set $\mathbf{S}$, there appears to be no simple way of ensuring that $Z[D(s,\mathbf{q})] \subset \mathbf{S}$. In fact, with the parameterization in (3.2.11), finding a $\mathbf{q} \in \mathbb{R}^{n_D}$, such that $Z[D(s,\mathbf{q})] \subset \mathbf{S}$ is almost as difficult as placing the zeros of a parameterized characteristic polynomial $\chi(s,\mathbf{p})$ in $\mathbf{S}$.

A much better, general purpose parameterization of $D(s,\mathbf{q})$, for $\mathbf{B} = \mathring{\mathbb{C}}_+$ and any $\mathbf{S} \subset \mathring{\mathbb{C}}_-$, is the following one:

$$D(s,\mathbf{q}) = \prod_{i=1}^{N} (s - \alpha_i - j\beta_i), \tag{3.2.12}$$

where N is the degree of $D(s,\mathbf{q})$ and $\mathbf{q} \triangleq [\alpha_1, \alpha_2, \cdots, \alpha_N, \beta_1, \beta_2, \cdots, \beta_N]^T \in \mathbb{R}^{2N}$.

Suppose that $\mathbf{S}$ is defined as in (3.2.1b), that $\mathbf{B} = \mathring{\mathbb{C}}_+$ and that $D(s,\mathbf{q})$ is parameterized as in (3.2.12). Referring to the requirements in Theorem 3.2.4, we see that given any $\chi(s) \in P_N$ such that $Z[\chi(s)] \subset \mathbf{S}$, the parameterization (3.2.12) allows us to choose a $\mathbf{q}_\chi \in \mathbb{R}^{2N}$ such that $D(s,q_\chi) \equiv \chi(s)$. Hence (3.2.10a) is satisfied. Next, since $\chi(s)/D(s,\mathbf{q}_\chi) \equiv 1$ and $1+0j \in \mathring{\mathbb{C}}_+$, we see that (3.2.10b) is satisfied. It therefore follows from Theorem 3.2.10b that a stabilizing parameter $\mathbf{p}_S \in \mathbb{R}^{n_\Sigma}$ for the parameterized system $\Sigma(\mathbf{p})$ can be obtained by solving the following set of ordinary and semi-infinite inequalities for a $\mathbf{p}_S \in \mathbb{R}^{n_\Sigma}$ and a $\mathbf{q}_S \in \mathbb{R}^{n_D}$:

$$\alpha_i - f(\beta_i) + \epsilon \leq 0, \qquad \text{for } i = 1, 2, \cdots, N, \tag{3.2.13a}$$

$$\chi(f(\omega)+j\omega)/D(f(\omega)+j\omega,\mathbf{q})-\epsilon \geq 0, \quad \forall\ \omega\in(-\infty,\infty), \tag{3.2.13b}$$

for some $\epsilon > 0$. Current semi-infinite optimization algorithms have no difficulty solving this system of inequalities.

Note that as defined in (3.2.12), $D(s,\mathbf{q})$ is a polynomial with complex coefficients whose zeros need not occur in complex conjugate pairs, and that $n_D = 2N$. There are a number of cases where one can use a parameterization of $D(s,\mathbf{q})$ with fewer than $2N$ parameters. We shall give two examples.

Example 3.2.1 : Consider the case shown in Fig. 3.2a, where S is defined as a sector with damping angle $\Theta\in(0,\pi/2)$ and $\gamma=0$, i.e.,

$$\mathbf{S} \triangleq \{\, s=(\sigma+j\omega)\in\ \mathbb{C} \mid \sigma+|\omega/\tan\Theta| < 0 \,\}, \tag{3.2.14}$$

Before giving a parameterization for $D(s,\mathbf{q})$ of arbitrary order, let us examine linear and quadratic polynomials. We see that when $a,\ b\in\mathbb{R}$, $Z[(s+a)]\subset\mathbf{S}$ if and only if

$$a > 0, \tag{3.2.15a}$$

and that $Z[(s^2+as+b)]\subset\mathbf{S}$ if and only if

$$a > 0, \tag{3.2.15b}$$

$$b > 0, \tag{3.2.15c}$$

$$a^2-4b\cos^2\Theta > 0. \tag{3.2.15d}$$

To take advantage of this observation, when the degree of $D(s,\mathbf{q})$ is even, we set

$$D(s,\mathbf{q}) \triangleq \prod_{i=1}^{n}(s^2+a_is+b_i), \tag{3.2.16a}$$

where $\mathbf{q}\triangleq[a_1,a_2,\cdots,a_n,b_1,b_2,\cdots,b_n]^T\in\mathbb{R}^{2n}$, and $n_D = N = 2n$. When the degree of $D(s,\mathbf{q})$ is odd, we set

$$D(s,\mathbf{q}) \triangleq (s+a_0)\prod_{i=1}^{n}(s^2+a_is+b_i), \tag{3.2.16b}$$

where $\mathbf{q}\triangleq[a_0,a_1,a_2,\cdots,a_n,b_1,b_2,\cdots,b_n]^T\in\mathbb{R}^{2n+1}$, and $n_D = N = 2n+1$.

Hence, when the polynomial $D(s,\mathbf{q})$ is expressed as a product of linear and quadratic polynomials, as in (3.2.16a) or (3.2.16b), it is very easy to ensure that its zeros are in the set $\mathbf{S}$ (defined in (3.2.14)).

Next we return to the requirements of Theorem 3.2.4. Since characteristic polynomials have real coefficients, their zeros occur in complex conjugate pairs. Therefore, given any $\chi(s) \in P_N$, such that $Z[\chi(s)] \subset \mathbf{S}$, there exists a parameter vector $\mathbf{q}_\chi$ such that $D(s,\mathbf{q}_\chi) \equiv \chi(s)$ (with $D(s,\mathbf{q})$ defined as in (3.2.16a) or (3.2.16b), as appropriate). Therefore, if we set $\mathbf{B} = \mathring{\mathbb{C}}_+$, we see that (3.2.10a), (3.2.10b) are satisfied. It therefore follows from Theorem 3.2.4 that an S-stabilizing parameter $\mathbf{p}_S$ for the system $\Sigma(\mathbf{p})$ can be obtained by solving the following set of inequalities for a $\mathbf{p}_S \in \mathbb{R}^{n_\Sigma}$ and a $\mathbf{q}_S \in \mathbb{R}^N$:

$$\mathbf{q}^i - \epsilon \geq 0, \qquad \text{for } i = 1,2,\cdots,N, \tag{3.2.17a}$$

$$a_i^2 - 4b_i\cos^2\Theta - \epsilon \geq 0, \qquad \text{for } i = 1,2,\cdots,n, \tag{3.2.17b}$$

$$\mathrm{Re}[\chi(re^{j(\pi-\Theta)},\mathbf{p})/D(re^{j(\pi-\Theta)},\mathbf{q})] - \epsilon \geq 0, \qquad \forall\ r \in [0,\infty), \tag{3.2.17c}$$

for some $\epsilon > 0$ where $\mathbf{q}^i$ is the ith component of the vector $\mathbf{q}$. Note that because of symmetry, if (3.2.17c) is satisfied, then it is also satisfied when Θ is replaced by $-\Theta$.

As a trivial corollary of the above, we find that when $\mathbf{S} = \mathring{\mathbb{C}}_-$, we can compute a $\mathbf{p}_S \in \mathbb{R}^{n_\Sigma}$ such that $\Sigma(\mathbf{p}_S)$ is exponentially stable, by solving the following set of inequalities for a $\mathbf{p}_S \in \mathbb{R}^{n_\Sigma}$ and a $\mathbf{q}_S \in \mathbb{R}^N$:

$$\mathbf{q}^i - \epsilon \geq 0, \qquad \text{for } i = 1,2,\cdots,N, \tag{3.2.18a}$$

$$\mathrm{Re}[\chi(j\omega,\mathbf{p})/D(j\omega,\mathbf{q})] - \epsilon \geq 0, \qquad \forall\ \omega \in [0,\infty). \tag{3.2.18b}$$

∎

Example 3.2.2 : We saw in Example 3.2.1 that the parameterization (3.2.16a), (3.2.16b) has the property that for any $\chi(s) \in P_N$ there exists a $\mathbf{q}_\chi \in \mathbb{R}^{n_D}$ such that $D(s,\mathbf{q}_\chi) \equiv \chi(s)$. Hence it satisfies (3.2.10a), (3.2.10b) for any "stability" set $\mathbf{S} \subset \mathring{\mathbb{C}}_-$ (defined as in Definition 3.2.1) and any simply connected set $\mathbf{B} \subset \mathbb{C}$ such that $0+0j \notin \mathbf{B}$ and $1+0j \in \mathbf{B}$. However, as a practical matter, it can only be used for

ensuring S -stability in the case where one can construct simple inequalities for ensuring that $Z[D(s,\mathbf{q})] \subset \mathbf{S}$. In Example 3.2.1, we gave such a case. We shall now show that the parameterization (3.2.16a), (3.2.16b) can be used with a whole family of sets S.

Consider the case where

$$\mathbf{S} = \{ s = (\sigma + j\omega) \in \mathbb{C} \mid \sigma < f(\omega),\ \omega \in (-\infty, \infty) \} \tag{3.2.19a}$$

with

$$f(\omega) = g(\omega^2) = \min_{j \in \underline{m}} g^j(\omega^2). \tag{3.2.19b}$$

We make three assumptions for all $j \in \underline{m}$: (i) $g^j(\cdot)$ are continuous, negative valued functions, (ii) $g^j(\omega^2) \leq g(0)$ for all $\omega \in \mathbb{R}$, and (iii) $g^j(-\omega^2) \geq g(0)$ for all $\omega \in \mathbb{R}$.

The following three examples illustrate the kind of sets S can one define within the above framework:

$$g^i(z) = -\alpha, \qquad \forall\ z \in \mathbb{R}; \tag{3.2.20a}$$

$$g^j(z) = -z - \alpha, \qquad \forall\ z \in \mathbb{R}; \tag{3.2.20b}$$

$$g^k(z) = \begin{cases} -\beta\sqrt{z} + \gamma_1, & \forall\ z > \delta \\ -\alpha z + \gamma_2, & \forall\ |z| \leq \delta \\ \beta\sqrt{-z} + \gamma_1, & \forall\ z < -\delta \end{cases} \tag{3.2.20c}$$

To set up the inequalities for ensuring that $Z[D(s,\mathbf{q})] \subset \mathbf{S}$, we again consider linear and quadratic factors. Clearly $-a$ is the only zero of $(s+a)$. Hence, $Z[(s+a)] \subset \mathbf{S}$ if and only if

$$-a < g(0). \tag{3.2.21a}$$

Next, the zeros of $(s^2 + as + b)$ are $s_1 = -a/2 + \sqrt{a^2/4 - b}$, and $s_2 = -a/2 - \sqrt{a^2/4 - b}$. Hence $Z[(s^2 + as + b)] \subset \mathbf{S}$ if and only if

$$a^2/4 - b \leq 0 \ \text{ and } \ -a/2 < g(b - a^2/4) \tag{3.2.21b}$$

or

$$a^2/4 - b > 0 \ \text{ and } \ -a/2 \pm \sqrt{a^2/4 - b} < g(0). \tag{3.2.21c}$$

The relations (3.2.21b), (3.2.21c) cannot be used in conjunction with a semi-infinite optimization algorithm. Therefore we propose to replace them with the following set of three inequalities which we shall show to be equivalent to (3.2.21b), (3.2.21c):

$$-a/2-g(0)<0 \tag{3.2.22a}$$

$$-a/2-g(b-a^2/4)<0 \tag{3.2.22b}$$

$$(a^2/4-b)-[g(0)+a/2]^2<0\,. \tag{3.2.22c}$$

Proposition 3.2.5 : The systems of inequalities (3.2.21b), (3.2.21c) and (3.2.22a), (3.2.22b), (3.2.22c) are equivalent.

Proof: (=>) Suppose that $s_1=-a/2+\sqrt{a^2/4-b}$ satisfies either (3.2.21b) or (3.2.21c) (This ensures that s_2 satisfies either (3.2.21b) or (3.2.21c) as well). Suppose that (3.2.21b) holds. Then (3.2.22b) and (3.2.22c) hold automatically, while (3.2.22a) holds because $g(0)\geq g(b-a^2/4)$. Next suppose that (3.2.21c) holds. Then (3.2.22a) and (3.2.22c) holds automatically. By assumption, because $a^2/4-b>0$, we have that $g(b-a^2/4)\geq g(0)$. Hence $g(b-a^2/4)+a/2\geq g(0)+a/2\geq 0$, which shows that (3.2.22b) holds.

(<=) Suppose that $s_1=-a/2+\sqrt{a^2/4-b}$ satisfies (3.2.22a) - (3.2.22c). If $a^2/4-b\leq 0$, then (3.2.22b) implies that (3.2.21b) is satisfied. If $a^2/4-b>0$, then (3.2.22a), (3.2.22c) imply that (3.2.21c) is satisfied. Hence the two systems are equivalent.

We conclude that if we set $\mathbf{B}=\mathring{\mathbb{C}}_+$, then a stabilizing parameter $\mathbf{p}_S$ for the system $\Sigma(p)$ can be obtained by solving the following set of inequalities for a $\mathbf{p}_S\in\mathbb{R}^{n_\Sigma}$ and a $\mathbf{q}_S\in\mathbb{R}^N$ (we state the inequalities for the case where N is odd) :

$$-a_0-g(0)+\epsilon\leq 0, \tag{3.2.23a}$$

$$-a_i/2-g(0)+\epsilon\leq 0, \qquad \text{for } i=1,2,\cdots,n, \tag{3.2.23b}$$

$$-a_i/2-g(b_i-a_i^2/4)+\epsilon\leq 0, \qquad \text{for } i=1,2,\cdots,n, \tag{3.2.23c}$$

$$(a_i^2/4-b_i)-[g(0)+a_i/2]^2+\epsilon\leq 0; \quad \text{for } i=1,2,\cdots,n, \tag{3.2.23d}$$

$$\mathrm{Re}[\chi(f(\omega)+j\omega),\mathbf{p})/D(f(\omega)+j\omega),\mathbf{q})]-\epsilon\geq 0\,, \qquad \forall\ \omega\in[0,\infty), \tag{3.2.23e}$$

for some $\epsilon > 0$. When N is even, the constraint (3.2.23a) is dropped.

∎

Example 3.2.3: Consider the case shown in Fig. 3.2b, where $\mathbf{S}$ is defined as an intersection of a sector $\mathbf{S}_1$ with damping angle $\Theta \in (0, \pi/2)$ and a half plane $\mathbf{S}_2$ with decay rate $-\gamma \leq 0$, i.e.,

$$\mathbf{S}_1 \triangleq \{ (\sigma + j\omega) \in \mathbb{C} \mid \sigma + |\omega/\tan\Theta| < 0 \}, \tag{3.2.24a}$$

$$\mathbf{S}_2 \triangleq \{ (\sigma + j\omega) \in \mathbb{C} \mid \sigma < -\gamma \}, \tag{3.2.24b}$$

$$\mathbf{S} \triangleq \{ (\sigma + j\omega) \in \mathbb{C} \mid \sigma + |\omega/\tan\Theta| < 0 \, , \sigma < -\gamma \} = \mathbf{S}_1 \cap \mathbf{S}_2 \, . \tag{3.2.24c}$$

Referring to the results of Example 3.2.2, let $g(z) = -\gamma$, $\forall\, z \in \mathbb{R}$, then we see that $Z[(s+a)] \subset \mathbf{S}_2$ if and only if

$$a > \gamma, \tag{3.2.25a}$$

and that $Z[(s^2 + as + b)] \subset \mathbf{S}_2$ if and only if

$$a > 2\gamma, \tag{3.2.25b}$$

$$b > a\gamma - \gamma^2, \tag{3.2.25c}$$

Since $\mathbf{S} = \mathbf{S}_1 \cap \mathbf{S}_2$, it follows from the equations (3.2.25a,b,c) and (3.2.15a,b,c,d) that $Z[(s+a)] \subset \mathbf{S}$ if and only if

$$a > \gamma, \tag{3.2.26a}$$

and that $Z[(s^2 + as + b)] \subset \mathbf{S}$ if and only if

$$a > 2\gamma, \tag{3.2.26b}$$

$$b > a\gamma - \gamma^2, \tag{3.2.26c}$$

$$a^2 - 4b\cos^2\Theta > 0. \tag{3.2.26d}$$

If $D(s, \mathbf{q})$ is defined as in (3.2.16a,b) and $\mathbf{B} = \mathring{\mathbb{C}}_+$, then a stabilizing parameter $\mathbf{p}_S$ for the system $\Sigma(\mathbf{p})$ can be obtained by solving the following set of inequalities for a $\mathbf{p}_S \in \mathbb{R}^{n_\Sigma}$ and a $\mathbf{q}_S \in \mathbb{R}^N$:

$$a_0 - \epsilon \geq \gamma, \qquad \text{if the degree of } D(s,\mathbf{q}) \text{ is odd}, \tag{3.2.27a}$$

$$a_i - \epsilon \geq 2\gamma, \qquad \text{for } i = 1, 2, \cdots, n, \tag{3.2.27b}$$

$$b_i - \epsilon \geq a_i\gamma - \gamma^2, \qquad \text{for } i = 1, 2, \cdots, n, \tag{3.2.27c}$$

$$a_i^2 - 4b_i\cos^2\Theta - \epsilon \geq 0, \qquad \text{for } i = 1, 2, \cdots, n, \tag{3.2.27d}$$

$$-\mathrm{Re}[\chi(s, \mathbf{p})/D(s, \mathbf{q})] - \epsilon \geq 0, \qquad \forall \ s \in \partial S, \omega > 0, \tag{3.2.27e}$$

for some $\epsilon > 0$. Let $T(s, \mathbf{p}, \mathbf{q}) \triangleq \chi(s, \mathbf{p})/D(s, \mathbf{q})$, then a typical modified Nyquist diagram is shown in Fig.3.4.

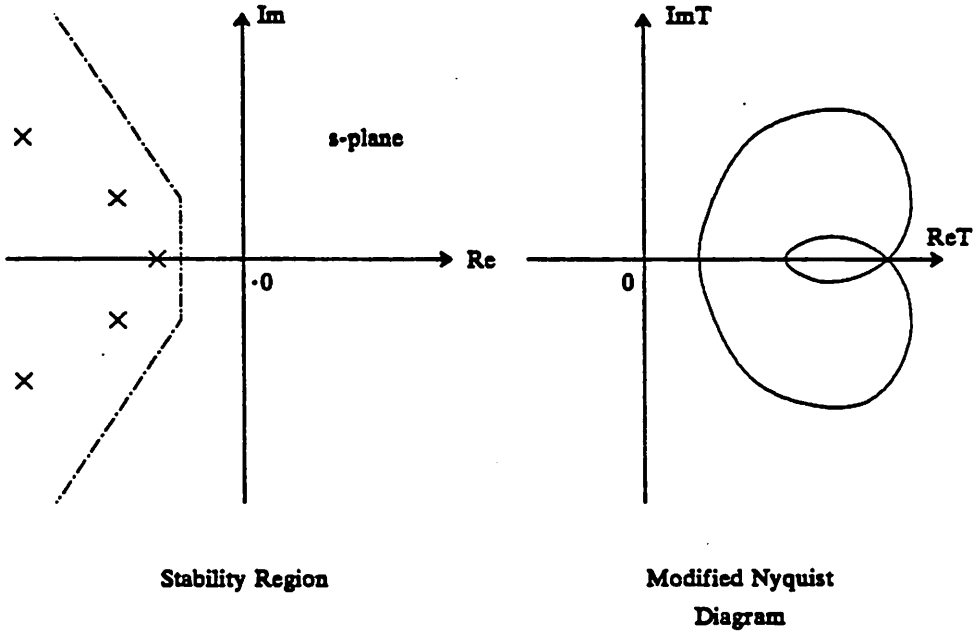
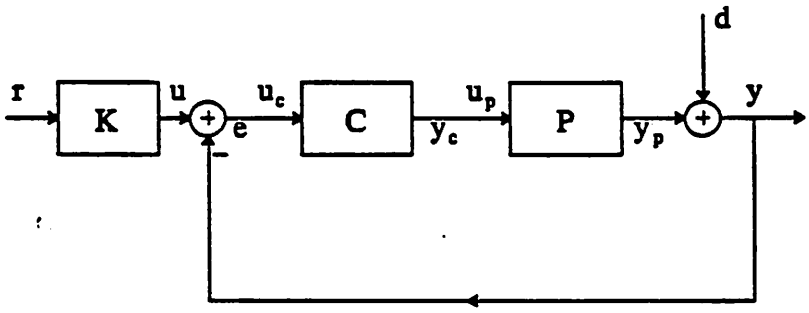


Fig. 3.4. Modified Nyquist diagram

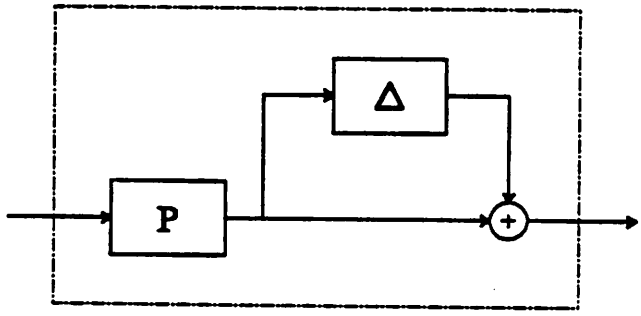
3.3 Design Specifications

We shall illustrate the process of transcribing control system design specifications into semi-infinite inequalities by considering a typical example: the design of a two-degree-of-freedom control system of the form shown in Fig.3.5a.

We assume that the plant P, compensator C and pre-compensator K are linear, finite-dimensional systems described in the form of (2.3.1a,b). The free parameters of



(a)



(b)

Fig. 3.5. Two-degree-of-freedom control system

the state equations of the compensators make up the design vector $\mathbf{p} \in \mathbb{R}^n$. We shall consider time and frequency domain specifications both on the nominal closed-loop system as well as on the family of closed-loop systems resulting from multiplicative perturbations of the plant as shown in Fig.3.5b. Physical constraints on design parameters also will be taken into consideration. Then, we shall generalize the results of this special case to the general interconnected system $\Sigma(\mathbf{p}_c)$ described in (2.4.4a,b,c).

In formulating design specifications as discrete or semi-infinite inequalities one must take care to ensure that the resulting functions satisfy certain minimum continuity requirements and that the implied computations are reasonably well conditioned. In particular, in order to apply current algorithms or develop rigorously

algorithms for optimization problems involving nondifferentiable functions, it is necessary that these functions be at least locally Lipschitz continuous [Cla.1].

3.3.1 Frequency Domain Specifications

In the design of a SISO control system, specifications on quantities such as phase and gain margins in the frequency domain are commonplace. With the development of modern MIMO control system design theory, these SISO frequency domain specifications have been generalized in terms of norm (e.g. largest singular value, or H^∞ norm) bounds on various transfer function matrices.

- *Stability Robustness:*

First, we turn to the problem of ensuring closed-loop system stability in the presence of unstructured plant uncertainty. Referring to [Che.2, Doy.1, San.1], we see that if the nominal design for the plant is bounded-input-bounded-output (BIBO) stable, then the closed-loop system will remain BIBO stable for all perturbed plants $\tilde{P}$ as in Fig.3.5b with $\Delta(s)$ satisfying

$$\bar{\sigma}[\Delta(j\omega)] \leq b(\omega), \qquad \forall\, \omega \geq 0\,. \tag{3.3.1a}$$

if and only if the nominal feedback system satisfies

$$\bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{yu}}(j\omega,\mathbf{p})] < 1/b(\omega), \qquad \forall\, \omega \in [\omega',\omega''], \tag{3.3.1b}$$

where $[\omega',\omega'']$ is a critical range of frequencies. The constraint is as shown in Fig.3.6a.

- *Disturbance Suppression and Good Command Tracking:*

The effect of the output disturbance on the output of the nominal closed-loop system can be suppressed by making the closed-loop system disturbance-to-output transfer function matrix $\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})$ "small" in a critical frequency range. This is usually accomplished by imposing bounds on the maximum singular value of the transfer function matrix $\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})$. Typically, these bounds have the form

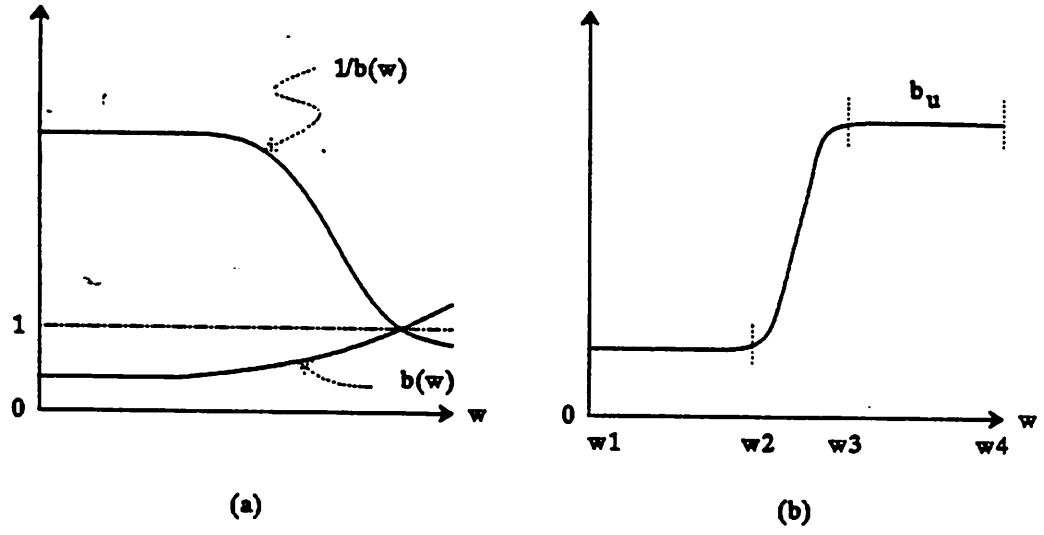


Fig. 3.6. Constraints in frequency domain

$$\bar{\sigma}[\hat{\mathbf{H}}_{yd}(j\omega,\mathbf{p})] \leq b_d(\omega), \qquad \forall\ \omega \in [\omega', \omega''] . \tag{3.3.2}$$

By the generalized *Bode's* theorem [Boy.1]

$$\int_0^\infty \log\{\bar{\sigma}[\hat{\mathbf{H}}_{yd}(j\omega,\mathbf{p})]\} d\omega \geq 0 . \tag{3.3.3}$$

Therefore, if $\bar{\sigma}[\hat{\mathbf{H}}_{yd}(j\omega,\mathbf{p})]$ is small (<1) in the frequency range $[\omega_1, \omega_2]$ of interest, then $\bar{\sigma}[\hat{\mathbf{H}}_{yd}(j\omega,\mathbf{p})]$ will be greater than 1 in another frequency range, say $[\omega_3, \omega_4]$. In order to avoid an excessive "bump" outside the frequency range of interest, the following semi-infinite optimization problem can be set up.

$$\min_{\mathbf{p}} \max_{\omega \in [\omega_1, \omega_2]} \bar{\sigma}[\hat{\mathbf{H}}_{yd}(j\omega,\mathbf{p})] \qquad \text{subject to} \tag{3.3.4}$$

$$\bar{\sigma}[\hat{\mathbf{H}}_{yd}(j\omega,\mathbf{p})] \leq b_u , \qquad \forall\ \omega \in [\omega_3, \omega_4] .$$

where $b_u > 1$. The constraint is as shown in Fig.3.6b.

- *Plant Saturation Avoidance:*

To avoid plant saturation by, say, output disturbances, it may be necessary to impose constraints on the norm (maximum singular value $\bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{u},\mathbf{d}}(j\omega,\mathbf{p})]$) of the transfer function $\hat{\mathbf{H}}_{\mathbf{u},\mathbf{d}}$ over a range of frequencies. This can be done conveniently only for the nominal design, as follows:

$$\bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{u},\mathbf{d}}(j\omega,\mathbf{p})] \leq b_s(\omega), \qquad \forall\ \omega \in [\omega',\omega''] . \tag{3.3.5}$$

- *I/O Map Decoupling:*

In many design problems, it is desirable to have a *decoupled* (*diagonal*) I/O map $\hat{\mathbf{H}}_{\mathbf{yu}}$. This can be achieved by reducing interaction in the I/O map, as follows:

$$| \hat{\mathbf{H}}_{\mathbf{y}^i\mathbf{u}^j}(j\omega,\mathbf{p}) | < \epsilon, \qquad \forall\ i \neq j\ ,\ \forall\ \omega \in \mathbb{R}\ , \tag{3.3.6}$$

where $\hat{\mathbf{H}}_{\mathbf{y}^i\mathbf{u}^j}(s,\mathbf{p}) = [\hat{\mathbf{H}}_{\mathbf{yu}}(s,\mathbf{p})]_{ij}$ is the scalar transfer function from the input $\mathbf{u}^j$ to the output $\mathbf{y}^i$.

In summary, constraints in the frequency domain can be formulated as bounds on a norm of the transfer function. When the induced L_2 norm is used, the L_2 norm can be computed by making use of the fact that it is equal to the maximum singular value of a transfer function matrix or to the absolute value of the scalar transfer function. Therefore, for the interconnected system $\Sigma(\mathbf{p})$, the constraints in the frequency domain have the following mathematical forms:

$$\bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{vw}}(j\omega,\mathbf{p})] \leq b(\omega), \qquad \forall\ \omega \in [\omega',\omega''] . \tag{3.3.7a}$$

or

$$| [\hat{\mathbf{H}}_{\mathbf{vw}}(j\omega,\mathbf{p})]_{ij} | \leq b_{ij}(\omega), \qquad \forall\ \omega \in [\omega',\omega''] . \tag{3.3.7b}$$

3.3.2 Time Domain Specifications

In many design problems, time domain design specifications are essential. In general, there are no simple relations between time responses and frequency responses. Therefore, time domain design specifications can not be transcribed into frequency domain specifications. In this subsection, we shall show several classes of time domain constraints.

- *Input Signal Tracking:*

Satisfactory input signal tracking for inputs $r(t)$ can be ensured only for the nominal design. Tracking is specified by semi-infinite inequalities of the form (see Fig.3.7):

$$\underline{b}^j(t) \leq y^j(t, p, r(\cdot)) \leq \overline{b}^j(t), \qquad \forall\ t \in [0, T],\ j = 1, 2, \cdots, m\ . \quad (3.3.8)$$

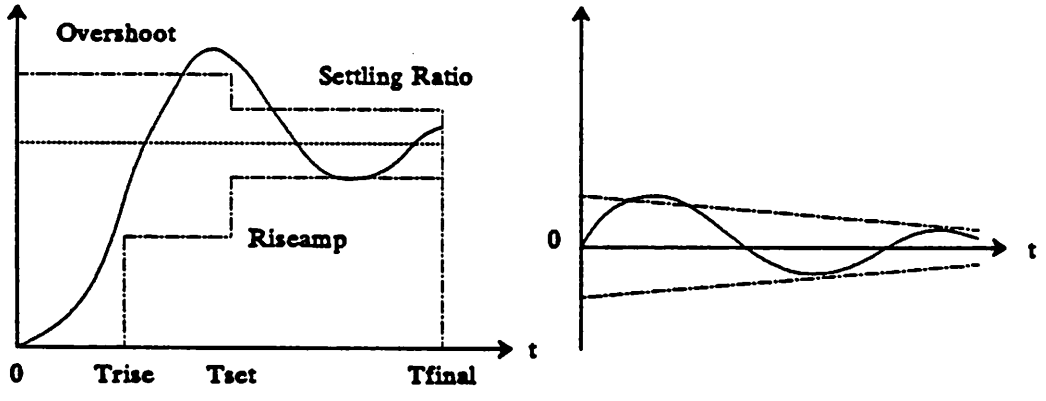


Fig. 3.7. Envelope constraint for time responses.

- *Plant Saturation Avoidance by Input Signal:*

To avoid plant saturation by input signals, it may be necessary to impose time domain constraints directly on the input of the plant. This can be done conveniently

only for the nominal design, as follows:

$$| \mathbf{u}_p{}^j(t, \mathbf{p}, \mathbf{r}(\cdot)) | \leq \mathbf{b}_p{}^j, \qquad \forall\ t \in [0, T_p],\ j = 1, 2, \cdots, m\ . \tag{3.3.9}$$

Plant saturation avoidance by the bounded inputs can be ensured by imposing the constraints either on impulse response matrix $\mathbf{H}_{\mathbf{ru}_p}$ of the form:

$$\bar{\sigma}[\mathbf{H}_{\mathbf{ru}_p}(t, \mathbf{p})] \leq a \cdot e^{-\lambda t}, \qquad \forall\ t \in [0, \infty)\ . \tag{3.3.10a}$$

or on the scalar impulse response of the form:

$$| [\mathbf{H}_{\mathbf{ru}_p}(t, \mathbf{p})]_{ij} | \leq a_{ij} \cdot e^{-\lambda_{ij} t}, \qquad \forall\ t \in [0, T],\ \text{for some } i, j\ . \tag{3.3.10b}$$

where a, λ, a_{ij} and λ_{ij} are positive. In some situations, the design based on the formulations (3.3.10a.b) in order to avoid plant saturation is conservative. Formulations in the H^∞ norm can be found in [Sal.1].

In summary, constraints in the time domain can be divided into two categories. The first category includes constraints imposed on time responses directly. The second category includes constraints imposed on the norm of the impulse response. Therefore, for the interconnected system $\Sigma(\mathbf{p})$, the constraints in the time domain have the following mathematical forms:

$$\underline{\mathbf{b}}^j(t) \leq \mathbf{y}^j(t, \mathbf{p}, \mathbf{r}(\cdot)) \leq \bar{\mathbf{b}}^j, \qquad \forall\ t \in [0, T],\ j = 1, 2, \cdots, m, \tag{3.3.11a}$$

$$\bar{\sigma}[\mathbf{H}_{\mathbf{vw}}(t, \mathbf{p})] \leq b(t), \qquad \forall\ t \in [0, T], \tag{3.3.11b}$$

and

$$| [\mathbf{H}_{\mathbf{vw}}(t, \mathbf{p})]_{ij} | \leq b_{ij}(t), \qquad \forall\ t \in [0, T], \quad \text{for some } i, j\ . \tag{3.3.11c}$$

3.3.3 Design Parameter Limitations

There are some physical constraints which should be taken into consideration in the implementation of the real systems. The most common constraints are on values of the design parameters. This can be achieved by imposing box constraints:

$$\underline{b}^i \leq p^i \leq \overline{b}^i, \qquad \text{for some } i. \tag{3.3.12}$$

where $\underline{b}^i$ and $\overline{b}^i \in \mathbb{R}$.

3.4 Concluding Remarks

As described in the section 3.1, the appropriate canonical optimization problem into which control system design problems must be transcribed has the form

$$\min \{ f(x) \mid g^j(x) \leq 0,\ j = 1,2,\cdots,m;$$
$$\phi^k(x,y_k) \leq 0,\ y_k \in Y_k,\ k = 1,2,\cdots,l \}, \tag{3.4.1}$$

where the Y_k are finite time or frequency intervals. A family of algorithms for solving a problem of this form were developed over the years by Polak and Mayne [Pol.12], Gonzaga, Polak, and Trahan [Gon.1], Polak, Trahan and Mayne [Pol.5], Polak and Wardi [Pol.2], and Polak [Pol.3, Pol.4]. The algorithms were constructed with broad engineering applications in mind, while the Polak-Wardi algorithm [Pol.2] was specifically conceived for control system design.

In the formulation of design specifications as inequalities or semi-infinite inequalities, we have ensured that the resulting functions are at least locally Lipschitz continuous. This guarantees the existence of gradients or generalized gradients of these functions. In order to apply the above mentioned algorithms to a set of inequalities and semi-infinite inequalities described in this chapter, the evaluations of the functions and their gradients or their generalized gradients must be provided. Because of the intensive computation necessary for interactive control system design using optimization, an efficient, reliable, robust simulator for the class of interconnected system $\Sigma(p_c)$ described in (2.4.4a,b,c) was deemed necessary.

CHAPTER 4

SIMULATION OF MULTIVARIABLE INTERCONNECTED SYSTEMS

4.1 Introduction

One of the key requirements in the analysis and design of linear time-invariant control systems is the need to examine both time domain and frequency domain responses. When designing a linear control system using semi-infinite optimization techniques, it is necessary to compute not only time and frequency domain responses but also their sensitivities to the design parameters, see for example, [Pol.1].

The class of dynamic systems to be considered in this dissertation is described by parameterized linear time-invariant state equations. Frequency domain models can be derived via Laplace transform from time domain models. Sensitivity models of the dynamic system can be obtained either by differentiating the system equations with respect to the design parameters [Kok.1], by applying the "sensitivity points method" when the system is of a special form [Kok.2], or by applying the "structural method" [Kok.1].

There is considerable literature on the solution of state equations and the evaluation of frequency responses, see, e.g., [Lau.1, Lau.2, Lau.3, Par.1, Par.2, Bav.1]. The existing software for computing time domain and frequency domain responses is reliable and efficient. However, the existing software for the computation of the sensitivity functions is not as efficient as that for the computation of the time and frequency responses. In the past, particular attention was focused on the simultaneous generation of all sensitivity functions with respect to design parameters. Since the system of differential equations (including the sensitivity equations) tends to be rather large, a great deal of effort has been invested to find ways for either decomposing or reducing the dimension of the system (see e.g., [Cru.1, Fra.1, Kok.1, Kok.2]).

The purpose of this chapter is to describe the mathematical theory and numerical algorithms which were used to create the stable and efficient simulator: *MIMO*. Particular attention is focused on the efficient computation of both time and frequency

domain response sensitivities with respect to design parameters.

4.2 Frequency Domain Simulation

Optimization based feedback system design in the frequency domain requires repeated computations of the closed-loop characteristic polynomial and of various transfer functions. These are needed to ensure stability (by either checking that the closed loop characteristic polynomial has all its roots in the left half plane or by using the modified Nyquist criterion) and to obtain useful performance measures such as constraints on singular values of transfer function matrices. Since these functions are closely related, redundant computations can occur. In this section, we present a way to avoid these redundant computations in frequency domain simulation.

4.2.1 Evaluation of Characteristic Polynomials and Their Derivatives

As mentioned in Chapter 3, one major objection to using the closed loop eigenvalues in a stability test, in optimization-based design, stems from the fact that eigenvalues are not locally Lipschitz continuous functions. Hence we use the *Modified Nyquist Criterion* (3.2.10a), (3.2.10b) which requires the evaluation of the characteristic polynomial $\chi(s,\mathbf{p}_c)$ and its partial derivatives. We shall now describe the method we chose to perform these evaluations.

The characteristic polynomial of the interconnected system $\Sigma(\mathbf{p}_c)$ in (2.4.4a,b,c) is defined by $\chi(s,\mathbf{p}_c)=\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$. For $s \in \mathbb{C}$, the evaluation of $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ can be very time consuming, partly because it involves complex arithmetic and partly because the dimension of the matrix $\mathbf{A}_c$ may be large. Hence, to ensure the efficient evaluation of $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ over a range of values of s, matrix decomposition methods must be used.

Matrix decomposition methods are based on similarity transformations yielding a matrix

$$\widetilde{\mathbf{A}}_c(\mathbf{p}_c) = \mathbf{V}(\mathbf{p}_c)^{-1}\mathbf{A}_c(\mathbf{p}_c)\mathbf{V}(\mathbf{p}_c)\,. \tag{4.2.1}$$

For the purpose of facilitating the evaluation of $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$, it is necessary to find

a transformation matrix $\mathbf{V}(\mathbf{p}_c)$ which results in a matrix $\widetilde{\mathbf{A}}_c(\mathbf{p}_c)$ such that $\det[s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]$ is easy to compute. The simplest situation occurs when the matrix $\mathbf{A}_c(\mathbf{p}_c)$ is diagonalizable, i.e., there exists a matrix of eigenvectors $\mathbf{V}(\mathbf{p}_c)$ such that $\widetilde{\mathbf{A}}_c(\mathbf{p}_c)=\Lambda_c(\mathbf{p}_c)$, with $\Lambda_c(\mathbf{p}_c)$ a diagonal matrix whose diagonal elements are the eigenvalues $\lambda_j(\mathbf{p}_c)$ of the matrix $\mathbf{A}_c(\mathbf{p}_c)$. In that case, assuming that $\mathbf{A}_c(\mathbf{p}_c)$ is an $n_{s,c}\times n_{s,c}$ matrix,

$$\chi(s,\mathbf{p}_c) = \det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)] = \det[s\mathbf{I}-\Lambda_c(\mathbf{p}_c)] = \prod_{j=1}^{n_{s,c}}[s-\lambda_j(\mathbf{p}_c)]\,. \tag{4.2.2}$$

Unfortunately, the diagonalization of a nonsymmetric matrix $\mathbf{A}_c(\mathbf{p}_c)$ can be arbitrarily ill-conditioned. Consequently, diagonalization cannot be used reliable to compute $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ when $\mathbf{A}_c(\mathbf{p}_c)$ has fewer than $n_{s,c}$ eigenvectors or when the transformation matrix $\mathbf{V}(\mathbf{p}_c)$ is close to singular, see [Lau.1]. Thus formula (4.2.2) should not be used when the condition number $cond(\mathbf{V}(\mathbf{p}_c)) \triangleq \|\mathbf{V}(\mathbf{p}_c)\|\,\|\mathbf{V}(\mathbf{p}_c)^{-1}\|$ is large.

When diagonalization cannot be used, one can simplify the computation of $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ by first reducing $\mathbf{A}_c(\mathbf{p}_c)$ to upper Hessenberg form $\mathbf{H}_c(\mathbf{p}_c)$ by means of an orthogonal similarity transformation:

$$\mathbf{H}_c(\mathbf{p}_c) = \mathbf{V}(\mathbf{p}_c)^T\mathbf{A}_c(\mathbf{p}_c)\mathbf{V}(\mathbf{p}_c)\,, \tag{4.2.3a}$$

where $\mathbf{V}(\mathbf{p}_c)$ is a unitary matrix, so that $cond(\mathbf{V}(\mathbf{p}_c))=1$. This leads to the formula

$$\chi(s,\mathbf{p}_c) = \det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)] = \det[s\mathbf{I}-\mathbf{H}_c(\mathbf{p}_c)]\,. \tag{4.2.3b}$$

The Hessenberg form $\mathbf{H}_c(\mathbf{p}_c)$ is cheaper to compute than the diagonal form $\Lambda_c(\mathbf{p}_c)$. Furthermore, the computation of $\mathbf{H}_c(\mathbf{p}_c)$ is stable and the computation of $\det[s\mathbf{I}-\mathbf{H}_c(\mathbf{p}_c)]$ only requires some simple pivoting. Consequently, the computation of $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ by computing $\det[s\mathbf{I}-\mathbf{H}_c(\mathbf{p}_c)]$ is numerically stable and, for a *single* value of s, it is definitely less costly than the computation of $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ by evaluation of $\det[s\mathbf{I}-\Lambda_c(\mathbf{p}_c)]$. However, when, as in our case, one needs to evaluate $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ for *many* values of s, the cost of pivoting in the computation of $\det[s\mathbf{I}-\mathbf{H}_c(\mathbf{p}_c)]$ dominates and it is preferable to use formula (4.2.2), provided that

$\mathbf{A}_c(\mathbf{p}_c)$ is not near defective.

Next we turn to the computation of the partial derivatives of $\chi(s,\mathbf{p})$. When the eigenvalues $\lambda_j(\mathbf{p}_c)$ of $\mathbf{A}_c(\mathbf{p}_c)$ are distinct, they are differentiable (see [Kat.1]) and their partial derivatives are given by

$$\frac{\partial \lambda_j(\mathbf{p}_c)}{\partial \mathbf{p}_c{}^i} = \frac{\left\langle \mathbf{u}_j, \frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}_c{}^i}\mathbf{v}_j \right\rangle}{\langle \mathbf{u}_j, \mathbf{v}_j \rangle}, \tag{4.2.4}$$

where $\mathbf{v}_j$ and $\mathbf{u}_j$ are the right and left eigenvectors, respectively, of $\mathbf{A}_c(\mathbf{p}_c)$, corresponding to the eigenvalue $\lambda_j(\mathbf{p}_c)$. Therefore when the eigenvalues of $\mathbf{A}_c(\mathbf{p}_c)$ are distinct, the partial derivatives of $\chi(s,\mathbf{p})$ can be computed making use of the following formula:

$$\frac{\partial \chi(s,\mathbf{p}_c)}{\partial \mathbf{p}_c{}^i} = \sum_{j=1}^{n_{s,c}} \left\{ -\frac{\partial \lambda_j(\mathbf{p}_c)}{\partial \mathbf{p}_c{}^i} \prod_{\substack{k=1 \\ k\neq j}}^{n_{s,c}} [s-\lambda_k(\mathbf{p}_c)] \right\}$$

$$= \det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)] \sum_{j=1}^{n_{s,c}} -\frac{\partial \lambda_j(\mathbf{p}_c)}{\partial \mathbf{p}_c{}^i} \frac{1}{s-\lambda_j(\mathbf{p}_c)}. \tag{4.2.5}$$

When the eigenvalues of $\mathbf{A}_c(\mathbf{p}_c)$ are not distinct, the computation of $\partial \det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]/\partial \mathbf{p}_c{}^i$ becomes much more difficult and requires the use of a general formula which we shall now develop.

Proposition 4.2.1: Let $\mathbf{M}(s,\mathbf{p}_c) \triangleq s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)$, let $\mathbf{m}_{j,\bullet}$ and $\mathbf{m}_{\bullet,j}$ denote the jth row and the jth column of $\mathbf{M}(s,\mathbf{p}_c)$, let $\mathbf{M}^i_{j,\bullet}(s,\mathbf{p}_c)$ denote the matrix obtained from $\mathbf{M}(s,\mathbf{p}_c)$ by replacing its jth row by $\frac{\partial \mathbf{m}_{j,\bullet}}{\partial \mathbf{p}_c{}^i}$, and let $\mathbf{M}^i_{\bullet,j}(s,\mathbf{p}_c)$ denote the matrix obtained from $\mathbf{M}(s,\mathbf{p}_c)$ by replacing its jth column by $\frac{\partial \mathbf{m}_{\bullet,j}}{\partial \mathbf{p}_c{}^i}$. Then,

$$\frac{\partial \det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]}{\partial \mathbf{p}_c{}^i} = \sum_{j=1}^{n_{s,c}} \det[\mathbf{M}^i_{j,\bullet}(s,\mathbf{p}_c)] = \sum_{j=1}^{n_{s,c}} \det[\mathbf{M}^i_{\bullet,j}(s,\mathbf{p}_c)]. \tag{4.2.6}$$

Proof: We shall give a proof by induction. Equation (4.2.6) is clearly true for 2×2

matrices. Expanding a $(k+1)\times(k+1)$ determinant in terms of $k\times k$ cofactors, we see that if formula (4.2.6) holds for $k\times k$ matrices, it must also hold for $(k+1)\times(k+1)$ matrices.

∎

The computation of $\partial\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]/\partial\mathbf{p}_c^{\,i}$ according to (4.2.6) is very expensive. Fortunately, closed loop control systems seldom have repeated eigenvalues. In addition, the matrix $\partial\mathbf{A}_c(\mathbf{p}_c)/\partial\mathbf{p}_c^{\,i}$ is sometimes very sparse, with the likely consequence that $\partial\mathbf{m}_{j,\cdot}/\partial\mathbf{p}_c^{\,i}=0$ for some j and hence that $\det[\mathbf{M}^i_{j,\cdot}(s,\mathbf{p}_c)]=0$.

To conclude this subsection, we shall summarize our suggestion for evaluating $\chi(s,\mathbf{p}_c)$ and its partial derivatives in the form of an algorithm. We make use of the fact that one of the most robust methods (used in EISPACK [Smi.1]) for diagonalizing a matrix $\mathbf{A}_c(\mathbf{p}_c)$ is first to reduce it to upper Hessenberg form $\mathbf{H}_c(\mathbf{p}_c)$, using orthogonal similarity transformations, and then to reduce the Hessenberg form $\mathbf{H}_c(\mathbf{p}_c)$ to Schur form $\mathbf{S}_c(\mathbf{p}_c)$ by iterative unitary similarity transformations of the QR method. Finally, back substitutions are applied to accumulate eigenvectors.

Algorithm 4.2.1: (*Evaluates* $\chi(s,\mathbf{p}_c)$ *and its partial derivatives*)

Data : Matrix $\mathbf{A}_c(\mathbf{p}_c)$, $\kappa >> 0$, an upper bound on acceptable condition number.

Step 1 : Reduce matrix $\mathbf{A}_c(\mathbf{p}_c)$ to Hessenberg form $\mathbf{H}_c(\mathbf{p}_c)$ by orthogonal transformation $\mathbf{Q}(\mathbf{p}_c)$ so that

$$\mathbf{H}_c(\mathbf{p}_c)=\mathbf{Q}^T(\mathbf{p}_c)\mathbf{A}_c(\mathbf{p}_c)\mathbf{Q}(\mathbf{p}_c)\,. \tag{4.2.8}$$

Step 2 : Attempt to reduce Hessenberg form $\mathbf{H}_c(\mathbf{p}_c)$ to Schur form $\mathbf{S}_c(\mathbf{p}_c)$ by the QR method.

(i) If the QR method fails to converge, i.e. the eigenvalues of $\mathbf{A}_c(\mathbf{p}_c)$ cannot be computed, then compute $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ using (4.2.3b) and $\partial\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]/\partial\mathbf{p}_c^{\,i}$, $i=1,\cdots,n_{p,c}$, using formula (4.2.6).
Stop and exit.

(ii) If the reduction to Schur form was successful, then a unitary matrix $\mathbf{R}(\mathbf{p}_c)$ was constructed such that

$$\mathbf{S}_c(\mathbf{p}_c) = \mathbf{R}(\mathbf{p}_c)^* \mathbf{H}_c(\mathbf{p}_c)\mathbf{R}(\mathbf{p}_c)\,. \tag{4.2.9}$$

The eigenvalues $\lambda_i(\mathbf{p}_c)$ of $\mathbf{A}_c(\mathbf{p}_c)$ are given by

$$\lambda_i(\mathbf{p}_c) = [\mathbf{S}_c(\mathbf{p}_c)]_{ii}\,,\quad i = 1, 2, \cdots, n_{s,c}\,. \tag{4.2.10}$$

Step 3 : Compute a matrix of eigenvectors, $\mathbf{P}(\mathbf{p}_c)$, of the upper triangular Schur form $\mathbf{S}_c(\mathbf{p}_c)$ by back substitutions and compute its inverse $\mathbf{P}(\mathbf{p}_c)^{-1}$ (if it exists) and its condition number $cond(\mathbf{P}(\mathbf{p}_c))$.

Step 4 : If $cond(\mathbf{P}(\mathbf{p}_c)) \geq \kappa$ or $\mathbf{P}(\mathbf{p}_c)$ is singular, compute $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ and its partial derivatives using (4.2.3b), (4.2.6).

Else construct the diagonal matrix $\Lambda_c(\mathbf{p}_c) = \mathrm{diag}[\lambda_1(\mathbf{p}_c), \cdots, \lambda_{n_{s,c}}(\mathbf{p}_c)]$ which satisfies

$$\Lambda_c(\mathbf{p}_c) = \mathbf{P}(\mathbf{p}_c)^{-1}\mathbf{S}_c(\mathbf{p}_c)\mathbf{P}(\mathbf{p}_c)\,, \tag{4.2.11}$$

and compute the right and left eigenvector matrices

$$\mathbf{V}(\mathbf{p}_c) = \mathbf{Q}(\mathbf{p}_c)\mathbf{R}(\mathbf{p}_c)\mathbf{P}(\mathbf{p}_c)\,, \tag{4.2.12a}$$

$$\mathbf{U}(\mathbf{p}_c) \triangleq \mathbf{V}^{-1}(\mathbf{p}_c) = \mathbf{P}^{-1}(\mathbf{p}_c)\mathbf{R}^*(\mathbf{p}_c)\mathbf{Q}^T(\mathbf{p}_c)\,. \tag{4.2.12b}$$

Note : since $\mathbf{Q}$ and $\mathbf{R}$ are orthogonal matrices, $cond(\mathbf{V}) = cond(\mathbf{P})$.

Step 5 : Set $\mathbf{v}_j$ to be the jth column of $\mathbf{V}(\mathbf{p}_c)$, set $\mathbf{u}_j^T$ to be the jth row of $\mathbf{U}(\mathbf{p}_c)$ and compute $\det[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]$ and its partial derivatives using (4.2.2), (4.2.5). Stop and exit.

■

4.2.2 Evaluation of Transfer Functions and Their Sensitivities

Consider the multivariable interconnected system Σ described by (2.4.4a,b,c). The frequency response matrices from the input to each summing node $\mathbf{r}_k$ to the input $\mathbf{u}_i$, the state $\mathbf{x}_i$ and the output $\mathbf{y}_i$ of each subsystem are required. Therefore, the

three transfer function matrices $\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c)$, $\hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c)$ and $\hat{\mathbf{H}}_{ur}(s,\mathbf{p}_c)$ are needed for frequency domain analysis and design. These transfer function matrices and their derivatives are given by

$$\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c) = [s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]^{-1}\mathbf{B}_c(\mathbf{p}_c), \tag{4.2.13a}$$

$$\hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c) = \mathbf{C}_c(\mathbf{p}_c)\,\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c) + \mathbf{D}_c(\mathbf{p}_c), \tag{4.2.13b}$$

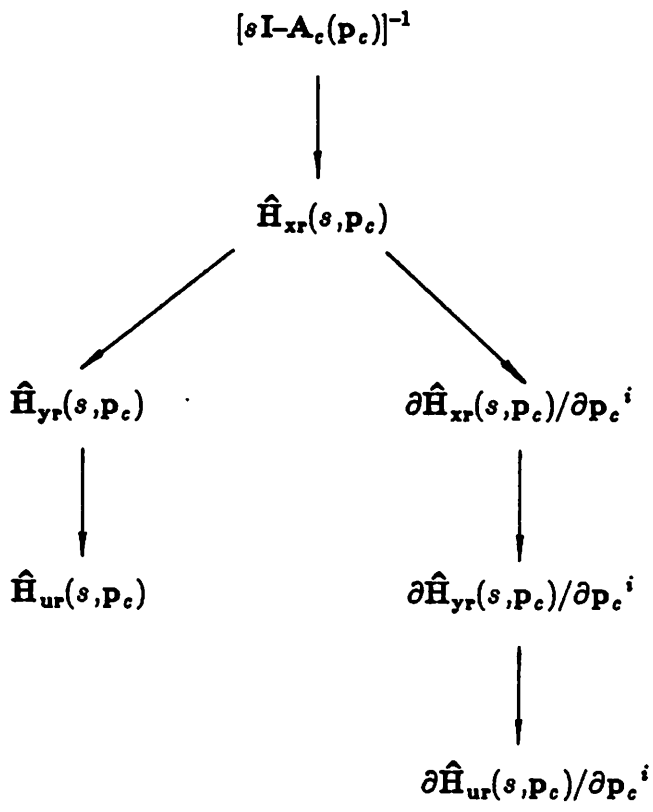
$$\hat{\mathbf{H}}_{ur}(s,\mathbf{p}_c) = \mathbf{I} + \mathbf{E}\,\hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c), \tag{4.2.13c}$$

$$\frac{\partial \hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}} = [s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]^{-1}\left[\frac{\partial \mathbf{A}_c(\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}}\,\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c) + \frac{\partial \mathbf{B}_c(\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}}\right], \tag{4.2.13d}$$

$$\frac{\partial \hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}} = \mathbf{C}_c(\mathbf{p}_c)\frac{\partial \hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}} + \frac{\partial \mathbf{C}_c(\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}}\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c) + \frac{\partial \mathbf{D}_c(\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}} \tag{4.2.13e}$$

$$\frac{\partial \hat{\mathbf{H}}_{ur}(s,\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}} = \mathbf{E}\,\frac{\partial \hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c)}{\partial \mathbf{p}_c^{\,i}}. \tag{4.2.13f}$$

Since these six functions are closely related, an efficient computational scheme is needed to avoid repeating computations. A natural flow of computation for these functions can be expressed by the following diagram:



This shows the dependence of each function. Tracing down the tree shown in the diagram, the algorithm will guarantee that each function will be computed at most once for each $(s,\mathbf{p}_c)$. For example, if the function $\partial\hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c)/\partial\mathbf{p}_c{}^i$ is required, the algorithm will first check if $\partial\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c)/\partial\mathbf{p}_c{}^i$ has been computed. If yes, then the equation (4.2.13d) is evaluated; if not, then trace the tree diagram one level up.

From the diagram and equations (4.2.13a,b,c,d,e,f), it is clear that the computation of $[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]^{-1}$ is the bottle neck for the frequency domain simulation. Again, *matrix decomposition methods* are likely to be most efficient for the evaluation of $[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]^{-1}$. All matrix decompositions are based on similarity transformations of the form in (4.2.1). Therefore,

$$[s\mathbf{I}-\mathbf{A}_c(\mathbf{p}_c)]^{-1} = \mathbf{V}(\mathbf{p}_c)[s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1}\mathbf{V}^{-1}(\mathbf{p}_c)\,. \tag{4.2.14}$$

This is equivalent to performing the transformation, $\xi_c(t,\mathbf{p}_c) \triangleq \mathbf{V}^{-1}\mathbf{x}_c(t,\mathbf{p}_c)$, on the state space. Let $\widetilde{\mathbf{A}}_c \triangleq \mathbf{V}^{-1}\mathbf{A}_c\mathbf{V}$, $\widetilde{\mathbf{B}}_c \triangleq \mathbf{V}^{-1}\mathbf{B}_c$, $\widetilde{\mathbf{C}}_c \triangleq \mathbf{C}_c\mathbf{V}$, $\widetilde{\mathbf{D}}_c \triangleq \mathbf{D}_c$, then the

new state equations are

$$\dot{\xi}_c(t,\mathbf{p}_c) = \widetilde{\mathbf{A}}_c(\mathbf{p}_c)\,\xi_c(t,\mathbf{p}_c) + \widetilde{\mathbf{B}}_c(\mathbf{p}_c)\,\mathbf{r}_c(t), \tag{4.2.15a}$$

$$\mathbf{y}_c(t,\mathbf{p}_c) = \widetilde{\mathbf{C}}_c(\mathbf{p}_c)\,\xi_c(t,\mathbf{p}_c) + \widetilde{\mathbf{D}}_c(\mathbf{p}_c)\,\mathbf{r}_c(t), \tag{4.2.15b}$$

$$\mathbf{u}_c = \mathbf{r}_c + \mathbf{E}\,\mathbf{y}_c, \tag{4.2.15c}$$

Let $\dfrac{\Delta\widetilde{\mathbf{A}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}} \triangleq \mathbf{V}^{-1}(\mathbf{p}_c)\dfrac{\partial\mathbf{A}_c(\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}}\mathbf{V}(\mathbf{p}_c)$, $\dfrac{\Delta\widetilde{\mathbf{B}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}} \triangleq \mathbf{V}^{-1}(\mathbf{p}_c)\dfrac{\partial\mathbf{B}_c(\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}}$, $\dfrac{\Delta\widetilde{\mathbf{C}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}} \triangleq \dfrac{\partial\mathbf{C}_c(\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}}\mathbf{V}(\mathbf{p}_c)$, and $\dfrac{\Delta\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}} \triangleq \mathbf{V}^{-1}(\mathbf{p}_c)\dfrac{\partial\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}}$, then the transfer functions can be expressed as follows

$$\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c) = [s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1}\widetilde{\mathbf{B}}_c(\mathbf{p}_c), \tag{4.2.16a}$$

$$\hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c) = \widetilde{\mathbf{C}}_c(\mathbf{p}_c)\,\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c) + \widetilde{\mathbf{D}}_c(\mathbf{p}_c), \tag{4.2.16b}$$

$$\hat{\mathbf{H}}_{ur}(s,\mathbf{p}_c) = \mathbf{I} + \mathbf{E}\,\hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c), \tag{4.2.16c}$$

$$\frac{\Delta\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}} = [s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1}\left[\frac{\Delta\widetilde{\mathbf{A}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}}\,\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c) + \frac{\Delta\widetilde{\mathbf{B}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}}\right], \tag{4.2.16d}$$

$$\frac{\partial\hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}} = \widetilde{\mathbf{C}}_c(\mathbf{p}_c)\frac{\Delta\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}} + \frac{\Delta\widetilde{\mathbf{C}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}}\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c) + \frac{\partial\widetilde{\mathbf{D}}_c(\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}} \tag{4.2.16e}$$

$$\frac{\partial\hat{\mathbf{H}}_{ur}(s,\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}} = \mathbf{E}\,\frac{\partial\hat{\mathbf{H}}_{yr}(s,\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}}, \tag{4.2.16f}$$

$$\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c) = \mathbf{V}(\mathbf{p}_c)\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c), \tag{4.2.16g}$$

$$\frac{\partial\hat{\mathbf{H}}_{xr}(s,\mathbf{p}_c)}{\partial\mathbf{p}_c^{\,i}} = \mathbf{V}(\mathbf{p}_c)\frac{\Delta\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}}. \tag{4.2.16h}$$

Note that $\dfrac{\Delta\widetilde{\mathbf{A}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}}$, $\dfrac{\Delta\widetilde{\mathbf{B}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}}$, $\dfrac{\Delta\widetilde{\mathbf{C}}_c(\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}}$, and $\dfrac{\Delta\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c)}{\Delta\mathbf{p}_c^{\,i}}$ are *not* the derivatives of $\widetilde{\mathbf{A}}_c(\mathbf{p}_c)$, $\widetilde{\mathbf{B}}_c(\mathbf{p}_c)$, $\widetilde{\mathbf{C}}_c(\mathbf{p}_c)$, and $\hat{\mathbf{H}}_{\xi r}(s,\mathbf{p}_c)$.

The reason for using matrix decomposition methods in (4.2.1) and (4.2.14) is that they enable us to find a transformation matrix $\mathbf{V}(\mathbf{p}_c)$ for which $[s\mathbf{I}-\widetilde{\mathbf{A}}_c]^{-1}$ is easy to compute. As s result, the evaluation of the transfer functions via (4.2.16a,b,c,d,e,f,g,h) is more efficient than by using the equations (4.2.15a,b,c,d,e,f). It is clear that if $\widetilde{\mathbf{A}}_c$ is a diagonal matrix, $[s\mathbf{I}-\widetilde{\mathbf{A}}_c]^{-1}$ can be obtained without using numerical inversion. However, as discussed in the section 4.2.1, the condition number of the eigenvector matrix $\mathbf{V}(\mathbf{p}_c)$ may approach infinity, when the matrix $\mathbf{A}_c(\mathbf{p}_c)$ is near defective. Therefore, the Hessenberg form or the Shur form must be used when the condition number $\text{cond}(\mathbf{V}(\mathbf{p}_c))$ exceeds an acceptable level. We now state an algorithm of computing $[s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1}$ and $\mathbf{V}(\mathbf{p}_c)$:

Algorithm 4.2.2 : (*Computes* $[s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1}$ *and* $\mathbf{V}(\mathbf{p}_c)$)

Data : Matrix $\mathbf{A}_c(\mathbf{p}_c)$, $\kappa >> 0$, an upper bound on condition number.

Step 1 : Reduce matrix $\mathbf{A}_c(\mathbf{p}_c)$ to Hessenberg form $\mathbf{H}_c(\mathbf{p}_c)$ by orthogonal transformation $\mathbf{Q}(\mathbf{p}_c)$ so that

$$\mathbf{H}_c(\mathbf{p}_c) = \mathbf{Q}^T(\mathbf{p}_c)\mathbf{A}_c(\mathbf{p}_c)\mathbf{Q}(\mathbf{p}_c)\,. \tag{4.2.17}$$

Step 2: Attempt to reduce Hessenberg form $\mathbf{H}_c(\mathbf{p}_c)$ to Schur form $\mathbf{S}_c(p_c)$ by the QR method.

(i) If the QR method fails to converge, i.e. the eigenvalues cannot be computed, then

$$[s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1} = [s\mathbf{I}-\mathbf{H}_c(\mathbf{p}_c)]^{-1}; \qquad \mathbf{V}(\mathbf{p}_c) = \mathbf{Q}(\mathbf{p}_c)\,, \tag{4.2.18}$$

stop and exit.

(ii) If the reduction to Schur form $\mathbf{S}_c(\mathbf{p}_c)$ was successful, a unitary matrix $\mathbf{R}(\mathbf{p}_c)$ was constructed such that

$$\mathbf{S}_c(\mathbf{p}_c) = \mathbf{R}(\mathbf{p}_c)^*\mathbf{H}_c(\mathbf{p}_c)\mathbf{R}(\mathbf{p}_c)\,, \tag{4.2.19}$$

the eigenvalues $\lambda_i(\mathbf{p}_c)$ of $\mathbf{A}_c(\mathbf{p}_c)$ are given by

$$\lambda_i(\mathbf{p}_c) = [\mathbf{S}_c(\mathbf{p}_c)]_{ii}\,, \quad i = 1\,,2\,,\cdots\,,n_{s,c}\,. \tag{4.2.20}$$

Step 3 : Compute a matrix of eigenvectors, $\mathbf{P}(\mathbf{p}_c)$, of the upper triangular Schur form $\mathbf{S}_c(\mathbf{p}_c)$ by back substitutions and compute its inverse $\mathbf{P}(\mathbf{p}_c)^{-1}$ (if it exists) and its condition number $cond(\mathbf{P}(\mathbf{p}_c))$.

Step 4 : If $cond(\mathbf{P}(\mathbf{p}_c)) \geq \kappa$ or $\mathbf{P}(\mathbf{p}_c)$ is singular, compute

$$[s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1} = [s\mathbf{I}-\mathbf{S}_c(\mathbf{p}_c)]^{-1}; \qquad \mathbf{V}(\mathbf{p}_c) = \mathbf{Q}(\mathbf{p}_c)\mathbf{R}(\mathbf{p}_c), \tag{4.2.21}$$

Else construct the diagonal matrix $\Lambda_c(\mathbf{p}_c) = \text{diag}[\lambda_1(\mathbf{p}_c), \cdots, \lambda_{n_{s,c}}(\mathbf{p}_c)]$ which satisfies

$$\Lambda_c(\mathbf{p}_c) = \mathbf{P}(\mathbf{p}_c)^{-1}\mathbf{S}_c(\mathbf{p}_c)\mathbf{P}(\mathbf{p}_c), \tag{4.2.22}$$

and compute

$$\mathbf{V}(\mathbf{p}_c) = \mathbf{Q}(\mathbf{p}_c)\mathbf{R}(\mathbf{p}_c)\mathbf{P}(\mathbf{p}_c), \tag{4.2.23a}$$

$$[s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1} = [s\mathbf{I}-\Lambda_c(\mathbf{p}_c)]^{-1}, \tag{4.2.23b}$$

stop and exit.

∎

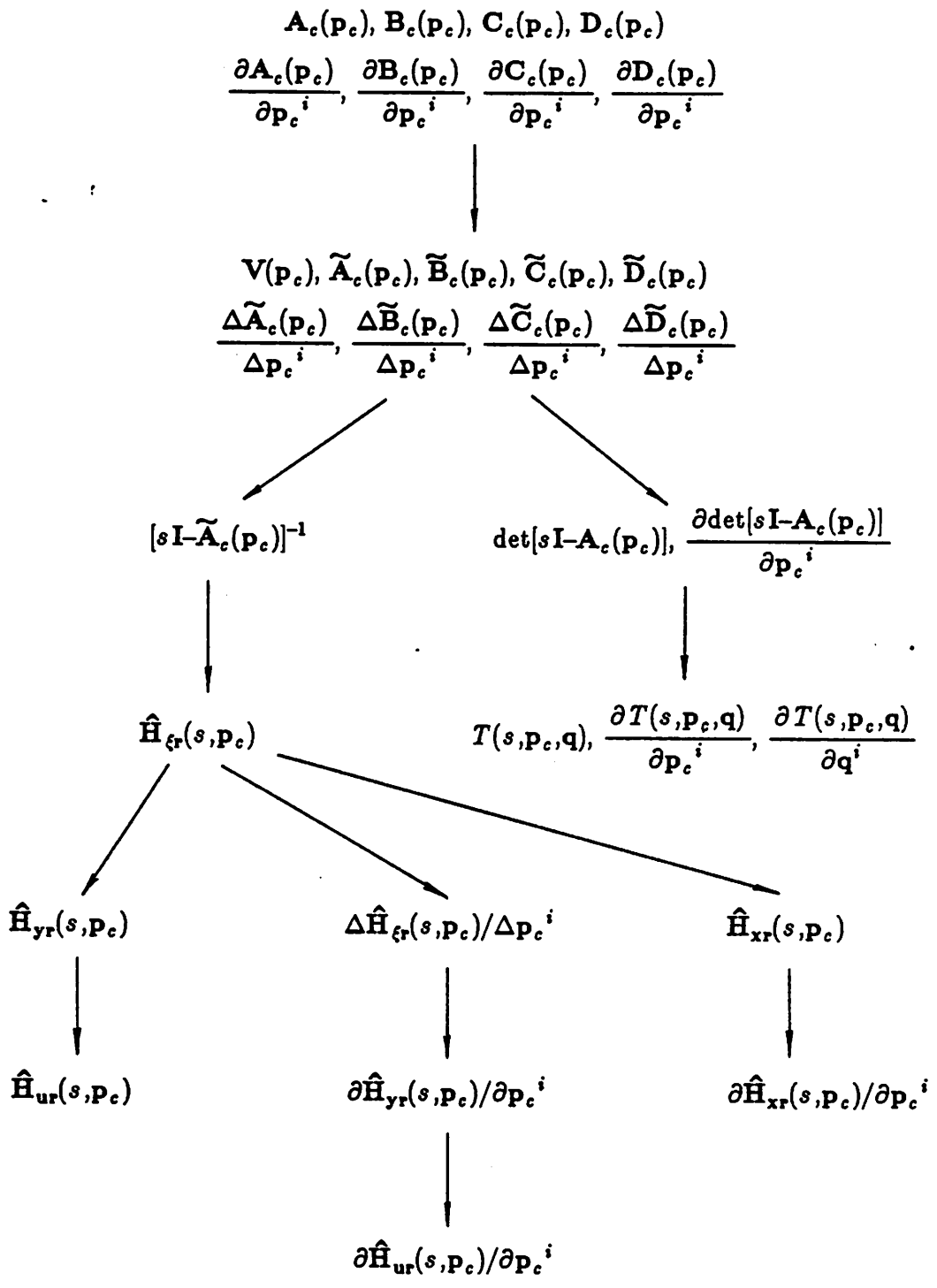
Note that since $\mathbf{Q}$ and $\mathbf{R}$ are orthogonal matrices, $cond(\mathbf{V}) = cond(\mathbf{P})$.

4.2.3 Summary of Frequency Domain Simulation

The *Modified Nyquist Criterion* stability test (3.2.10a) and (3.2.10b) requires the evaluation of $T(s, \mathbf{p}_c, \mathbf{q}) \triangleq \chi(s, \mathbf{p}_c)/D(s, \mathbf{q})$, $\partial T(s, \mathbf{p}_c, \mathbf{q})/\partial \mathbf{p}_c{}^i$ and $\partial T(s, \mathbf{p}_c, \mathbf{q})/\partial \mathbf{q}^i$. This can be easily computed by using Algorithm 4.2.1.

In the frequency domain analysis and design, the transfer functions given by (4.2.16a,b,c,d,e,f,g,h) need to be evaluated. By applying Algorithm 4.2.2 to obtain the matrix $[s\mathbf{I}-\widetilde{\mathbf{A}}_c(\mathbf{p}_c)]^{-1}$, these functions can be computed efficiently.

Both Algorithm 4.2.1 and Algorithm 4.2.2 are based on the same matrix decomposition technique, hence it is easy to combine these two algorithms together. We now can summarize the results in the following tree diagram which shows the precedence of the computation in frequency domain simulations.



4.3 Time Domain Simulation

Time domain simulation has its own important role in computer-aided control system design. It provides a direct sense of system performance in real time, which sometimes can not be shown in frequency domain simulation. In this section, we focus

on the computation of time domain responses and their sensitivity functions.

Consider the interconnected system Σ described by equations (2.5.6a,b) and (2.5.8a,b). Including the sensitivities of time domain responses, there are six time domain functions, $\mathbf{x}(t,\mathbf{p})$, $\mathbf{y}(t,\mathbf{p})$, $\mathbf{u}_c(t,\mathbf{p})$, $\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$, $\frac{\partial \mathbf{y}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ and $\frac{\partial \mathbf{u}_c(t,\mathbf{p})}{\partial \mathbf{p}^i}$, which are needed in time domain design. It is easy to see that the functions $\mathbf{y}(t,\mathbf{p})$, $\mathbf{u}_c(t,\mathbf{p})$, $\frac{\partial \mathbf{y}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ and $\frac{\partial \mathbf{u}_c(t,\mathbf{p})}{\partial \mathbf{p}^i}$ can be computed from $\mathbf{x}(t,\mathbf{p})$ and $\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ by using the equations (2.5.6b), (2.5.5) and (2.4.4c). Therefore in this section we only consider the computation of the functions $\mathbf{x}(t,\mathbf{p})$ and $\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$.

It follows from the assumptions in Chapter 2 and (2.5.7), that $\mathbf{A}(\cdot)$, $\mathbf{B}(\cdot)$ are continuously differentiable in $\mathbf{p}$, and hence the solution of the state equation (2.5.6a) is continuously differentiable in $(\mathbf{p},t)$ [Hal.1]. It is given by

$$\mathbf{x}(t,\mathbf{p}) = e^{\mathbf{A}(\mathbf{p})t}\mathbf{x}(0,\mathbf{p}) + \int_0^t e^{\mathbf{A}(\mathbf{p})(t-\tau)}\mathbf{B}(\mathbf{p})\mathbf{r}(\tau,\mathbf{p})d\tau, \tag{4.3.1}$$

and its partial derivatives are given by

$$\begin{aligned}\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i} &= \frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}\mathbf{x}(0,\mathbf{p}) + e^{\mathbf{A}(\mathbf{p})t}\frac{\partial \mathbf{x}(0,\mathbf{p})}{\partial \mathbf{p}^i} \\ &\quad + \int_0^t \frac{\partial e^{\mathbf{A}(\mathbf{p})(t-\tau)}}{\partial \mathbf{p}^i}\mathbf{B}(\mathbf{p})\mathbf{r}(\tau,\mathbf{p})d\tau + \int_0^t e^{\mathbf{A}(\mathbf{p})(t-\tau)}\frac{\partial \mathbf{B}(\mathbf{p})}{\partial \mathbf{p}^i}\mathbf{r}(\tau,\mathbf{p})d\tau \\ &\quad + \int_0^t e^{\mathbf{A}(\mathbf{p})(t-\tau)}\mathbf{B}(\mathbf{p})\frac{\partial \mathbf{r}(\tau,\mathbf{p})}{\partial \mathbf{p}^i}d\tau,\end{aligned} \tag{4.3.2}$$

for all $i \in \underline{n_p}$. Therefore, the evaluation of $\mathbf{x}(t,\mathbf{p})$ and $\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ involves the computation of the matrix exponential $e^{\mathbf{A}(\mathbf{p})t}$ and its partial derivative.

In this section, we first discuss the computation of the matrix $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$. A new formula for the matrix $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$ is presented. Then, the evaluation of the matrix exponential $e^{\mathbf{A}(\mathbf{p})t}$ is discussed. Finally, the completion of the computation of $\mathbf{x}(t,\mathbf{p})$ and $\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ is presented.

4.3.1 A Formula for the Matrix $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$

If we are to use (4.3.2) to compute $\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$, we must evaluate $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$.

Proposition 4.3.1 : Let $\mathbf{A}(\mathbf{p})$ be defined as in equation (2.5.6a), then

$$\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i} = e^{\mathbf{A}(\mathbf{p})t} \int_0^t e^{-\mathbf{A}(\mathbf{p})\tau} \frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i} e^{\mathbf{A}(\mathbf{p})\tau} d\tau. \tag{4.3.3}$$

Proof : By assumption, $\mathbf{A}(\cdot)$ is continuously differentiable. Hence $e^{\mathbf{A}(\mathbf{p})t}$ is continuously differentiable in $(\mathbf{p},t)$, and it therefore follows that (see [Mar.1])

$$\frac{d}{dt}\left[\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}\right] = \frac{\partial}{\partial \mathbf{p}^i}\left[\frac{de^{\mathbf{A}(\mathbf{p})t}}{dt}\right] = \frac{\partial}{\partial \mathbf{p}^i}\left[\mathbf{A}(\mathbf{p})\cdot e^{\mathbf{A}(\mathbf{p})t}\right]$$

$$= \mathbf{A}(\mathbf{p})\cdot\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i} + \frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i}\cdot e^{\mathbf{A}(\mathbf{p})t} \tag{4.3.4}$$

Integrating the linear differential equation (4.3.4) from 0 to t, we obtain

$$\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i} = e^{\mathbf{A}(\mathbf{p})t}\left[\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}\right]_{t=0} + e^{\mathbf{A}(\mathbf{p})t}\int_0^t e^{-\mathbf{A}(\mathbf{p})\tau}\frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i}e^{\mathbf{A}(\mathbf{p})\tau}d\tau \tag{4.3.5}$$

Since $\left[\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}\right]_{t=0} = 0$, (4.3.3) follows directly.

∎

The evaluation of $e^{\mathbf{A}(\mathbf{p})t}$ in (4.3.3) is relatively easy and can be carried out by some of the methods described in [Mol.1, Par.1, Lau.1]. An algorithm for the computation of the matrix exponential function will be presented in the next section. In general, the evaluation of the term, $\int_0^t e^{-\mathbf{A}(\mathbf{p})\tau}\frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i}e^{\mathbf{A}(\mathbf{p})\tau}d\tau$, in (4.3.3) is more problematic. However, the following two observations lead to the conclusion that there may be cases where this term can be computed without resorting to numerical integration.

(a) Let $\mathbf{V}\in\mathbb{R}^{n_s\times n_s}$ be such that $\mathbf{A}(\mathbf{p})\mathbf{V}-\mathbf{V}\mathbf{A}(\mathbf{p})=0$. Then $e^{\mathbf{A}(\mathbf{p})t}\mathbf{V}=\mathbf{V}e^{\mathbf{A}(\mathbf{p})t}$ and

$$\int_0^t e^{-\mathbf{A}(\mathbf{p})\tau}\,\mathbf{V}\,e^{\mathbf{A}(\mathbf{p})\tau}d\tau = t\mathbf{V}. \tag{4.3.6}$$

(b) For any $\mathbf{U}\in\mathbb{R}^{n_s\times n_s}$,

$$e^{-\mathbf{A}(\mathbf{p})t}\{\mathbf{A}(\mathbf{p})\mathbf{U}-\mathbf{U}\mathbf{A}(\mathbf{p})\}e^{\mathbf{A}(\mathbf{p})t} = -\frac{d}{dt}(e^{-\mathbf{A}(\mathbf{p})t}\mathbf{U}\,e^{\mathbf{A}(\mathbf{p})t}). \tag{4.3.7}$$

Proposition 4.3.2 : Let $\mathbf{A}(\mathbf{p})$ be defined as in equation (2.5.6a). If there exist $\mathbf{V}\in\mathbb{R}^{n_s\times n_s}$ and $\mathbf{U}\in\mathbb{R}^{n_s\times n_s}$, such that

$$\frac{\partial\mathbf{A}(\mathbf{p})}{\partial\mathbf{p}^i} = \mathbf{V}+\{\mathbf{A}(\mathbf{p})\mathbf{U}-\mathbf{U}\mathbf{A}(\mathbf{p})\}, \tag{4.3.8a}$$

$$\mathbf{A}(\mathbf{p})\mathbf{V}-\mathbf{V}\mathbf{A}(\mathbf{p}) = 0, \tag{4.3.8b}$$

for all $\mathbf{p}\in\mathbb{R}^{n_p}$, then

$$\int_0^t e^{-\mathbf{A}(\mathbf{p})\tau}\frac{\partial\mathbf{A}(\mathbf{p})}{\partial\mathbf{p}^i}e^{\mathbf{A}(\mathbf{p})\tau}d\tau = t\mathbf{V}+e^{-\mathbf{A}(\mathbf{p})t}\{e^{\mathbf{A}(\mathbf{p})t}\mathbf{U}-\mathbf{U}e^{\mathbf{A}(\mathbf{p})t}\}. \tag{4.3.9}$$

Proof : Since

$$\int_0^t e^{-\mathbf{A}(\mathbf{p})\tau}\frac{\partial\mathbf{A}(\mathbf{p})}{\partial\mathbf{p}^i}e^{\mathbf{A}(\mathbf{p})\tau}d\tau = \int_0^t e^{-\mathbf{A}(\mathbf{p})\tau}\{\mathbf{V}+[\mathbf{A}(\mathbf{p})\mathbf{U}-\mathbf{U}\mathbf{A}(\mathbf{p})]\}e^{\mathbf{A}(\mathbf{p})\tau}d\tau$$

$$= t\mathbf{V}+e^{-\mathbf{A}(\mathbf{p})t}\{e^{\mathbf{A}(\mathbf{p})t}\mathbf{U}-\mathbf{U}e^{\mathbf{A}(\mathbf{p})t}\}, \tag{4.3.10}$$

the result follows directly.

∎

In establishing the existence and uniqueness of solutions (U,V) for equations (4.3.8a,b), we shall make use of the following result which can be found in [Tay.1].

Proposition 4.3.3 : Let $\mathcal{A}$ be a linear operator mapping a finite dimensional linear space $\mathcal{V}$ into itself and let $R(\mathcal{A})$ and $N(\mathcal{A})$ denote the range space and the null space of $\mathcal{A}$. Given any scalar λ, if q is the smallest nonnegative integer such that $N[(\mathcal{A}-\lambda)^q] = N[(\mathcal{A}-\lambda)^{q+1}]$, then

$$\mathcal{V} = N[(\mathcal{A}-\lambda I)^q]\oplus R[(\mathcal{A}-\lambda I)^q]. \tag{4.3.11}$$

∎

Corollary 4.3.4 : If $\mathcal{A}$ is diagonalizable, then $\mathcal{V} = R(\mathcal{A}) \oplus N(\mathcal{A})$.

Proof : If $\mathcal{A}$ is diagonalizable, then $N[(\mathcal{A}-\lambda I)] = N[(\mathcal{A}-\lambda I)^2]$ for any scalar λ.

∎

Proposition 4.3.5 : Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ and let the Lie bracket type operator $\mathcal{A}$: $\mathbb{R}^{n\times n} \rightarrow \mathbb{R}^{n\times n}$ be defined by

$$\mathcal{A}(\mathbf{X}) = [\mathbf{A},\mathbf{X}] \triangleq \mathbf{AX}-\mathbf{XA}. \tag{4.3.12}$$

Then $\mathbb{R}^{n\times n} = R(\mathcal{A}) \oplus N(\mathcal{A})$, if and only if $\mathbf{A}$ is diagonalizable.

Proof : "<=": Suppose that $\mathbf{A}$ is diagonalizable. Let $\lambda_1, \lambda_2, \cdots, \lambda_n$ be the eigenvalues of $\mathbf{A}$, let $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n$ be a set of linearly independent corresponding right eigenvectors of $\mathbf{A}$ and let $\mathbf{v}_1^T, \mathbf{v}_2^T, \cdots, \mathbf{v}_n^T$ be a set linearly independent of corresponding left eigenvectors of $\mathbf{A}$. Then (see [Gan.1]) $(\lambda_i-\lambda_j)$ for all $i,j \in \underline{n}$ is an eigenvalue of $\mathcal{A}$, and $\mathbf{u}_i\mathbf{v}_j^T$ is an eigenvector of $\mathcal{A}$ corresponding to $(\lambda_i-\lambda_j)$ for $i,j \in \underline{n}$. Since the set $\{ \mathbf{u}_i\mathbf{v}_j^T \mid i,j \in \underline{n} \}$ contains n^2 linearly independent eigenvectors, $\mathcal{A}$ is diagonalizable. By Corollary 4, $\mathbb{R}^{n\times n} = R(\mathcal{A}) \oplus N(\mathcal{A})$.

"=>": We give a proof by contraposition. Assume that $\mathbf{A}$ is not diagonalizable. Then, by the Jordan form theorem [Gan.1], there exists a nonsingular $n \times n$ matrix U such that

$$\mathbf{U}^{-1}\mathbf{A}\mathbf{U} = \begin{bmatrix} \mathbf{J}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{J}_2 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & \cdots & \mathbf{J}_\rho \end{bmatrix} = \mathbf{J} \tag{4.3.13}$$

where the $\mathbf{J}_i$ are $n_i \times n_i$ Jordan blocks associated with eigenvalue λ_i, and $n = \sum_{i=1}^{\rho} n_i$. If the columns of $\mathbf{U}$ are denoted by $\mathbf{u}_i$, so that $\mathbf{U} = [\mathbf{u}_1,\mathbf{u}_2, \cdots,\mathbf{u}_n]$, and the rows of $\mathbf{U}^{-1}$ are denoted by $\mathbf{v}_i^T$, so that $\mathbf{V} = [\, \mathbf{v}_1,\mathbf{v}_2, \cdots,\mathbf{v}_n \,]^T \triangleq \mathbf{U}^{-1}$, then $\mathbf{u}_1,\mathbf{u}_2, \cdots,\mathbf{u}_n$ are generalized right eigenvectors of $\mathbf{A}$ and $\mathbf{v}_1^T,\mathbf{v}_2^T, \cdots,\mathbf{v}_n^T$ are generalized left eigenvectors of $\mathbf{A}$, see [Gan.1]. Also the dyads in $\{\mathbf{u}_i\mathbf{v}_j^T \mid i, j \in \underline{n}\}$ form a basis for $\mathbb{R}^{n\times n}$. Because $\mathbf{J}$ is block diagonal, the equations $\mathbf{AU} = \mathbf{UJ}$ and $\mathbf{VA} = \mathbf{JV}$ decompose into equations of the form

$$\mathbf{A}\mathbf{u}_i = \lambda_i \mathbf{u}_i + \nu_i \mathbf{u}_{i-1}\,; \qquad \mathbf{v}_i^T\mathbf{A} = \lambda_i \mathbf{v}_i^T + \mu_i \mathbf{v}_{i+1}^T, \tag{4.3.14}$$

where ν_i, μ_i can have only the values 0 or 1. Consider the n_1 equations in (4.3.14) corresponding to the first Jordan block J_1:

$$\begin{aligned}
&\mathbf{A}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1\,; && \mathbf{v}_1^T\mathbf{A} = \lambda_1 \mathbf{v}_1^T + \mathbf{v}_2^T \\
&\mathbf{A}\mathbf{u}_2 = \lambda_1 \mathbf{u}_2 + \mathbf{u}_1\,; && \mathbf{v}_2^T\mathbf{A} = \lambda_1 \mathbf{v}_2^T + \mathbf{v}_3^T \\
&\quad\cdots && \quad\cdots \\
&\mathbf{A}\mathbf{u}_{n_1-1} = \lambda_1 \mathbf{u}_{n_1-1} + \mathbf{u}_{n_1-2}\,; && \mathbf{v}_{n_1-1}^T\mathbf{A} = \lambda_1 \mathbf{v}_{n_1-1}^T + \mathbf{v}_{n_1}^T \\
&\mathbf{A}\mathbf{u}_{n_1} = \lambda_1 \mathbf{u}_{n_1} + \mathbf{u}_{n_1-1}\,; && \mathbf{v}_{n_1}^T\mathbf{A} = \lambda_1 \mathbf{v}_{n_1}^T
\end{aligned} \tag{4.3.15}$$

Hence $\mathcal{A}(\mathbf{u}_1\mathbf{v}_{n_1}^T) = 0$ and $\mathcal{A}(\mathbf{u}_1\mathbf{v}_{n_1-1}^T) = -\mathbf{u}_1\mathbf{v}_{n_1}^T$. This implies that $\mathbf{u}_1\mathbf{v}_{n_1}^T \in N(\mathcal{A}) \bigcap R(\mathcal{A})$ and therefore that $\mathbb{R}^{n\times n} \neq N(\mathcal{A}) \oplus R(\mathcal{A})$.

∎

Corollary 4.3.6 : Let $\mathbf{A}\in\mathbb{R}^{n\times n}$ be diagonalizable, then for any $\mathbf{M}\in\mathbb{R}^{n\times n}$, there exist $\mathbf{V}, \mathbf{U} \in \mathbb{R}^{n\times n}$ such that

$$\mathbf{M} = \mathbf{V} + [\mathbf{A},\mathbf{U}]; \qquad [\mathbf{A},\mathbf{V}] = 0\,. \tag{4.3.16}$$

Furthermore, the solution $\mathbf{V}$ of $[\mathbf{A},\mathbf{V}] = 0$ in (4.3.16) is unique.

Proof : By Proposition 4.3.5, for any $\mathbf{M}\in\mathbb{R}^{n\times n}$ there exists $U\in\mathbf{R}^{n\times n}$ and $\mathbf{V}\in N(\mathcal{A})$ such that $\mathbf{M} = \mathbf{V} + \mathcal{A}(\mathbf{U})$. By the definition of direct sum, $\mathbf{V}$ and $\mathcal{A}(\mathbf{U})$ are unique.

∎

Corollary 4.3.7 : Let $\mathbf{A}(\mathbf{p})$ be defined as in equation (2.5.6a). If $\mathbf{A}(\mathbf{p})$ is diagonalizable for a given $\mathbf{p}\in\mathbb{R}^{n_p}$, then

$$\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i} = t\mathbf{V}e^{\mathbf{A}(\mathbf{p})t} + [e^{\mathbf{A}(\mathbf{p})t},\mathbf{U}] \tag{4.3.17}$$

for $i \in \underline{n_p}$, where $\mathbf{V}$, $\mathbf{U}$ satisfy following equations:

$$\frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i} = \mathbf{V} + [\mathbf{A}(\mathbf{p}),\mathbf{U}]\,; \qquad [\mathbf{A}(\mathbf{p}),\mathbf{V}] = 0\,. \tag{4.3.18}$$

Proof : The proof follows directly from the results of Proposition 4.3.2 and Corollary 4.3.6.

∎

4.3.2. Procedures for Computing the Matrix $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$ via Diagonalization

We shall now state two procedures, based on (4.3.17) and Corollaries 4.3.6 and 4.3.7, for computing $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$ when $\mathbf{A}(\mathbf{p})$ is an $n_s \times n_s$, continuously differentiable, diagonalizable matrix. Since $\mathbf{A}(\mathbf{p})$ is diagonalizable, there exists a matrix $\mathbf{T}(\mathbf{p}) \in \mathbb{C}^{n_s \times n_s}$ such that

$$\Lambda(\mathbf{p}) = \mathbf{T}^{-1}(\mathbf{p})\mathbf{A}(\mathbf{p})\mathbf{T}(\mathbf{p}), \tag{4.3.19}$$

where

$$\Lambda(\mathbf{p}) \triangleq \text{diag}\{ \lambda_1(\mathbf{p}), \lambda_2(\mathbf{p}), \cdots, \lambda_{n_s}(\mathbf{p}) \}, \tag{4.3.20}$$

and $\lambda_j(\mathbf{p}) \in \mathbb{C}$ for all $j \in \underline{n_s}$. Since the entries of $\mathbf{T}(\mathbf{p})$ and the eigenvalues $\lambda_i(\mathbf{p})$ are complex numbers, the following procedure will be based on complex arithmetic. To simplify the notation, we denote $\frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i}$ by $\mathbf{M}(\mathbf{p})$ and denote n_s by n.

Algorithm 4.3.1 : (*Complex Arithmetic Version*)

Data: A continuously differentiable, diagonalizable $n \times n$ matrix $\mathbf{A}(\mathbf{p})$.

Step 1: Diagonalize $\mathbf{A}(\mathbf{p})$ by computing a matrix of linearly independent eigenvectors $\mathbf{T}(\mathbf{p}) \in \mathbb{C}^{n \times n}$ such that the equations (4.3.19) and (4.3.20) hold.

Step 2: Compute $\widetilde{\mathbf{M}}(\mathbf{p}) = \mathbf{T}^{-1}(\mathbf{p})M(\mathbf{p})\mathbf{T}(\mathbf{p})$.

Step 3: Construct the matrices $\widetilde{\mathbf{U}}(\mathbf{p}), \widetilde{\mathbf{V}}(\mathbf{p}) \in \mathbb{C}^{n \times n}$ as follows:

$$\tilde{v}_{i,j}(\mathbf{p}) = \begin{cases} \tilde{m}_{i,j}(\mathbf{p}) & \text{if } \lambda_i(\mathbf{p}) = \lambda_j(\mathbf{p}) \\ 0 & \text{otherwise} \end{cases} \tag{4.3.21}$$

$$\tilde{u}_{i,j}(\mathbf{p}) = \begin{cases} \dfrac{\tilde{m}_{i,j}(\mathbf{p})}{\lambda_i(\mathbf{p}) - \lambda_j(\mathbf{p})} & \text{if } \lambda_i(\mathbf{p}) \neq \lambda_j(\mathbf{p}) \\ 0 & \text{otherwise} \end{cases} \tag{4.3.22}$$

for all $i,j \in \underline{n}$.

Comment: Note that $[\Lambda,\widetilde{\mathbf{V}}\,] = 0$, and

$$\{\widetilde{\mathbf{V}} + [\Lambda,\widetilde{\mathbf{U}}\,]\}_{i,j} = \tilde{v}_{i,j} + \lambda_i \tilde{u}_{i,j} - \tilde{u}_{i,j}\lambda_j = \begin{cases} \tilde{v}_{i,j} & \text{if } \lambda_i = \lambda_j \\ \tilde{m}_{i,j} & \text{if } \lambda_i \neq \lambda_j \end{cases} \quad (4.3.23)$$

Hence $\widetilde{\mathbf{M}} = \widetilde{\mathbf{V}} + [\Lambda,\widetilde{\mathbf{U}}\,]$.

Step 4: Compute $V(\mathbf{p}) = \mathbf{T}(\mathbf{p})\widetilde{\mathbf{V}}(\mathbf{p})\mathbf{T}^{-1}(\mathbf{p})$ and $U(\mathbf{p}) = \mathbf{T}(\mathbf{p})\widetilde{\mathbf{U}}(\mathbf{p})\mathbf{T}^{-1}(\mathbf{p})$, which solve equations (2.16), and set

$$\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i} = t\mathbf{V}(\mathbf{p})e^{\mathbf{A}(\mathbf{p})t} + [e^{\mathbf{A}(\mathbf{p})t},\mathbf{U}(\mathbf{p})]\,. \quad (4.3.24)$$

If the sensitivity of the diagonalized system is required, set

$$\mathbf{T}^{-1}(\mathbf{p})\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}T(\mathbf{p}) = t\widetilde{\mathbf{V}}(\mathbf{p})e^{\mathbf{A}(\mathbf{p})t} + [e^{\mathbf{A}(\mathbf{p})t},\widetilde{\mathbf{U}}(\mathbf{p})]\,. \quad (4.3.25)$$

∎

In order to avoid complex arithmetic, a block diagonal form of $\Lambda(\mathbf{p})$ with n_r scalar diagonal blocks and $n_\omega = (n-n_r)/2$ two by two diagonal blocks will be used, where n_r is the number of real eigenvalues of $\mathbf{A}(\mathbf{p})$; the (2×2) blocks represent complex conjugate pairs of eigenvalues. Let λ_i, $i \in \underline{n_r}$, denote the real eigenvalues, and let $\sigma_i \pm j\omega_i$, $i \in \underline{n_\omega}$, denote the complex conjugate pairs of eigenvalues, then there exists a matrix $\mathbf{T}(\mathbf{p}) \in \mathbb{R}^{n\times n}$ such that $\Lambda(\mathbf{p}) = \mathbf{T}^{-1}(\mathbf{p})\mathbf{A}(\mathbf{p})\mathbf{T}(\mathbf{p})$, where

$$\Lambda(\mathbf{p}) \triangleq diag\{\,\lambda_1(\mathbf{p}), \lambda_2(\mathbf{p}), \cdots, \lambda_{n_r}(\mathbf{p}), \Psi_1(\mathbf{p}), \Psi_2(\mathbf{p}), \cdots, \Psi_{n_\omega}(\mathbf{p})\}\,, \quad (4.3.26)$$

and $\Psi_i(\mathbf{p}) \triangleq \begin{bmatrix} \sigma_i(\mathbf{p}) & \omega_i(\mathbf{p}) \\ -\omega_i(\mathbf{p}) & \sigma_i(\mathbf{p}) \end{bmatrix}$, for all $i \in \underline{n_\omega}$.

We now establish the following lemmas which lead to the real arithmetic version of Algorithm 4.3.1.

Lemma 4.3.8 : Let $\sigma,\omega \in \mathbb{R}$ and $\omega \neq 0$. For any $\mathbf{M} \triangleq \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \in \mathbb{R}^{2\times2}$, the (U, V) matrix pair given by

$$\mathbf{U} \triangleq \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} = \frac{1}{4\omega}\begin{bmatrix} -m_{12}-m_{21} & m_{11}-m_{22} \\ m_{11}-m_{22} & m_{12}+m_{21} \end{bmatrix}, \quad (4.3.27)$$

$$\mathbf{V} \triangleq \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} m_{11}+m_{22} & m_{12}-m_{21} \\ m_{21}-m_{12} & m_{11}+m_{22} \end{bmatrix}, \tag{4.3.28}$$

is a solution of the following equations

$$\begin{bmatrix} \sigma & \omega \\ -\omega & \sigma \end{bmatrix}\cdot\mathbf{U}-\mathbf{U}\cdot\begin{bmatrix} \sigma & \omega \\ -\omega & \sigma \end{bmatrix}+\mathbf{V}=\mathbf{M}, \tag{4.3.29a}$$

$$\begin{bmatrix} \sigma & \omega \\ -\omega & \sigma \end{bmatrix}\cdot\mathbf{V}-\mathbf{V}\cdot\begin{bmatrix} \sigma & \omega \\ -\omega & \sigma \end{bmatrix}=0. \tag{4.3.29b}$$

■

Lemma 4.3.9 : Let $\sigma_1,\sigma_2,\omega_1,\omega_2 \in \mathbb{R}$ and $\omega_1 \neq \omega_2$, $\sigma_1 \neq \sigma_2$. For any $\mathbf{M} \triangleq \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \in \mathbb{R}^{2\times 2}$, the matrix $\mathbf{U} \triangleq \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \in \mathbb{R}^{2\times 2}$ given by

$$\begin{aligned} u_{11} &= [\,\alpha m_{11}-\beta m_{12}-\gamma m_{21}+\psi m_{22}\,]/d\,, \\ u_{12} &= [\,\beta m_{11}+\alpha m_{12}-\psi m_{21}-\gamma m_{22}\,]/d\,, \\ u_{21} &= [\,\gamma m_{11}-\psi m_{12}+\alpha m_{21}-\beta m_{22}\,]/d\,, \\ u_{22} &= [\,\psi m_{11}+\gamma m_{12}+\beta m_{21}+\alpha m_{22}\,]/d\,, \end{aligned} \tag{4.3.30}$$

is the solution of the following equation

$$\begin{bmatrix} \sigma_1 & \omega_1 \\ -\omega_1 & \sigma_1 \end{bmatrix}\cdot\mathbf{U}-\mathbf{U}\cdot\begin{bmatrix} \sigma_2 & \omega_2 \\ -\omega_2 & \sigma_2 \end{bmatrix}=\mathbf{M}, \tag{4.3.31}$$

where

$$\begin{aligned} &\alpha = (\sigma_1-\sigma_2)[(\sigma_1-\sigma_2)^2+\omega_1^2+\omega_2^2]\,; && \beta = \omega_2[(\sigma_1-\sigma_2)^2-\omega_1^2+\omega_2^2]\,; \\ &\gamma = \omega_1[(\sigma_1-\sigma_2)^2+\omega_1^2-\omega_2^2]\,; && \psi = 2(\sigma_1-\sigma_2)\omega_1\omega_2\,; \\ &d = (\sigma_1-\sigma_2)\alpha+\omega_2\beta+\omega_1\gamma\,. \end{aligned} \tag{4.3.32}$$

Proof : The equation (4.3.31) can be reduced to the following equation

$$\begin{bmatrix} \sigma_1-\sigma_2 & \omega_2 & \omega_1 & 0 \\ -\omega_2 & \sigma_1-\sigma_2 & 0 & \omega_1 \\ -\omega_1 & 0 & \sigma_1-\sigma_2 & \omega_2 \\ 0 & -\omega_1 & -\omega_2 & \sigma_1-\sigma_2 \end{bmatrix}\begin{bmatrix} u_{11} \\ u_{12} \\ u_{21} \\ u_{22} \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{12} \\ m_{21} \\ m_{22} \end{bmatrix}. \tag{4.3.33}$$

Then, the solution of the equation (4.3.31) can be obtained by solving the equation (4.3.33).

∎

Lemma 4.3.10 : Let $\lambda,\sigma,\omega \in \mathbb{R}$ and $\omega \neq 0$, $\lambda \neq \sigma$.

(i) For any $\mathbf{M} \triangleq [\, m_{11}\ m_{12}] \in \mathbb{R}^{1\times 2}$, the matrix $\mathbf{U} \triangleq [\, u_{11}\ u_{12}] \in \mathbb{R}^{1\times 2}$ defined by

$$\mathbf{U} = \frac{1}{(\lambda-\sigma)^2+\omega^2}\cdot\mathbf{M}\cdot\begin{bmatrix}\lambda-\sigma & \omega\\ -\omega & \lambda-\sigma\end{bmatrix}, \tag{4.3.34}$$

is the solution of the equation

$$\lambda\cdot\mathbf{U}-\mathbf{U}\cdot\begin{bmatrix}\sigma & \omega\\ -\omega & \sigma\end{bmatrix}=\mathbf{M}. \tag{4.3.35}$$

(ii) For any $\mathbf{M} \triangleq [\, m_{11}\ m_{21}]^{\mathrm{T}} \in \mathbb{R}^{2\times 1}$, the matrix $\mathbf{U} \triangleq [\, u_{11}\ u_{21}]^{T} \in \mathbb{R}^{2\times 1}$ defined by

$$\mathbf{U} = \frac{1}{(\sigma-\lambda)^2+\omega^2}\begin{bmatrix}\sigma-\lambda & -\omega\\ \omega & \sigma-\lambda\end{bmatrix}\cdot\mathbf{M}, \tag{4.3.36}$$

is the solution of the following equation

$$\begin{bmatrix}\sigma & \omega\\ -\omega & \sigma\end{bmatrix}\cdot\mathbf{U}-\mathbf{U}\cdot\lambda=\mathbf{M}, \tag{4.3.37}$$

∎

We now are ready to present the procedure based on real arithmetic.

Algorithm 4.3.2 : (*Real Arithmetic Version*)

Data: A continuously differentiable, diagonalizable $n\times n$ matrix $\mathbf{A(p)}$.

Step 1: Reduce $\mathbf{A(p)}$ to block diagonal form by computing a matrix of linearly independent eigenvectors $\mathbf{T(p)}\in\mathbb{R}^{n\times n}$ such that the equations (4.3.19) and (4.3.26) hold.

Step 2: Compute $\widetilde{\mathbf{M}}(\mathbf{p}) = \mathbf{T}^{-1}(\mathbf{p})M(\mathbf{p})\mathbf{T}(\mathbf{p})$.

Step 3: Construct the matrices $\widetilde{\mathbf{U}}(\mathbf{p}), \widetilde{\mathbf{V}}(\mathbf{p}) \in \mathbb{R}^{n\times n}$ as follows:

(i) When $i,j \leq n_r$, set

$$\tilde{v}_{i,j}(\mathbf{p}) = \begin{cases} \tilde{m}_{i,j}(\mathbf{p}) & \text{if } \lambda_i(\mathbf{p}) = \lambda_j(\mathbf{p}) \\ 0 & \text{otherwise} \end{cases} \tag{4.3.38}$$

$$\tilde{u}_{i,j}(\mathbf{p}) = \begin{cases} \dfrac{\tilde{m}_{i,j}(\mathbf{p})}{\lambda_i(\mathbf{p})-\lambda_j(\mathbf{p})} & \text{if } \lambda_i(\mathbf{p}) \neq \lambda_j(\mathbf{p}) \\ 0 & \text{otherwise} \end{cases} \tag{4.3.39}$$

(ii) When $n_r < i,j \leq n$, let $i=n_r+2k-1$ and $j=n_r+2l-1$ where $k,l \in \underline{n_\omega}$.

If $\Psi_k(\mathbf{p}) = \Psi_l(\mathbf{p})$, we may use the result from Lemma 4.3.8 and set

$$\begin{bmatrix} \tilde{v}_{i,j} & \tilde{v}_{i,j+1} \\ \tilde{v}_{i+1,j} & \tilde{v}_{i+1,j+1} \end{bmatrix}(\mathbf{p}) = \frac{1}{2}\begin{bmatrix} \tilde{m}_{i,j}+\tilde{m}_{i+1,j+1} & \tilde{m}_{i,j+1}-\tilde{m}_{i+1,j} \\ \tilde{m}_{i+1,j}-\tilde{m}_{i,j+1} & \tilde{m}_{i,j}+\tilde{m}_{i+1,j+1} \end{bmatrix}(\mathbf{p}), \tag{4.3.40}$$

$$\begin{bmatrix} \tilde{u}_{i,j} & \tilde{u}_{i,j+1} \\ \tilde{u}_{i+1,j} & \tilde{u}_{i+1,j+1} \end{bmatrix}(\mathbf{p}) = \frac{1}{4\omega_k(\mathbf{p})}\begin{bmatrix} -\tilde{m}_{i,j+1}-\tilde{m}_{i+1,j} & \tilde{m}_{i,j}-\tilde{m}_{i+1,j+1} \\ \tilde{m}_{i,j}-\tilde{m}_{i+1,j+1} & \tilde{m}_{i,j+1}+\tilde{m}_{i+1,j} \end{bmatrix}(\mathbf{p}) \tag{4.3.41}$$

If $\Psi_k(\mathbf{p}) \neq \Psi_l(\mathbf{p})$, (from Lemma 4.3.9) compute

$$\begin{aligned} \alpha &= [\sigma_k(\mathbf{p})-\sigma_l(\mathbf{p})]\{[\sigma_k(\mathbf{p})-\sigma_l(\mathbf{p})]^2+\omega_k^2(\mathbf{p})+\omega_l^2(\mathbf{p})\}, \\ \beta &= \omega_l(\mathbf{p})\{[\sigma_k(\mathbf{p})-\sigma_l(\mathbf{p})]^2-\omega_k^2(\mathbf{p})+\omega_l^2(\mathbf{p})\}, \\ \gamma &= \omega_k(\mathbf{p})\{[\sigma_k(\mathbf{p})-\sigma_l(\mathbf{p})]^2+\omega_k^2(\mathbf{p})-\omega_l^2(\mathbf{p})\}, \\ \psi &= 2[\sigma_k(\mathbf{p})-\sigma_l(\mathbf{p})]\omega_k(\mathbf{p})\omega_l(\mathbf{p}), \\ d &= [\sigma_k(\mathbf{p})-\sigma_l(\mathbf{p})]\alpha+\omega_l(\mathbf{p})\beta+\omega_k(\mathbf{p})\gamma. \end{aligned} \tag{4.3.42}$$

and set

$$\tilde{v}_{i,j}(\mathbf{p}) = \tilde{v}_{i+1,j}(\mathbf{p}) = \tilde{v}_{i,j+1}(\mathbf{p}) = \tilde{v}_{i+1,j+1}(\mathbf{p}) = 0, \tag{4.3.43}$$

$$\begin{bmatrix} \tilde{u}_{i,j}(\mathbf{p}) \\ \tilde{u}_{i,j+1}(\mathbf{p}) \\ \tilde{u}_{i+1,j}(\mathbf{p}) \\ \tilde{u}_{i+1,j+1}(\mathbf{p}) \end{bmatrix} = \frac{1}{d}\begin{bmatrix} \alpha & -\beta & -\gamma & \psi \\ \beta & \alpha & -\psi & -\gamma \\ \gamma & -\psi & \alpha & -\beta \\ \psi & \gamma & \beta & \alpha \end{bmatrix}\begin{bmatrix} \tilde{m}_{i,j}(\mathbf{p}) \\ \tilde{m}_{i,j+1}(\mathbf{p}) \\ \tilde{m}_{i+1,j}(\mathbf{p}) \\ \tilde{m}_{i+1,j+1}(\mathbf{p}) \end{bmatrix}. \tag{4.3.44}$$

(iii) When $i \leq n_r$ and $n_r < j \leq n$, let $j = n_r + 2l - 1$ where $l \in \underline{n_\omega}$, compute

$$d = 1/\{[\lambda_i(\mathbf{p}) - \sigma_l(\mathbf{p})]^2 + \omega_l^2(\mathbf{p})\}, \tag{4.3.45}$$

and (from Lemma 4.3.10(i)) set

$$\tilde{v}_{i,j}(\mathbf{p}) = \tilde{v}_{i,j+1}(\mathbf{p}) = 0, \tag{4.3.46}$$

$$[\tilde{u}_{i,j}(\mathbf{p})\ \tilde{u}_{i,j+1}(\mathbf{p})] = \frac{1}{d}[\tilde{m}_{i,j}(\mathbf{p})\ \tilde{m}_{i,j+1}(\mathbf{p})] \begin{bmatrix} \lambda_i(\mathbf{p}) - \sigma_l(\mathbf{p}) & \omega_l(\mathbf{p}) \\ -\omega_l(\mathbf{p}) & \lambda_i(\mathbf{p}) - \sigma_l(\mathbf{p}) \end{bmatrix} \tag{4.3.47}$$

(iv) When $n_r < i \leq n$ and $j \leq n_r$, let $i = n_r + 2k - 1$ where $k \in \underline{n_\omega}$, compute

$$d = 1/\{[\sigma_k(\mathbf{p}) - \lambda_j(\mathbf{p})]^2 + \omega_k^2(\mathbf{p})\}, \tag{4.3.48}$$

and (from Lemma 4.3.10(ii)) set

$$\tilde{v}_{i,j}(\mathbf{p}) = \tilde{v}_{i+1,j}(\mathbf{p}) = 0, \tag{4.3.49}$$

$$\begin{bmatrix} \tilde{u}_{i,j}(\mathbf{p}) \\ \tilde{u}_{i+1,j}(\mathbf{p}) \end{bmatrix} = \frac{1}{d} \begin{bmatrix} \sigma_k(\mathbf{p}) - \lambda_j(\mathbf{p}) & -\omega_k(\mathbf{p}) \\ \omega_k(\mathbf{p}) & \sigma_k(\mathbf{p}) - \lambda_j(\mathbf{p}) \end{bmatrix} \begin{bmatrix} \tilde{m}_{i,j}(\mathbf{p}) \\ \tilde{m}_{i+1,j}(\mathbf{p}) \end{bmatrix}. \tag{4.3.50}$$

Step 4: Compute $\mathbf{V}(\mathbf{p}) = \mathbf{T}(\mathbf{p})\tilde{\mathbf{V}}(\mathbf{p})\mathbf{T}^{-1}(\mathbf{p})$ and $\mathbf{U}(\mathbf{p}) = \mathbf{T}(\mathbf{p})\tilde{\mathbf{U}}(\mathbf{p})\mathbf{T}^{-1}(\mathbf{p})$, which solve equations (2.16), and set

$$\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i} = t\mathbf{V}(\mathbf{p})e^{\mathbf{A}(\mathbf{p})t} + [e^{\mathbf{A}(\mathbf{p})t}, \mathbf{U}(\mathbf{p})]. \tag{4.3.51}$$

If the sensitivity of the diagonalized system is required, set

$$\mathbf{T}^{-1}(\mathbf{p})\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}T(\mathbf{p}) = t\tilde{\mathbf{V}}(\mathbf{p})e^{\mathbf{A}(\mathbf{p})t} + [e^{\mathbf{A}(\mathbf{p})t}, \tilde{\mathbf{U}}(\mathbf{p})]. \tag{4.3.52}$$

■

4.3.3 An Efficient Procedure for Computing the Matrix $e^{\mathbf{A}(\mathbf{p})t}$

In principle, the exponential of a matrix can be computed in many ways, such as methods involving approximation theory, differential equations, the matrix eigenvalues, the matrix characteristic polynomial, etc. An excellent survey of computational techniques for the exponential of a matrix was given in [Mol.1]. In practice, consideration of computational stability and efficiency indicates that none is

completely satisfactory.

For our purposes, we are interested not only in the computation of the matrix $e^{\mathbf{A}(\mathbf{p})t}$, but also in the computation of the matrix $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$. If we assume that the matrix $\mathbf{A}(\mathbf{p})$ is diagonalizable, a closed form for $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$ can be written in terms of $e^{\mathbf{A}(\mathbf{p})t}$, see the equations (4.3.17), (4.3.25) and (4.3.52). Therefore the diagonalization method for the computation of $e^{\mathbf{A}(\mathbf{p})t}$, given by

$$e^{\mathbf{A}(\mathbf{p})t} = \mathbf{T}(\mathbf{p})e^{\mathbf{\Lambda}(\mathbf{p})t}\mathbf{T}^{-1}(\mathbf{p}), \tag{4.3.53}$$

is most attractive. It is clear that $e^{\mathbf{\Lambda}(\mathbf{p})t}$ can be obtained easily. Hence the equation (4.3.53) also shows that the diagonalization method is most efficient for problems involving large matrices $\mathbf{A}(\mathbf{p})$ or repeated evaluation of $e^{\mathbf{A}(\mathbf{p})t}$ for many different values of t.

However, the diagonalization method is unstable when the eigenstructure of $\mathbf{A}(\mathbf{p})$ is defective or near defective. We have discussed in the section 4.2.1 that we may use the condition number, $cond(\mathbf{T})$, of the matrix $\mathbf{T}$ as a testing function for the defectiveness of $\mathbf{A}(\mathbf{p})$. If $\mathbf{A}(\mathbf{p})$ is nearly (exactly) defective, then the $cond(\mathbf{T})$ is large (infinite). Any errors in $\mathbf{A}(\mathbf{p})$, including roundoff errors in its computation and roundoff errors from the eigenvalue computation, may be magnified in the final result of the decomposition $\mathbf{A}(\mathbf{p}) = \mathbf{T}(\mathbf{p})\mathbf{\Lambda}(\mathbf{p})\mathbf{T}^{-1}(\mathbf{p})$ by $cond(\mathbf{T})$. Consequently, when $cond(\mathbf{T})$ is large, the computed $e^{\mathbf{A}(\mathbf{p})t}$ and $\frac{\partial e^{\mathbf{A}(\mathbf{p})t}}{\partial \mathbf{p}^i}$ will most likely be inaccurate. For consideration of accuracy and stability, it has been shown in [Mol.1, Par.1] that the diagonalization method is acceptable when $cond(\mathbf{T}) \lesssim 100$.

There is still debate as to what is the most effective numerical algorithm, particularly when $\mathbf{A}(\mathbf{p})$ is defective or near-defective. If a very accurate solution for the matrix exponential function is required, the method based-on the Schur transformation will be most attractive when $cond(\mathbf{T}) \gtrsim 100$, see reference [Par.1]. Two efficient algorithms based on the Schur form have been implemented [Par.1, Wra.1]. The first algorithm is due to B. N. Parlett and it is based on a simple recurrence that permits

the computation of the off diagonal blocks once the diagonal blocks of e^{St} are known, see [Par.1, Par.2]. The second algorithm, due to G. W. Stewart, uses extra similarity transforms to reduce S to Schur block diagonal form with as many blocks as is consistent with keeping the condition number of the transformed matrices bounded , see [Bav.1, Mol.1, Wra.1]. In this dissertation, the *Schur form method* will mean either one of these two stable methods.

As mentioned in Section 4.2.1 the Schur form $\mathbf{S}_c(\mathbf{p}_c)$ of the matrix $\mathbf{A}_c(\mathbf{p}_c)$ can be obtained automatically in the process of diagonalization. Note that the matrix $\mathbf{A}(\mathbf{p})$ described in (2.5.7) is different from the matrix $\mathbf{A}_c(\mathbf{p}_c)$ described in (2.4.5a). A combined algorithm of the diagonalization method and the Schur form method for computing the matrix exponential $e^{\mathbf{A}(\mathbf{p})t}$ is given as follows:

Algorithm 4.3.3 : (*Computation of the Exponential of a Matrix*)

Data: $A \in \mathbb{R}^{n\times n}$ and $\kappa >> 0$, an upper bound on condition number.

Step 1: Reduce matrix **A** to (real) Schur form **S** by computing a (orthogonal) unitary transformation **Q** so that

$$\mathbf{S} = \mathbf{Q}^*\mathbf{A}\mathbf{Q}\,. \tag{4.3.54}$$

If the QR method fails to converge, i.e. the Schur form cannot be computed, then stop and exit.

Step 2: Compute an eigenvector matrix **R** of the upper triangular matrix **S** by back substitutions:

$$\mathbf{R}\Lambda = \mathbf{S}\mathbf{R}\,. \tag{4.3.55}$$

Step 3: Given **x**, estimate *cond*(**R**) by solving

$$\mathbf{R}\mathbf{y} = \mathbf{Q}^*\mathbf{x}\,. \tag{4.3.56}$$

Step 4: If $cond(\mathbf{R}) > \kappa$, then compute

$$e^{\mathbf{A}t} = \mathbf{Q}e^{\mathbf{S}t}\mathbf{Q}^*, \tag{4.3.57}$$

by the Schur form method.

If $cond(\mathbf{R}) \leq \kappa$, then set $\mathbf{T} = \mathbf{QR}$ and compute $\mathbf{T}^{-1} = \mathbf{R}^{-1}\mathbf{Q}^*$ by solving

$$\mathbf{RT}^{-1} = \mathbf{Q}^*\mathbf{I}. \tag{4.3.58}$$

Then we compute

$$e^{\mathbf{A}t} = \mathbf{T}e^{\Lambda t}\mathbf{T}^{-1}. \tag{4.3.59}$$

Stop and exit.

∎

Comment: Since $\mathbf{Q}$ is a unitary (orthogonal) matrix, $cond(\mathbf{T})$=$cond(\mathbf{R})$.

Note that the computation of the transformation matrices $\mathbf{Q}$ and $\mathbf{R}$ will be only done by once. The equations (4.3.57) and (4.3.59) can be evaluated at many different values of t.

4.3.4 State Response and Its Sensitivities via Diagonalization

Based on the result of Section 4.3.1, an efficient scheme for computing $\mathbf{x}(t,\mathbf{p})$ and $\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ can be established if $\mathbf{A}(\mathbf{p})$ is diagonalizable for a given parameter $\mathbf{p}$. In this section, we will assume that $\mathbf{A}(\mathbf{p})$ is diagonalizable for a given $\mathbf{p} \in \mathbb{R}^{n_p}$. That is, there exist matrices $\mathbf{T}(\mathbf{p})$ and $\mathbf{T}^{-1}(\mathbf{p})$ such that $\Lambda(\mathbf{p}) = \mathbf{T}^{-1}(\mathbf{p})\mathbf{A}(\mathbf{p})\mathbf{T}(\mathbf{p})$ where $\Lambda(\mathbf{p}) = \text{diag}\{\lambda_1(\mathbf{p}), \lambda_2(\mathbf{p}), \cdots, \lambda_n(\mathbf{p})\}$.

In order to avoid complex arithmetic, the diagonal form of $\Lambda(\mathbf{p})$ will not actually be used. Instead the block diagonal form described in the equation (4.3.26) with n_r scalar diagonal blocks and n_ω=$(n-n_r)/2$ two by two diagonal blocks will be used, where n_r is the number of real eigenvalues of $\mathbf{A}(\mathbf{p})$; n_ω is the number of the (2×2) blocks which represent complex conjugate pairs of eigenvalues. Therefore, there exists a matrix $\mathbf{T}(\mathbf{p}) \in \mathbb{R}^{n\times n}$ such that $\Lambda(\mathbf{p}) = \mathbf{T}^{-1}(\mathbf{p})\mathbf{A}(\mathbf{p})\mathbf{T}(\mathbf{p})$, where

$$\Lambda(\mathbf{p}) \triangleq \text{diag}\{\lambda_1(\mathbf{p}), \lambda_2(\mathbf{p}), \cdots, \lambda_{n_r}(\mathbf{p}), \Psi_1(\mathbf{p}), \Psi_2(\mathbf{p}), \cdots, \Psi_{n_\omega}(\mathbf{p})\}, \tag{4.3.60}$$

and $\Psi_i(\mathbf{p}) \triangleq \begin{bmatrix} \sigma_i(\mathbf{p}) & \omega_i(\mathbf{p}) \\ -\omega_i(\mathbf{p}) & \sigma_i(\mathbf{p}) \end{bmatrix}$, for all $i \in \underline{n_\omega}$.

Suppose that $\tilde{\mathbf{x}}(t,\mathbf{p}) \triangleq \mathbf{T}^{-1}(\mathbf{p})\mathbf{x}(t,\mathbf{p})$ so that $\tilde{\mathbf{x}}(0,\mathbf{p}) \triangleq \mathbf{T}^{-1}(\mathbf{p})\mathbf{x}(0,\mathbf{p})$. From (4.3.1) and (2.5.2), the state vector can be evaluated as follows:

$$\begin{aligned}\tilde{\mathbf{x}}(t,\mathbf{p}) &= e^{\Lambda(\mathbf{p})t}\tilde{\mathbf{x}}(0,\mathbf{p}) + \int_0^t e^{\Lambda(\mathbf{p})(t-\tau)}\mathbf{T}^{-1}\mathbf{B}(\mathbf{p})\mathbf{r}(\tau,\mathbf{p})d\tau \\ &= e^{\Lambda(\mathbf{p})t}\tilde{\mathbf{x}}(0,\mathbf{p}) + \sum_{k=0}^{m}\left\{\left[\int_0^t \tau^k e^{\Lambda(\mathbf{p})(t-\tau)}d\tau\right]\mathbf{T}^{-1}\mathbf{B}(\mathbf{p})\mathbf{c}_k(\mathbf{p})\right\} \qquad (4.3.61)\end{aligned}$$

A closed-form expression for $\int_0^t \tau^k e^{\Lambda(\mathbf{p})\tau}d\tau$ is easily obtained element by element by using the following recursive formulas:

(i) For real eigenvalues,

$$\int_0^t \tau^k e^{\lambda\tau}d\tau = \begin{cases} \dfrac{1}{\lambda}(e^{\lambda t}-1)\,; & k=0, \\ \dfrac{t^k e^{\lambda t}}{\lambda} - \dfrac{k}{\lambda}\displaystyle\int_0^t \tau^{k-1}e^{\lambda\tau}d\tau\,; & k>0. \end{cases} \qquad (4.3.62)$$

(ii) For complex eigenvalue pairs, since

$$e^{\begin{bmatrix}\sigma & \omega \\ -\omega & \sigma\end{bmatrix}t} = \begin{bmatrix} e^{\sigma t}\cos\omega t & e^{\sigma t}\sin\omega t \\ -e^{\sigma t}\sin\omega t & e^{\sigma t}\cos\omega t \end{bmatrix}. \qquad (4.3.63)$$

We obtain that if $k=0$, then

$$\int_0^t \tau^k e^{\sigma\tau}\cos\omega\tau d\tau = \frac{e^{\sigma t}(\sigma\cos\omega t + \omega\sin\omega t)}{\sigma^2+\omega^2} - \frac{\sigma}{\sigma^2+\omega^2}, \qquad (4.3.64a)$$

$$\int_0^t \tau^k e^{\sigma\tau}\sin\omega\tau d\tau = \frac{e^{\sigma t}(\sigma\sin\omega t - \omega\cos\omega t)}{\sigma^2+\omega^2} + \frac{\omega}{\sigma^2+\omega^2}. \qquad (4.3.64b)$$

If $k>0$, then

$$\begin{aligned}\int_0^t \tau^k e^{\sigma\tau}\cos\omega\tau d\tau &= \frac{t^k e^{\sigma t}(\sigma\cos\omega t + \omega\sin\omega t)}{\sigma^2+\omega^2} \\ &\quad - \frac{\sigma k}{\sigma^2+\omega^2}\int_0^t \tau^{k-1}e^{\sigma\tau}\cos\omega\tau d\tau \qquad (4.3.65a) \\ &\quad - \frac{\omega k}{\sigma^2+\omega^2}\int_0^t \tau^{k-1}e^{\sigma\tau}\sin\omega\tau d\tau,\end{aligned}$$

and

$$\int_0^t \tau^k e^{\sigma\tau}\sin\omega\tau d\tau = \frac{t^k e^{\sigma t}(\sigma\sin\omega t - \omega\cos\omega t)}{\sigma^2+\omega^2}$$

$$- \frac{\sigma k}{\sigma^2+\omega^2}\int_0^t \tau^{k-1} e^{\sigma\tau}\sin\omega\tau d\tau \qquad (4.3.65b)$$

$$+ \frac{\omega k}{\sigma^2+\omega^2}\int_0^t \tau^{k-1} e^{\sigma\tau}\cos\omega\tau d\tau .$$

Therefore, $\mathbf{x}(t,\mathbf{p})$ can be evaluated by using (4.3.61) and the recursive formulae (4.3.62), (4.3.63), (4.3.64a,b) and (4.3.65a,b).

Suppose that $\frac{\Delta\tilde{\mathbf{x}}(t,\mathbf{p})}{\Delta\mathbf{p}^i} \triangleq \mathbf{T}^{-1}(\mathbf{p})\frac{\partial\mathbf{x}(t,\mathbf{p})}{\partial\mathbf{p}^i}$ and $\frac{\Delta\tilde{\mathbf{x}}(0,\mathbf{p})}{\Delta\mathbf{p}^i} \triangleq \mathbf{T}^{-1}(\mathbf{p})\frac{\partial\mathbf{x}(0,\mathbf{p})}{\partial\mathbf{p}^i}$. By equation (4.3.2) and (4.3.17), the state sensitivities can be evaluated as follows:

$$\begin{aligned}\frac{\Delta\tilde{\mathbf{x}}(t,\mathbf{p})}{\Delta\mathbf{p}^i} &= t\tilde{\mathbf{V}}e^{\Lambda(\mathbf{p})t}\tilde{\mathbf{x}}(0,\mathbf{p}) + [e^{\Lambda(\mathbf{p})t},\tilde{\mathbf{U}}]\tilde{\mathbf{x}}(0,\mathbf{p}) + e^{\Lambda(\mathbf{p})t}\frac{\Delta\tilde{\mathbf{x}}(0,\mathbf{p})}{\Delta\mathbf{p}^i} \\ &+ \tilde{\mathbf{V}}\sum_{k=0}^{m}\left\{[\int_0^t (t-\tau)e^{\Lambda(\mathbf{p})(t-\tau)}\tau^k d\tau]\mathbf{T}^{-1}\mathbf{B}(\mathbf{p})\mathbf{c}_k(\mathbf{p})\right\} \\ &+ \sum_{k=0}^{m}\left\{[\int_0^t \tau^k e^{\Lambda(\mathbf{p})(t-\tau)}d\tau]\tilde{\mathbf{U}}\mathbf{T}^{-1}\mathbf{B}(\mathbf{p})\mathbf{c}_k(\mathbf{p})\right\} \qquad (4.3.66) \\ &- \tilde{\mathbf{U}}\sum_{k=0}^{m}\left\{[\int_0^t \tau^k e^{\Lambda(\mathbf{p})(t-\tau)}d\tau]\mathbf{T}^{-1}\mathbf{B}(\mathbf{p})\mathbf{c}_k(\mathbf{p})\right\} \\ &+ \sum_{k=0}^{m}\left\{[\int_0^t \tau^k e^{\Lambda(\mathbf{p})(t-\tau)}d\tau]\mathbf{T}^{-1}\frac{\partial\mathbf{B}(\mathbf{p})}{\partial\mathbf{p}^i}\mathbf{c}_k(\mathbf{p})\right\} \\ &+ \sum_{k=0}^{m}\left\{[\int_0^t \tau^k e^{\Lambda(\mathbf{p})(t-\tau)}d\tau]\mathbf{T}^{-1}\mathbf{B}(\mathbf{p})\frac{\partial\mathbf{c}_k(\mathbf{p})}{\partial\mathbf{p}^i}\right\}.\end{aligned}$$

Again, by applying the recursive formulae (4.3.62), (4.3.63), (4.3.64a,b) and (4.3.65a,b), we may compute state sensitivity functions quite efficiently.

4.3.5 State Response and Its Sensitivities via Schur Transformation

In this section, the computation of $\mathbf{x}(t,\mathbf{p})$ and $\frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ is discussed when the matrix $\mathbf{A}(\mathbf{p})$ can not be diagonalized by a reliable transformation $\mathbf{T}$. We first consider the computation of states $\mathbf{x}(t,\mathbf{p})$. It has been shown that the exponential of the matrix $\mathbf{A}(\mathbf{p})$ can be computed by using the Schur form method when $cond(\mathbf{T}) \lesssim 100$. We now consider the evaluation of the integral term in (4.3.1) when $\mathbf{A}(\mathbf{p})$ is not diagonalizable. A common approach to evaluate this term is by applying numerical integration, even though it is usually inefficient and inaccurate.

Fortunately, the inputs of the interconnected system Σ defined in the equation (2.5.2), $\mathbf{r}(t,\mathbf{p}) = \sum_{k=0}^{m} \mathbf{c}_k(\mathbf{p})t^k$, can be generated by the following state equation with a proper initial state assigned:

$$\dot{\mathbf{z}}(t,\mathbf{p}) = \mathbf{A}_z\mathbf{z}(t,\mathbf{p}), \tag{4.3.67a}$$

$$\mathbf{r}(t,\mathbf{p}) = \mathbf{z}_0(t,\mathbf{p}), \tag{4.3.67b}$$

where $\mathbf{z} \triangleq [\mathbf{z}_0^T, \mathbf{z}_1^T, \cdots, \mathbf{z}_m^T]^T$ and $\mathbf{z}_k(\cdot,\cdot)$ is a vector function with dimension $n_{i,\theta}$ for all $k \in \{0\} \cup \underline{m}$. The initial condition is $\mathbf{z}_k(0,\mathbf{p}) = \mathbf{c}_k(\mathbf{p}) \cdot k!$. Let $n_z \triangleq (m+1) \cdot n_{i,\theta}$. Then, $\mathbf{A}_z$ is a $n_z \times n_z$ constant matrix defined by

$$\mathbf{A}_z \triangleq \begin{bmatrix} 0 & I & 0 & \cdot & \cdot & 0 \\ 0 & 0 & I & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & \cdot & I & 0 \\ 0 & 0 & \cdot & \cdot & \cdot & I \\ 0 & 0 & 0 & \cdot & 0 & 0 \end{bmatrix}. \tag{4.3.68}$$

where I is an identity matrix with dimension $n_{i,\theta}$. Combining (2.5.6a) and (4.3.67), we obtain the following state equation:

$$\dot{\bar{\mathbf{x}}}(t,\mathbf{p}) = \bar{\mathbf{A}}(\mathbf{p})\bar{\mathbf{x}}(t,\mathbf{p}), \tag{4.3.69}$$

where $\bar{\mathbf{x}} \triangleq [\mathbf{x}^T, \mathbf{z}^T]^T$, and

$$\bar{\mathbf{A}}(\mathbf{p}) = \left[\begin{array}{c|cc} \mathbf{A}(\mathbf{p}) & \mathbf{B}(\mathbf{p}) & 0 \\ \hline 0 & & \mathbf{A}_z \end{array}\right]. \tag{4.3.70}$$

The solution of (4.3.69) has the form of

$$\bar{\mathbf{x}}(t,\mathbf{p}) = e^{\bar{\mathbf{A}}(\mathbf{p})t}\bar{\mathbf{x}}(0,\mathbf{p}). \tag{4.3.71}$$

Now, we may apply the Algorithm 4.3.3 to reduce $\bar{\mathbf{A}}(\mathbf{p})$ to the Schur form $\bar{\mathbf{S}}(\mathbf{p})$ and compute $e^{\bar{\mathbf{S}}(\mathbf{p})t}$ by using the Schur form method. One remarkable property of the matrix $\bar{\mathbf{A}}(\mathbf{p})$ in (4.3.70) is that $\mathbf{A}_z$ is already a Schur form. We only need to reduce the matrix $\mathbf{A}(\mathbf{p})$ to the Schur form $\mathbf{S}(\mathbf{p})$. However, $\mathbf{S}(\mathbf{p})$ has been computed in the process of diagonalization, see Algorithm 4.3.3. Let $\mathbf{Q} \in \mathbb{C}^{n\times n}$ be the unitary matrix such that

$$\mathbf{S}(\mathbf{p}) = \mathbf{Q}^*(\mathbf{p})\mathbf{A}(\mathbf{p})\mathbf{Q}(\mathbf{p}), \tag{4.3.72}$$

Then, the Schur form $\bar{\mathbf{S}}(\mathbf{p})$ of the matrix $\bar{\mathbf{A}}(\mathbf{p})$ can be obtained directly by the following equation:

$$\bar{\mathbf{S}}(\mathbf{p}) = \left[\begin{array}{c|c} \mathbf{S}(\mathbf{p}) & \mathbf{Q}^*(\mathbf{p})\mathbf{B}(\mathbf{p}) \quad 0 \\ \hline 0 & \mathbf{A}_z \end{array}\right], \tag{4.3.73a}$$

$$= \begin{bmatrix} \mathbf{Q}^*(\mathbf{p}) & 0 \\ 0 & I \end{bmatrix} \left[\begin{array}{c|c} \mathbf{A}(\mathbf{p}) & \mathbf{B}(\mathbf{p}) \quad 0 \\ \hline 0 & \mathbf{A}_z \end{array}\right] \begin{bmatrix} \mathbf{Q}(\mathbf{p}) & 0 \\ 0 & I \end{bmatrix}. \tag{4.3.73b}$$

The solution of equation (4.3.69) now has the form of

$$\bar{\mathbf{x}}(t,\mathbf{p}) = \bar{\mathbf{Q}}(\mathbf{p})e^{\bar{\mathbf{S}}(\mathbf{p})t}\bar{\mathbf{Q}}^*(\mathbf{p})\bar{\mathbf{x}}(0,\mathbf{p}), \tag{4.3.74}$$

where $\bar{Q}(\mathbf{p}) \triangleq \begin{bmatrix} \mathbf{Q}(\mathbf{p}) & 0 \\ 0 & I \end{bmatrix}$

Next, we consider the computation of $\dfrac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ by solving the equation (2.5.9a) when the matrix $\mathbf{A}(\mathbf{p})$ is not diagonalizable. The equations $\mathbf{x}(t,\mathbf{p})$ and $\dfrac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i}$ in (2.5.9a) can be rearranged as follows:

$$\frac{d}{dt}\begin{bmatrix} \dfrac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i} \\ \mathbf{x}(t,\mathbf{p}) \end{bmatrix} = \begin{bmatrix} \mathbf{A}(\mathbf{p}) & \dfrac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i} \\ 0 & \mathbf{A}(\mathbf{p}) \end{bmatrix} \begin{bmatrix} \dfrac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i} \\ \mathbf{x}(t,\mathbf{p}) \end{bmatrix} + \begin{bmatrix} \mathbf{B}(\mathbf{p}) & \dfrac{\partial \mathbf{B}(\mathbf{p})}{\partial \mathbf{p}^i} \\ 0 & \mathbf{B}(\mathbf{p}) \end{bmatrix} \begin{bmatrix} \dfrac{\partial \mathbf{r}(t,\mathbf{p})}{\partial \mathbf{p}^i} \\ \mathbf{r}(t,\mathbf{p}) \end{bmatrix}. \tag{4.3.75}$$

The inputs in (4.3.74) can be generated by the following state equation with a proper initial states assigned:

$$\frac{d}{dt}\hat{\mathbf{z}}(t,\mathbf{p}) = \hat{\mathbf{A}}_z\hat{\mathbf{z}}(t,\mathbf{p}), \tag{4.3.76a}$$

$$\begin{bmatrix} \dfrac{\partial \mathbf{r}(t,\mathbf{p})}{\partial \mathbf{p}^i} \\ \mathbf{r}(t,\mathbf{p}) \end{bmatrix} = \hat{\mathbf{z}}_0(t,\mathbf{p}), \tag{4.3.76b}$$

where $\hat{\mathbf{z}} \triangleq [\hat{\mathbf{z}}_0^T, \hat{\mathbf{z}}_1^T, \cdots, \hat{\mathbf{z}}_m^T]^T$ and $\hat{\mathbf{z}}_k(\cdot,\cdot)$ is a vector function with dimension $2n_{i,s}$ for all $k \in \{0\} \cup \underline{m}$. The initial condition is assigned as follows:

$$\hat{\mathbf{z}}_k(0,\mathbf{p}) = \begin{bmatrix} \dfrac{\partial \mathbf{c}_k(\mathbf{p})}{\partial \mathbf{p}^i} \cdot k! \\ \mathbf{c}_k(\mathbf{p}) \cdot k! \end{bmatrix}. \tag{4.3.77}$$

Let $\hat{n}_z \triangleq (m+1)\cdot 2n_{i,s}$. Then, $\hat{\mathbf{A}}_z$ is a $\hat{n}_z \times \hat{n}_z$ constant matrix defined by

$$\hat{\mathbf{A}}_z \triangleq \begin{bmatrix} 0 & I & 0 & \cdots & 0 \\ 0 & 0 & I & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & I & 0 \\ 0 & 0 & \cdots & \cdots & I \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{4.3.78}$$

where I is an identity matrix with dimension $2n_{i,s}$. Combining (4.3.75) and (4.3.77), we obtain the following state equation:

$$\frac{d}{dt}\hat{\mathbf{x}}(t,\mathbf{p}) = \hat{\mathbf{A}}(\mathbf{p})\hat{\mathbf{x}}(t,\mathbf{p}), \tag{4.3.79}$$

where $\hat{\mathbf{x}} \triangleq \left[\dfrac{\partial \mathbf{x}^T}{\partial \mathbf{p}^i}, \mathbf{x}^T, \hat{\mathbf{z}}^T\right]^T$, and

$$\hat{\mathbf{A}}(\mathbf{p}) = \left[\begin{array}{cc|cc|c} \mathbf{A}(\mathbf{p}) & \dfrac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{B}(\mathbf{p}) & \dfrac{\partial \mathbf{B}(\mathbf{p})}{\partial \mathbf{p}^i} & 0 \\ 0 & \mathbf{A}(\mathbf{p}) & 0 & \mathbf{B}(\mathbf{p}) & \\ \hline \multicolumn{2}{c|}{0} & \multicolumn{2}{c|}{\hat{\mathbf{A}}_z} & \end{array}\right]. \tag{4.3.80}$$

The solution of the equation (4.3.79) has the form of

$$\hat{\mathbf{x}}(t,\mathbf{p}) = e^{\hat{\mathbf{A}}(\mathbf{p})t}\hat{\mathbf{x}}(0,\mathbf{p}). \tag{4.3.81}$$

Since $\hat{\mathbf{A}}_z$ is already a Schur form and we can compute the Schur form $\hat{\mathbf{S}}(\mathbf{p})$ of the matrix $\hat{\mathbf{A}}(\mathbf{p})$ directly from the following equations:

$$\hat{\mathbf{S}}(\mathbf{p}) = \hat{\mathbf{Q}}^{*}(\mathbf{p})\hat{\mathbf{A}}(\mathbf{p})\hat{\mathbf{Q}}(\mathbf{p}), \tag{4.3.82a}$$

$$= \left[\begin{array}{cc|cc|c} \mathbf{S}(\mathbf{p}) & \mathbf{Q}^{*}(\mathbf{p})\frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^{i}}\mathbf{Q}(\mathbf{p}) & \mathbf{Q}^{*}(\mathbf{p})\mathbf{B}(\mathbf{p}) & \mathbf{Q}^{*}(\mathbf{p})\frac{\partial \mathbf{B}(\mathbf{p})}{\partial \mathbf{p}^{i}} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}(\mathbf{p}) & \mathbf{0} & \mathbf{Q}^{*}(\mathbf{p})\mathbf{B}(\mathbf{p}) & \\ \hline \multicolumn{2}{c|}{\mathbf{0}} & \multicolumn{2}{c|}{\hat{\mathbf{A}}_z} & \end{array}\right], \tag{4.3.82b}$$

where

$$\hat{\mathbf{Q}}(\mathbf{p}) = \begin{bmatrix} \mathbf{Q}(\mathbf{p}) & 0 & 0 \\ 0 & \mathbf{Q}(\mathbf{p}) & 0 \\ 0 & 0 & I \end{bmatrix}. \tag{4.3.83}$$

Therefore, the solution of the equation (4.3.79) has the form of

$$\hat{\mathbf{x}}(t,\mathbf{p}) = \hat{\mathbf{Q}}(\mathbf{p})e^{\hat{\mathbf{S}}(\mathbf{p})t}\hat{\mathbf{Q}}^{*}(\mathbf{p})\hat{\mathbf{x}}(0,\mathbf{p}), \tag{4.3.84}$$

which can be evaluated by the Schur form method.

4.4 Concluding Remarks

In this chapter, we have derived all the formulae and algorithms necessary for the simulation of both time and frequency responses as well as their sensitivity functions. All the methods and algorithms are based on matrix decompositions. If the system matrix $\mathbf{A}(\mathbf{p})$ or $\mathbf{A}_c(\mathbf{p}_c)$ can be diagonalized by a reliable transformation $\mathbf{T}(\mathbf{p})$, then the evaluations of time or frequency domain response will be efficient. However, the diagonalization method may cause numerical ill-conditioning, when the condition number of the matrix $\mathbf{T}(\mathbf{p})$, $cond(\mathbf{T}(\mathbf{p}))$ is large. This can be avoided by setting up a threshold number $\kappa > 0$ for the condition number $\mathbf{T}(\mathbf{p})$. When the diagonalization method fails, i.e., $cond(\mathbf{T}(\mathbf{p})) > \kappa$, one can always use the Hessenberg form in frequency domain and Schur form in time domain to evaluate the necessary responses and their sensitivity functions.

CHAPTER 5

OPTIMIZATION ALGORITHMS

5.1. Introduction

In engineering design, many problems, such as the design of circuits, control systems and structures[Pol.8, Aus.1, Bal.1] can be formulated as a nonlinear programming problems of the form

$$\min\{\, f(\mathbf{x}) \mid g^j(\mathbf{x}) \leq 0\,,\ j \in \underline{m}\,\}, \tag{5.1.1}$$

where $f, g^j : \mathbb{R}^n \to \mathbb{R}$, $j \in \underline{m} \triangleq \{1, 2, \cdots, m\}$, are locally Lipschitz continuous, which are frequently of the form of a max function, such as $f(\mathbf{x}) = \max_{\mathbf{y} \in Y} \phi(\mathbf{x}, \mathbf{y})$ with Y a compact set. Problem (5.1.1) can be treated in the equivalent form

$$\min\{\, f(\mathbf{x}) \mid \psi(\mathbf{x}) \leq 0\,\}, \tag{5.1.2}$$

where $\psi(\mathbf{x}) \triangleq \max_{j \in \underline{m}} g^j(\mathbf{x})$. In conventional Phase I - Phase II *Methods of Feasible Directions* [Pol.5, Zou.1], the problem of computing an initial feasible point for (5.1.2) is related to the problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \psi(\mathbf{x}). \tag{5.1.3}$$

The Phase I - Phase II methods reduce ψ when the current estimate of a solution $\mathbf{x}_i$ is not feasible, and reduce the cost f, while maintaining feasibility, when $\mathbf{x}_i$ is feasible. Since the formulation $\psi(\mathbf{x}) \leq 0$ gives no way of estimating the violation for each constraint, $g^j(\mathbf{x}) \leq 0$, separately, the equation $\psi(\mathbf{x}_{i+1}) < \psi(\mathbf{x}_i)$ can not guarantee a reduction of each constraint. Therefore, instead of the formulation (5.1.2) we need a new formulation which can take into account several important characteristics of a large class of design problems.

In design, specifications are often formulated as a sequence of constrained minimax problems. For example, suppose a design problem has two constraints: $g^1(\mathbf{x}) \leq 0$, $g^2(\mathbf{x}) \leq 0$, and suppose that function $g^2(\cdot)$ is defined only on the set $\{\mathbf{x} \in \mathbb{R}^n \mid g^1(\mathbf{x}) \leq 0\}$. In solving (5.1.1), the constraint $g^1(\mathbf{x}) \leq 0$ should be satisfied

first. Then, maintaining feasibility of the first constraint, the constraint $g^2(\mathbf{x}) \leq 0$ is solved. Finally, the cost is reduced in the feasible region.

In addition, in real world problems the importance of design specifications is not always uniformly distributed among the constraints. Some of the constraints may have higher priority than others. For example, besides stability requirements, avoiding obstacles in the robot path planning problems has much higher priority than any other design specifications. The formulation $\psi(\mathbf{x}) \leq 0$ fails to estimate the priority of some constraints.

Finally, given a set of constraints in (5.1.1), it is sometimes hard to determine whether the feasible set

$$F \triangleq \{\mathbf{x} \in \mathbb{R}^n \mid g^j(\mathbf{x}) \leq 0\,,\ j \in \underline{m}\}, \tag{5.1.4}$$

is empty or not. However we can see that an inclusion property holds for feasible sets:

$$F_1 \supset F_2 \supset \cdots \supset F_m = F\,, \tag{5.1.5}$$

where $F_j \triangleq \{\mathbf{x} \in \mathbb{R}^n \mid g^i(\mathbf{x}) \leq 0\,,\ i \in \underline{j}\}$. If the problem (5.1.1) is solved sequentially by first solving the problem

$$\min\{\, f(\mathbf{x}) \mid \mathbf{x} \in F_1\,\}, \tag{5.1.ba}$$

then after $\mathbf{x} \in F_1$ is satisfied, we switch to the next problem

$$\min\{\, f(\mathbf{x}) \mid \mathbf{x} \in F_2\,\}, \tag{5.1.ba}$$

and so on. If the feasible set F is empty, then the suggested method will go as far as possible to reach the "boundary" of feasibility for the constraints set.

Obviously, there is no unique way to get around all of the difficulties mentioned above. In [Pol.9, Nye.1], two Phase I-II-III Methods of Feasible Directions are proposed, which solve some of the difficulties. In this chapter, a Multi-Phase Method of Feasible Directions is described, which overcomes the difficulties outlined above.

5.2 Definitions and Theorems for Nonsmooth Analysis

To simplify the discussion in the following sections, some notations, definitions and theorems are described in this section. All the related analysis of nonsmooth functions can be found in [Cla.1, Pol.3, Pol.4].

Definition 5.2.1 : A function $f:\mathbb{R}^n \to \mathbb{R}$ is said to be *upper semi-continuous* at $\hat{\mathbf{x}}$ (u.s.c.) if for every $\delta>0$ there exists a $\hat{\rho} > 0$ such that

$$f(\mathbf{x})-f(\hat{\mathbf{x}}) \leq \delta, \qquad \forall\ \mathbf{x} \in B(\hat{\mathbf{x}},\hat{\rho}), \tag{5.2.1}$$

where $B(\hat{\mathbf{x}},\hat{\rho}) \triangleq \{\mathbf{x}\in\mathbb{R}^n \mid \|\mathbf{x}-\hat{\mathbf{x}}\| \leq \hat{\rho}\}$. $f(\cdot)$ is said to be u.s.c. if it is u.s.c. at all $\mathbf{x} \in \mathbb{R}^n$.

∎

Definition 5.2.2 : A function $f:\mathbb{R}^n \to \mathbb{R}$ is said to be *lower semi-continuous* if $-f(\cdot)$ *is u.s.c.*

∎

Next we turn to point-to-set functions. The most important concept for point-to-set maps in that of upper semi-continuity, though some use can also be made of lower semi-continuity. The definitions, below, of u.s.c. and l.s.c. of point-to-set functions have nothing to do with the ones that we gave for functions from $\mathbb{R}^n$ into $\mathbb{R}$.

Definition 5.2.3 : A function (map) $f:\mathbb{R}^n \to 2^{\mathbb{R}^m}$ is said to be *upper-semi-continuous* (u.s.c.) at $\hat{\mathbf{x}}$ if

(a) $f(\hat{\mathbf{x}})$ is compact and

(b) for every open set G such that $f(\hat{\mathbf{x}}) \subset G$, there exists a $\hat{\rho}>0$ such that $f(\mathbf{x}) \subset G$, for all $\mathbf{x} \in B(\hat{\mathbf{x}},\hat{\rho})$.

∎

Definition 5.2.4 : A function $f:\mathbb{R}^n \to 2^{\mathbb{R}^m}$ is said to be *lower-semi-continuous* (l.s.c.) at $\hat{\mathbf{x}}$ if for every open set G such that $f(\hat{\mathbf{x}}) \cap G \neq \emptyset$ there exists a $\hat{\rho}>0$ such that $f(\mathbf{x}) \cap G \neq \emptyset$ for all $\mathbf{x} \in B(\hat{x},\hat{\rho})$.

∎

A function $f:\mathbb{R}^n \to 2^{\mathbb{R}^m}$ is u.s.c. (l.s.c.) if it is u.s.c. (l.s.c.) at every $\mathbf{x} \in \mathbb{R}^n$.

Definition 5.2.5 : A function $f:\mathbb{R}^n \to 2^{\mathbb{R}^m}$ is said to be continuous if it is both u.s.c. and l.s.c.

∎

Definition 5.2.6 : We say that $f:\mathbb{R}^n \to \mathbb{R}^m$ is *locally Lipschitz continuous* (l.L.c.) at $\hat{\mathbf{x}}$ *if there exist* $L \in [0,\infty)$, $\hat{\rho} > 0$ *such that*

$$\|f(\mathbf{x})-f(\mathbf{x}')\| \leq L\,\|\mathbf{x}-\mathbf{x}'\|, \qquad \forall\, \mathbf{x},\mathbf{x}' \in B(\hat{\mathbf{x}},\hat{\rho}). \tag{5.2.2}$$

∎

Definition 5.2.7 : Let $f:\mathbb{R}^n \to \mathbb{R}$ be l.L.c. We define the (Clarke) *generalized directional derivative* of $f(\cdot)$ at $\mathbf{x} \in \mathbb{R}^n$ in the direction $\mathbf{h} \in \mathbb{R}^n$ by

$$d_0 f(\mathbf{x};\mathbf{h}) \triangleq \overline{\lim_{\substack{t \downarrow 0 \\ \mathbf{y} \to \mathbf{x}}}} \frac{f(\mathbf{y}+t\mathbf{h})-f(\mathbf{y})}{t}. \tag{5.2.3}$$

∎

Definition 5.2.8 : Let $f:\mathbb{R}^n \to \mathbb{R}$ be l.L.c. We define the (Clarke) *generalized gradient* of $f(\cdot)$ at $\mathbf{x}$ by

$$\partial f(\mathbf{x}) \triangleq \{\, \xi \in \mathbb{R}^n \mid d_0 f(\mathbf{x};\mathbf{h}) \geq \langle \xi, \mathbf{h} \rangle,\ \forall\, \mathbf{h} \in \mathbb{R}^n \,\}. \tag{5.2.4}$$

∎

Definition 5.2.9 : A l.L.c. function $f:\mathbb{R}^n \to \mathbb{R}$ is said to be *regular* if its direc-

∎

tional derivative $df(\mathbf{x};\mathbf{h})$ exists for all $\mathbf{x},\mathbf{h} \in \mathbb{R}^n$ and $df(\mathbf{x};\mathbf{h}) = d_0 f(\mathbf{x};\mathbf{h})$. .br

Definition 5.2.10 : We shall say that $\{G_\epsilon\psi(\cdot)\}_{\epsilon\geq 0}$, where, $G_\epsilon\psi:\mathbb{R}^n \to 2^{\mathbb{R}^n}$, is a family of *convergent direction finding* (c.d.f.) *maps* for the l.L.c. function $\psi:\mathbb{R}^n \to \mathbb{R}$ if

(a) For all $\mathbf{x} \in \mathbb{R}^n$, $\partial\psi(\mathbf{x}) = G_0\psi(\mathbf{x})$.

(b) For all $\mathbf{x} \in \mathbb{R}^n$, if $0 \leq \epsilon < \epsilon'$, then $G_\epsilon\psi(\mathbf{x}) \subset G_{\epsilon'}\psi(\mathbf{x})$.

(c) For any $\epsilon \geq 0$ and $\mathbf{x} \in \mathbb{R}^n$, $G_\epsilon\psi(\mathbf{x})$ is convex.

(d) For any $\epsilon \geq 0$, $G_\epsilon\psi(\mathbf{x})$ is bounded on bounded sets.

(e) $G_\epsilon\psi(\mathbf{x})$ is u.s.c. in $(\epsilon,\mathbf{x})$ at $(0, \hat{\mathbf{x}})$ for all $\hat{\mathbf{x}} \in \mathbb{R}^n$.

(f) Given any $\hat{\mathbf{x}} \in \mathbb{R}^n, \hat{\epsilon} > 0$ and $\hat{\delta} > 0$ there exists a $\hat{\rho} > 0$ such that for any $\hat{\xi} \in \partial\psi(\hat{\mathbf{x}})$ and any $x \in B(\hat{\mathbf{x}}, \hat{\rho})$, there exists a $\xi \in G_{\hat{\epsilon}}\psi(\mathbf{x})$ such that $\|\xi - \hat{\xi}\| \leq \hat{\delta}$.

∎

Lemma 5.2.1 : Suppose that $G_{\hat{\epsilon}}\psi(\cdot)$, $\hat{\epsilon} > 0$, is an element of a family of c.d.f. maps, defined as above. Then for any $\hat{\mathbf{x}} \in \mathbb{R}^n$, $\delta > 0$, there exists a $\hat{\rho} > 0$ such that for any $\mathbf{x}', \mathbf{x}'' \in B(\hat{\mathbf{x}}, \hat{\rho})$ and any $\xi' \in \partial\psi(\mathbf{x}')$ there exists an $\xi'' \in G_{\hat{\epsilon}}\psi(\mathbf{x}'')$ such that $\|\xi'' - \xi'\| \leq \delta$.

∎

Lemma 5.2.2 : Suppose that $g^j: \mathbb{R}^n \to \mathbb{R}$, $j \in \underline{m}$ are l.L.c and *regular* functions. Let $\psi(\mathbf{x}) \triangleq \max_{j \in \underline{m}} g^j(\mathbf{x})$ and $I(\mathbf{x}) \triangleq \{ j \in \underline{m} \mid \psi(\mathbf{x}) = g^j(\mathbf{x}) \}$. Then,

(i) $\psi(\cdot)$ is a l.L.c and *regular* function.

(ii) $\partial\psi(\mathbf{x}) = \mathrm{co}\,\{\partial g^j(\mathbf{x})\}_{j \in I(\mathbf{x})}$.

∎

Lemma 5.2.3 : Suppose that $f^j: \mathbb{R}^n \to \mathbb{R}$, $j \in \underline{m}$ are continuous differentiable functions, and that $\psi(\mathbf{x}) \triangleq \max_{j \in \underline{m}} f^j(\mathbf{x})$. Then, the family of maps $\{G_\epsilon\psi(\cdot)\}_{\epsilon \geq 0}$ defined by

$$G_\epsilon\psi(\mathbf{x}) \triangleq \mathrm{co}\,\{\nabla f^j(\mathbf{x})\}_{j \in I_\epsilon(\mathbf{x})}, \qquad \epsilon \geq 0, \tag{5.2.5}$$

with $I_\epsilon(\mathbf{x}) \triangleq \{ j \in \underline{m} \mid \psi(\mathbf{x}) - f^j(\mathbf{x}) \leq \epsilon \}$, is a family of c.d.f. maps for $\psi(\mathbf{x})$.

∎

Lemma 5.2.4 : Suppose that $\psi(\mathbf{x}) = \max_{y \in Y} \phi(\mathbf{x},\mathbf{y})$ where $\phi: \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is continuous, $\nabla_{\mathbf{x}}\phi(\mathbf{x},\mathbf{y})$ exists and is continuous and $Y \subset \mathbb{R}^m$ is compact. Let a family of maps $\{G_\epsilon\psi(\cdot)\}_{\epsilon \geq 0}$ be defined by

$$G_\epsilon\psi(\mathbf{x}) \triangleq \mathrm{co}\,\{\nabla_{\mathbf{x}}\phi(\mathbf{x},\mathbf{y})\}_{\mathbf{y} \in \hat{Y}_\epsilon(\mathbf{x})}, \tag{5.2.6}$$

where

$$\hat{Y}_\epsilon(\mathbf{x}) \triangleq \{\mathbf{y} \in Y \mid \psi(\mathbf{x}) - \phi(\mathbf{x},\mathbf{y}) \leq \epsilon, \quad \mathbf{y} \text{ is a local maximizer of } \phi(\mathbf{x},\cdot) \text{ in } Y\}. \tag{5.2.7}$$

Assume that $\hat{Y}_\epsilon(\mathbf{x})$ is finite for all $\mathbf{x} \in \mathbb{R}^n$ and $\epsilon \geq 0$. Then, $\{G_\epsilon\psi(\cdot)\}_{\epsilon\geq 0}$ is a family of c.d.f. maps.

∎

Theorem 5.2.1 : *(Lebourg Mean Value Theorem)* Let $f:\mathbb{R}^n \to \mathbb{R}$ be l.L.c. Then, given any $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$,

$$f(\mathbf{y}) - f(\mathbf{x}) = \langle\, \xi,\, \mathbf{y}-\mathbf{x} \,\rangle, \tag{5.2.8}$$

for some $\xi \in \partial f(\mathbf{x} + s(\mathbf{y}-\mathbf{x}))$, with $s \in (0,1)$.

∎

5.3 Multi-Phase Methods of Feasible Directions

Consider the problem (5.1.1). We shall assume that the functions $f(\cdot)$, $g^j(\cdot)$ are l.L.c. and *regular*. According to the priority of each constraint, we may divide the set $\underline{m}$ into μ subsets I_i, $i \in \underline{\mu}$ such that $\underline{m} = I_1 \cup I_2 \cup \cdots \cup I_\mu$, and $I_i \cap I_j = \phi$ for all $i,j \in \underline{\mu}$ and $i \neq j$. We define

$$\psi^i(\mathbf{x}) \triangleq \max\{\, g^j(\mathbf{x}) \mid j \in I_i \,\} \tag{5.3.1}$$

By Lemma 5.2.2, $\psi^i(\cdot)$, $i \in \underline{\mu}$, are locally Lipschitz continuous and *regular* functions. Then the problem (5.1.1) can be expressed in following form

$$\min\{\, f(\mathbf{x}) \mid \psi^j(\mathbf{x}) \leq 0\,,\, j \in \underline{\mu} \,\}. \tag{5.3.2}$$

In this chapter, we intend to solve the problem (5.3.2) in $\mu+1$ successive phases. In phase 1, the constraint ψ^1 is decreased for each iteration. Phase 2 to Phase μ, say Phase j, each iteration tries to decrease the constraint ψ^j, while maintaining the constraints, $\psi^1, \psi^2, \cdots$ and ψ^{j-1}, satisfied. Finally, in phase $\mu+1$, the cost f is progressively reduced, while maintaining feasibility of all constraints.

We shall assume that we have for all the functions f, ψ^j, $j \in \underline{\mu}$, families of convergent direction finding maps $\{G_\epsilon f(\cdot)\}_{\epsilon \geq 0}$ and $\{G_\epsilon \psi^j(\cdot)\}_{\epsilon \geq 0}$ (see Definition 5.2.10). We define $\Phi^k : \mathbb{R}^n \to \mathbb{R}$ and $\Phi^k(\cdot)_+$ for all $k \in \underline{\mu}$ as follows:

$$\Phi^k(\mathbf{x}) \triangleq \max\{ \psi^1(\mathbf{x}), \psi^2(\mathbf{x}), , \cdots, \psi^k(\mathbf{x}) \} \tag{5.3.3a}$$

$$\Phi^k(\mathbf{x})_+ = \max\{ 0, \Phi^k(\mathbf{x}) \} \tag{5.3.3b}$$

Thus, for all $\mathbf{x} \in \mathbb{R}^n$ we have

$$\begin{aligned} &\Phi^i(\mathbf{x}) \leq \Phi^j(\mathbf{x}) \\ &\Phi^i(\mathbf{x})_+ \leq \Phi^j(\mathbf{x})_+ \end{aligned} \qquad \forall\, i,j \in \underline{n},\ i \leq j\,. \tag{5.3.4}$$

We define a *phase index* function $\hat{k} : \mathbb{R}^n \to \mathbb{N}$ as follows:

$$\hat{k}(\mathbf{x}) \triangleq \begin{cases} \mu\,, & \text{if } \psi^k(\mathbf{x}) \leq 0, \quad \forall\, k \in \underline{\mu}; \\ \min\{\, k \in \underline{\mu} \mid \psi^k(\mathbf{x}) > 0 \,\}, & \text{otherwise.} \end{cases} \tag{5.3.5}$$

To simplify, $\hat{k}(\mathbf{x})$ will be denoted by $\hat{k}$.

We are about to state a multi-phase algorithm. First, for any $\mathbf{x} \in \mathbb{R}^n$ and $\epsilon \geq 0$, we define two ϵ-most violated constraint index sets

$$I_\epsilon^{\hat{k}}(\mathbf{x}) \triangleq \{\, j \in \{\hat{k}+1, \cdots, \mu\} \mid \psi^j(\mathbf{x}) \geq \Phi^{\hat{k}}(\mathbf{x})_+ - \epsilon \,\} \tag{5.3.6}$$

$$J_\epsilon^{\hat{k}}(\mathbf{x}) \triangleq \{\, j \in \underline{\hat{k}} \mid \psi^j(\mathbf{x}) \geq -\epsilon \,\} \tag{5.3.7}$$

Next we define

$$\begin{bmatrix} \gamma \\ G \end{bmatrix} \triangleq \{\, \overline{\xi} \in \mathbb{R}^{n+1} \mid \overline{\xi} = [\gamma, \xi]^T,\ \xi \in G \,\} \tag{5.3.8}$$

where $\gamma \in \mathbb{R}$ and $G \subset \mathbb{R}^n$. Let γ_j, $\forall\, j \in \underline{\mu}$, be positive real numbers, then a family of $\hat{k}$th-phase of *multi-phase augmented convergent direction finding (m.a.c.d.f.)* maps $\{\overline{G}_\epsilon^{\hat{k}}(\mathbf{x})\}$ for the problem (5.3.2) is defined as follows:

$$\overline{G}_\epsilon^{\hat{k}}(\mathbf{x}) \triangleq \mathrm{co}\left\{ \begin{bmatrix} \gamma_\mu \Phi^\mu(\mathbf{x})_+ \\ G_\epsilon f(\mathbf{x}) \end{bmatrix} \bigcup \begin{bmatrix} \gamma_{j-1} \Phi^{j-1}(\mathbf{x})_+ \\ G_\epsilon \psi^j(\mathbf{x}) \end{bmatrix}_{j \in I_\epsilon^{\hat{k}}(\mathbf{x})} \bigcup \begin{bmatrix} 0 \\ G_\epsilon \psi^j(\mathbf{x}) \end{bmatrix}_{j \in J_\epsilon^{\hat{k}}(\mathbf{x})} \right\} \tag{5.3.9a}$$

Next, we define the ϵ- *search direction* by

$$\bar{\mathbf{h}}_\epsilon(\mathbf{x}) \triangleq (\mathbf{h}_\epsilon^0(\mathbf{x}), \mathbf{h}_\epsilon(\mathbf{x})) \triangleq \operatorname{argmin}\{\, 1/2\,\|\bar{h}\|^2 \mid \bar{h} \in \bar{G}_\epsilon^{\hat{k}}(\mathbf{x}) \,\} \tag{5.3.9b}$$

and the ϵ *adjustment law* by

$$\epsilon(\mathbf{x}) \triangleq \max\{\, \epsilon \in E \mid \|\bar{\mathbf{h}}_\epsilon(\mathbf{x})\|^2 \geq \epsilon \,\} \tag{5.3.9c}$$

where $E \triangleq \{0,1,\nu,\nu^2,\nu^3, \cdots\}$ and $\nu \in (0,1)$.

Algorithm 5.3.1 : (Requires $\{G_\epsilon f(\cdot)\}_{\epsilon\geq 0}$, $\{G_\epsilon \psi^j(\cdot)\}_{\epsilon\geq 0}$, $j \in \underline{\mu}$ families of c.d.f. maps for $f(\cdot)$ and $\psi^j(\cdot)$; $\nu \in (0,1)$ for the set E in (5.3.9c)).

Data: $\mathbf{x}_0 \in \mathbb{R}^n$.

Step 0: Set i=0.

Step 1: Compute $\hat{k} = \hat{k}(\mathbf{x}_i)$ according to (5.3.5).

Step 2: Compute $\epsilon(\mathbf{x}_i)$ according to (5.3.9c) and the *search direction*

$$\mathbf{h}_i = -\mathbf{h}_{\epsilon(\mathbf{x}_i)}(\mathbf{x}_i)\,, \tag{5.3.10a}$$

according to (5.3.9b).

Step 3: Compute the *step size* as follows:

If $\Phi^\mu(\mathbf{x}_i) > 0$, then

$$\lambda_i \in \lambda_\Phi(\mathbf{x}_i) \triangleq \operatorname*{argmin}_{\lambda\geq 0}\{\, \psi^{\hat{k}}(\mathbf{x}_i + \lambda\mathbf{h}_i) \mid \psi^j(\mathbf{x}_i + \lambda\mathbf{h}_i) \leq 0, \forall\ j \in \underline{\hat{k}-1} \,\} \tag{5.3.10b}$$

If $\Phi^\mu(\mathbf{x}_i) \leq 0$, then

$$\lambda_i \in \lambda_{f,\Phi}(\mathbf{x}_i) \triangleq \operatorname*{argmin}_{\lambda\geq 0}\{\, f(\mathbf{x}_i + \lambda\mathbf{h}_i) \mid \Phi^\mu(\mathbf{x}_i + \lambda\mathbf{h}_i) \leq 0 \,\}. \tag{5.3.10c}$$

Step 4: Update:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda_i\mathbf{h}_i\,, \tag{5.3.10d}$$

replace i by $i+1$ and go to step 1.

∎

Note the effect of $\Phi^{\hat{k}}(\mathbf{x})$ on $\bar{\mathbf{h}}_\epsilon(\mathbf{x})$ in equation (5.3.9). When $\Phi^{\hat{k}}(\mathbf{x})_+$ is large, then $\mathbf{h}_\epsilon(\mathbf{x}) \cong \operatorname{argmin}\{\, 1/2\,\|\mathbf{h}\|^2 \mid \mathbf{h} \in \operatorname{co}\{G_\epsilon\psi^j(\mathbf{x})\}_{j\in J_\epsilon^{\hat{k}}(\mathbf{x})}\}$. The influence of suppressing the cost and some other violated constraints in this manner is that the algorithm

concentrates on decreasing $\psi^{\hat{k}}(\mathbf{x})$ at an infeasible point x without totally ignoring the cost and the other violated constraints.

Remark : Since it improves computational efficiency, the following multi-phase Armijo step size rule should be substituted for (5.3.10b,c) when implementing the above algorithm model:

If $\Phi^{\mu}(\mathbf{x}_i) > 0$, then

$$\lambda_i \triangleq \max\{\lambda \mid \lambda=\beta^l,\ l \in \mathbb{N},\ \psi^{\hat{k}}(\mathbf{x}_i+\lambda\mathbf{h}_i) - \psi^{\hat{k}}(\mathbf{x}_i) \leq -\lambda\alpha\epsilon(\mathbf{x}_i)\ ; \quad (5.3.11a)$$

$$\psi^j(\mathbf{x}_i+\lambda\mathbf{h}_i) \leq 0,\ \forall\ j \in \underline{\hat{k}-1}\}$$

If $\Phi^{\mu}(\mathbf{x}_i) \leq 0$, then

$$\lambda_i \triangleq \max\{\lambda \mid \lambda=\beta^l,\ l \in \mathbb{N},\ f(\mathbf{x}_i+\lambda\mathbf{h}_i)-f(\mathbf{x}_i) \leq -\lambda\alpha\epsilon(\mathbf{x}_i)\ ; \quad (5.3.11b)$$

$$\Phi^{\mu}(\mathbf{x}_i+\lambda\mathbf{h}_i) \leq 0\}$$

where $\alpha, \beta \in (0,1)$.

∎

Assumption 5.3.1 : For every $\mathbf{x} \in \mathbb{R}^n$ such that $\Phi^{\hat{k}}(\mathbf{x}) \geq 0$,

$$0 \notin \begin{cases} \mathrm{co}\{\partial\Phi^{\hat{k}}(\mathbf{x}) \bigcup \partial\Phi^{\hat{k}-1}(\mathbf{x})\} & \text{if } \hat{k} > 1 \\ \partial\Phi^{\hat{k}}(\mathbf{x}) & \text{if } \hat{k} = 1 \end{cases}, \quad (5.3.12)$$

where $\hat{k}$ is the solution of equation (5.3.5).

$0 \notin \partial\Phi^{\hat{k}}(\mathbf{x})$ for every $\mathbf{x} \in \mathbb{R}^n$ such that $\Phi^{\hat{k}}(\mathbf{x}) \geq 0$, where $\hat{k}$ is the solution of equation (5.3.5).

∎

The following result can be established.

Theorem 5.3.1 : Suppose that Assumption 5.3.1 holds and that $\{\mathbf{x}_i\}_{i=0}^{\infty}$ is a sequence constructed by Algorithm 5.3.1 in solving (5.3.2) with $f(\cdot)$, $\psi^j(\cdot)$ l.L.c. and *regular*. If $\mathbf{x}_i \xrightarrow{K} \hat{\mathbf{x}}$ as $i \to \infty$, then $\Phi^{\mu}(\hat{\mathbf{x}}) \leq 0$ and

$$0 \in \operatorname{co}\left\{ \partial f(\hat{\mathbf{x}}) \bigcup \{\partial \psi^j(\hat{\mathbf{x}})\}_{j \in J_\delta^\epsilon(\hat{\mathbf{x}})} \right\}. \tag{5.3.13}$$

Proof : Suppose that $\Phi^\mu(\hat{\mathbf{x}}) > 0$ or $0 \notin \operatorname{co}\{ \partial f(\hat{\mathbf{x}}) \cup \{\partial \psi^j(\hat{\mathbf{x}})\}_{j \in J_\delta^\epsilon(\hat{\mathbf{x}})}\}$ for the sake of obtaining a contradiction. We consider two cases:

Case 1 : There exists an $\hat{\imath} \in \mathbb{N}_+$ such that $\psi^{\hat{k}}(\mathbf{x}_i) > 0$ and $\hat{k}$ is a constant solution of equation (5.3.5) for all $i \geq \hat{\imath}$. Then, by (5.3.10) $\{\psi^{\hat{k}}(\mathbf{x}_i)\}_{i=0}^{\infty}$ is monotone decreasing, and $\psi^{\hat{k}}(\mathbf{x}_i) \xrightarrow{K} \psi^{\hat{k}}(\hat{\mathbf{x}})$ by continuity of $\psi^{\hat{k}}(\cdot)$. Hence $\psi^{\hat{k}}(\mathbf{x}_i) \to \psi^{\hat{k}}(\hat{\mathbf{x}})$ as $i \to \infty$ and $\psi^{\hat{k}}(\hat{\mathbf{x}}) \geq 0$.

(i) If $\psi^{\hat{k}}(\hat{\mathbf{x}}) > 0$, by Assumption 5.3.1 and equation (5.3.9a) we have $0 \notin \overline{G}_0^{\hat{k}}(\hat{\mathbf{x}})$.

(ii) If $\psi^{\hat{k}}(\hat{\mathbf{x}}) = 0$ and $\Phi^\mu(\hat{\mathbf{x}}) > 0$, again by Assumption 5.3.1 and equation (5.3.9a) we have $0 \notin \overline{G}_0^{\hat{k}}(\hat{\mathbf{x}})$.

(iii) If $\psi^{\hat{k}}(\hat{\mathbf{x}}) = 0$ and $\Phi^\mu(\hat{\mathbf{x}}) = 0$, then by assumption we have

$$0 \notin \operatorname{co}\{ \partial f(\hat{\mathbf{x}}) \cup \{\partial \psi^j(\hat{\mathbf{x}})\}_{j \in J_\delta^\epsilon(\hat{\mathbf{x}})}\} = \overline{G}_0^{\hat{k}}(\hat{\mathbf{x}})$$

Thus, by (i), (ii) and (iii) and $\overline{G}_0^{\hat{k}}(\cdot)$ is u.s.c. in $(\epsilon,\mathbf{x})$ at $(0,\hat{\mathbf{x}})$, it follows that there exist $i_0 \in \mathbb{N}_+$ and $\hat{\epsilon} > 0$, such that $\epsilon(\mathbf{x}_i) \geq \hat{\epsilon} > 0$ for all $i \in K$, $i \geq i_0$. Since $\overline{G}_0^{\hat{k}}(\mathbf{x})$ is bounded on bounded sets, there exists a $b \in (0,\infty)$ such that

$$0 < \hat{\epsilon} \leq \epsilon(\mathbf{x}_i) \leq \|\overline{\mathbf{h}}_\epsilon(\mathbf{x}_i)\|^2 \leq b^2$$

for all $i \in K$, $i \geq i_0$.

(a) Suppose that $\hat{k}=1$. By Lemma 5.2.1, given $\hat{\delta} = \hat{\epsilon}/2$, there exists a $\hat{\rho} > 0$ such that for any $x', x'' \in B(\hat{\mathbf{x}}, \hat{\rho})$, and any $\xi' \in \partial\psi^{\hat{k}}(\mathbf{x}')$, there exists a $\xi'' \in G_{\hat{\epsilon}}(\mathbf{x}'')$ such that $b \cdot \|\xi' - \xi''\| \leq \hat{\delta} = \hat{\epsilon}/2$. Hence there exists an $i_1 \geq i_0$ and a $\hat{\lambda} > 0$ such that for all $i \geq i_1$, $i \in K$, $\mathbf{x}_i \in B(\hat{\mathbf{x}}, \hat{\rho})$, $(\mathbf{x}_i + s\hat{\lambda}\mathbf{h}_i) \in B(\hat{\mathbf{x}}, \hat{\rho})$ for all $s \in (0,1)$. For any $\xi_{i\hat{\lambda}} \in \partial\psi^{\hat{k}}(\mathbf{x}_i + s\hat{\lambda}\mathbf{h}_i)$, there exists a $\xi_{i\hat{\lambda}}' \in G_{\hat{\epsilon}}\psi^{\hat{k}}(\mathbf{x}_i) \subset G_{\epsilon(\mathbf{x}_i)}(\mathbf{x}_i)$ such that

$\|\xi_{i\hat{\lambda}} - \xi_{i\hat{\lambda}}'\| \cdot b \leq \hat{\epsilon}/2$. By the Lebourg Mean Value Theorem 5.2.1, we obtain, for all $i \geq i_1$, $i \in K$, that

$$\psi^{\hat{k}}(\mathbf{x}_i + \lambda_i \mathbf{h}_i) - \psi^{\hat{k}}(\mathbf{x}_i) \leq \psi^{\hat{k}}(\mathbf{x}_i + \hat{\lambda}\mathbf{h}_i) - \psi^{\hat{k}}(\mathbf{x}_i) = \hat{\lambda} < \mathbf{h}_i, \xi_{i\hat{\lambda}} > \tag{5.3.14}$$

$$= \hat{\lambda}[< \mathbf{h}_i, \xi_{i\hat{\lambda}}' > + < \mathbf{h}_i, \xi_{i\hat{\lambda}} - \xi_{i\hat{\lambda}}' >]$$

$$\leq \hat{\lambda}[-\|\mathbf{h}_i\|^2 + \|\mathbf{h}_i\| \|\xi_{i\hat{\lambda}} - \xi_{i\hat{\lambda}}'\|]$$

$$\leq \hat{\lambda}[-\hat{\epsilon} + b \cdot \|\xi_{i\hat{\lambda}} - \xi_{i\hat{\lambda}}'\|]$$

$$\leq -\hat{\lambda}\hat{\epsilon}/2 < 0$$

where $\xi_{i\hat{\lambda}} \in \partial\psi^{\hat{k}}(\mathbf{x}_i + s_i\hat{\lambda}\mathbf{h}_i)$ for some $s_i \in (0,1)$ and $\xi_{i\hat{\lambda}}' \in G_{\epsilon(\mathbf{x}_i)}\psi^{\hat{k}}(\mathbf{x}_i)$. Now, $\{\psi^{\hat{k}}(\mathbf{x}_i)\}_{i=0}^{\infty}$ is monotone decreasing and $\psi^{\hat{k}}(\mathbf{x}_i) \overset{K}{\to} \psi^{\hat{k}}(\hat{\mathbf{x}})$, since $\psi^{\hat{k}}(\cdot)$ is continuous, hence $\psi^{\hat{k}}(\mathbf{x}_i) \to \psi^{\hat{k}}(\hat{x})$. But this contradicts (5.3.14). Hence we must have had $\Phi^{\mu}(\hat{\mathbf{x}}) \leq 0$ and $0 \in \text{co}\{\partial f(\hat{\mathbf{x}}) \cup \{\partial\psi^j(\hat{\mathbf{x}})\}_{j \in J_0^{\mu}(\hat{\mathbf{x}})}\}$.

(b) Suppose that $\hat{k} > 1$ and $\Phi^{\hat{k}-1}(\hat{\mathbf{x}}) < -\epsilon(\hat{\mathbf{x}})$. By the continuity of $\Phi^{\hat{k}-1}(\cdot)$, there exists an $i_1 \geq i_0$ and a $\hat{\lambda}_1 > 0$ such that

$$\Phi^{\hat{k}-1}(\mathbf{x}_i + \lambda\mathbf{h}_i) \leq 0 \tag{5.3.15}$$

for all $i \geq i_1$, $i \in K$ and $\lambda \in [0, \hat{\lambda}_1]$. By repeating the arguments of the proof of part (a) we can show that there is an $i_2 \geq i_1$ and an $\hat{\lambda}_2 \geq \hat{\lambda}_1$ such that

$$\psi^{\hat{k}}(\mathbf{x}_i + \lambda_i\mathbf{h}_i) - \psi^{\hat{k}}(\mathbf{x}_i) \leq -\hat{\lambda}_2\hat{\epsilon}/2 < 0 \tag{5.3.16}$$

for all $i \geq i_2$, $i \in K$. Now, $\{\psi^{\hat{k}}(\mathbf{x}_i)\}_{i=0}^{\infty}$ is monotone decreasing and $\psi^{\hat{k}}(\mathbf{x}_i) \overset{K}{\to} \psi^{\hat{k}}(\hat{\mathbf{x}})$, since $\psi^{\hat{k}}(\cdot)$ is continuous, hence $\psi^{\hat{k}}(\mathbf{x}_i) \to \psi^{\hat{k}}(\hat{\mathbf{x}})$. But this contradicts (5.3.16). Hence we must have $\Phi^{\mu}(\hat{\mathbf{x}}) \leq 0$ and $0 \in \text{co}\{\partial f(\hat{\mathbf{x}}) \cup \{\partial\psi^j(\hat{\mathbf{x}})\}_{j \in J_0^{\mu}(\hat{\mathbf{x}})}\}$.

(c) Suppose that $\hat{k}>1$ and $\Phi^{\hat{k}-1}(\hat{\mathbf{x}}) \geq -\epsilon(\hat{\mathbf{x}})$. For this case, $\mathbf{h}_i$ is a descent direction for both $\psi^{\hat{k}}(\cdot)$ and $\Phi^{\hat{k}-1}(\cdot)$. Since by equation (5.3.9b), we have

$$<-\overline{\mathbf{h}}_{\epsilon(\mathbf{x}_i)}(\mathbf{x}_i), \overline{\xi}> \geq \|\overline{\mathbf{h}}_{\epsilon(\mathbf{x}_i)}(\mathbf{x}_i)\|^2 \tag{5.3.17}$$

for all $\overline{\xi} \in \overline{G}^{\hat{k}}_{\epsilon(\mathbf{x}_i)}(\mathbf{x}_i)$. Therefore

$$<-\mathbf{h}_{\epsilon(\mathbf{x}_i)}(\mathbf{x}_i), \xi> \geq \|\overline{\mathbf{h}}_{\epsilon(\mathbf{x}_i)}(\mathbf{x}_i)\|^2 \tag{5.3.18}$$

for all $\xi \in G_{\epsilon(\mathbf{x}_i)}\psi^{\hat{k}}(\mathbf{x}_i)$ and all $\xi \in \{G_{\epsilon(\mathbf{x}_i)}\psi^j(\mathbf{x}_i)\}_{j \in J^{\hat{k}}_{\epsilon(\mathbf{x}_i)}(\mathbf{x}_i)}$. Hence there must exist an $i_1 \geq i_0$ and a $\hat{\lambda}_1 > 0$ such that

$$\Phi^{\hat{k}-1}(\mathbf{x}_i + \lambda \mathbf{h}_i) \leq 0 \tag{5.3.19}$$

for all $i \geq i_1$, $i \in K$ and $\lambda \in [0, \hat{\lambda}_1]$. By repeating the arguments of the proof of part (a) we can show that there is an $i_2 \geq i_1$ and an $\hat{\lambda}_2 \geq \hat{\lambda}_1$ such that

$$\psi^{\hat{k}}(\mathbf{x}_i + \lambda_i \mathbf{h}_i) - \psi^{\hat{k}}(\mathbf{x}_i) \leq -\hat{\lambda}_2 \hat{\epsilon}/2 < 0 \tag{5.3.20}$$

for all $i \geq i_2$, $i \in K$. Now, $\{\psi^{\hat{k}}(\mathbf{x}_i)\}_{i=0}^{\infty}$ is monotone decreasing and $\psi^{\hat{k}}(\mathbf{x}_i) \xrightarrow{K} \psi^{\hat{k}}(\hat{\mathbf{x}})$, since $\psi^{\hat{k}}(\cdot)$ is continuous, hence $\psi^{\hat{k}}(\mathbf{x}_i) \to \psi^{\hat{k}}(\hat{\mathbf{x}})$. But this contradicts (5.3.20). Hence we must have $\Phi^{\mu}(\hat{\mathbf{x}}) \leq 0$ and $0 \in \mathrm{co}\{\partial f(\hat{\mathbf{x}}) \cup \{\partial \psi^j(\hat{\mathbf{x}})\}_{j \in J^{\delta}_{\delta}(\hat{\mathbf{x}})}\}$.

Case 2 : There exists an $\hat{i} \in \mathbb{N}_+$ such that for all $i \geq i_0$, $\Phi^{\mu}(\mathbf{x}_i) \leq 0$. Then the proof will be similar to case 1. The roles of functions $\psi^{\hat{k}}(\cdot)$ and $\Phi^{\hat{k}-1}(\cdot)$ in case 1 are the same as the functions $f(\cdot)$ and $\Phi^{\mu}(\cdot)$ in this case. Therefore we may conclude that we must have $\Phi^{\mu}(\hat{\mathbf{x}}) \leq 0$ and $0 \in \mathrm{co}\{\partial f(\hat{\mathbf{x}}) \cup \{\partial \psi^j(\hat{\mathbf{x}})\}_{j \in J^{\delta}_{\delta}(\hat{\mathbf{x}})}\}$. This completes our proof.

■

As mentioned before, we shall consider that $\psi^j(\cdot)$ might be only defined on the set

$$\widetilde{F}_{j-1} \triangleq \{\mathbf{x} \in \mathbb{R} \mid \psi^i(\mathbf{x}) \leq 0, \ i \in \underline{j-1}\} \tag{5.3.21}$$

In this case, the family of *m.a.c.d.f.* maps (5.3.9a) can be modified into the following two forms:

$$\overline{G}_\epsilon^{1,\hat{k}}(\mathbf{x}) \triangleq \mathrm{co}\left\{ \begin{bmatrix} \gamma_{\hat{k}} \Phi^{\hat{k}}(\mathbf{x})_+ \\ G_\epsilon f(\mathbf{x}) \end{bmatrix} \cup \begin{bmatrix} 0 \\ G_\epsilon \psi^j(\mathbf{x}) \end{bmatrix}_{j \in J_\epsilon^1(\mathbf{x})} \right\} \tag{5.3.22a}$$

$$G_\epsilon^{2,\hat{k}}(\mathbf{x}) \triangleq \begin{cases} \mathrm{co}\left\{ \left[G_\epsilon f(\mathbf{x})\right] \cup \left[G_\epsilon \psi^j(\mathbf{x})\right]_{j \in J_\epsilon^1(\mathbf{x})} \right\}, & \text{if } \hat{k} = \mu \text{ and } \psi^\mu \leq 0; \\ \mathrm{co}\left\{ \left[G_\epsilon \psi^j(\mathbf{x})\right]_{j \in J_\epsilon^1(\mathbf{x})} \right\}, & \text{otherwise.} \end{cases} \tag{5.3.22b}$$

and the ϵ- *search direction* in Algorithm 5.3.1 (5.3.9b) can be replaced by

$$\overline{\mathbf{h}}_\epsilon(\mathbf{x}) \triangleq (\mathbf{h}_\epsilon^0(\mathbf{x}), \mathbf{h}_\epsilon^1(\mathbf{x})) \triangleq \mathrm{argmin}\{ 1/2 \|\overline{\mathbf{h}}\|^2 \mid \overline{\mathbf{h}} \in \overline{G}_\epsilon^{1,\hat{k}}(\mathbf{x}) \} \tag{5.3.23a}$$

or

$$\mathbf{h}_\epsilon^2(\mathbf{x}) \triangleq \mathrm{argmin}\{ 1/2 \|\mathbf{h}\|^2 \mid \mathbf{h} \in G_\epsilon^{2,\hat{k}}(\mathbf{x}) \} \tag{5.3.23b}$$

$$\overline{\mathbf{h}}_\epsilon(\mathbf{x}) \triangleq (0, \mathbf{h}_\epsilon^2(\mathbf{x}))$$

Then, we have following algorithms:

Algorithm 5.3.2 : (Requires $\{G_\epsilon f(\cdot)\}_{\epsilon \geq 0}$, $\{G_\epsilon \psi^j(\cdot)\}_{\epsilon \geq 0}$, $j \in \underline{\mu}$ families of c.d.f. maps for $f(\cdot)$ and $\psi^j(\cdot)$).

Data: $\pi \in \{1, 2\}$, $\nu \in (0,1)$ for the set E in (5.3.9c), $\mathbf{x}_0 \in \mathbb{R}^n$.

Step 0: Set i=0.

Step 1: Compute $\hat{k}(\mathbf{x}_i)$ according to (5.3.5).

Step 2: Compute $\epsilon(\mathbf{x}_i)$ according to (5.3.9c) and the *search direction*

$$\mathbf{h}_i^\pi = -\mathbf{h}_{\epsilon(\mathbf{x}_i)}^\pi(\mathbf{x}_i), \tag{5.3.24a}$$

according to (5.3.23).

Step 3: Compute the *step size* as follows:

If $\psi^{\hat{k}}(\mathbf{x}_i) > 0$, then

$$\lambda_i \in \lambda_\Phi(\mathbf{x}_i) \triangleq \underset{\lambda \geq 0}{\operatorname{argmin}}\{\, \psi^{\hat{k}}(\mathbf{x}_i + \lambda \mathbf{h}_i^T) \mid \tag{5.3.24b}$$

$$\psi^j(\mathbf{x}_i + \lambda \mathbf{h}_i^T) \leq 0,\ \forall\ j \in \underline{\hat{k}-1}\,\},$$

If $\psi^{\hat{k}}(\mathbf{x}_i) \leq 0$, then

$$\lambda_i \in \lambda_{f,\Phi}(\mathbf{x}_i) \triangleq \underset{\lambda \geq 0}{\operatorname{argmin}}\{\, f(\mathbf{x}_i + \lambda \mathbf{h}_i^T) \mid \Phi^\mu(\mathbf{x}_i + \lambda \mathbf{h}_i^T) \leq 0\,\}\,. \tag{5.3.24c}$$

Step 4: Update:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda_i \mathbf{h}_i^T\,, \tag{5.3.24d}$$

replace i by $i+1$ and go to step 1.

∎

Then, the following result can be established. The proof will be similar to the proof of Theorem 5.3.1.

Theorem 5.3.2 : Suppose that Assumption 5.3.1 holds and that $\{\mathbf{x}_i\}_{i=0}^{\infty}$ is a sequence constructed by Algorithm 5.3.2 in solving (5.3.2) with $f(\cdot)$, $\psi^j(\cdot)$ l.L.c. and *regular*. If $\mathbf{x}_i \overset{K}{\rightarrow} \hat{\mathbf{x}}$ as $i \rightarrow \infty$, then $\Phi^\mu(\hat{\mathbf{x}}) \leq 0$ and

$$0 \in \operatorname{co}\left\{ \partial f(\hat{\mathbf{x}}) \bigcup \{\partial \psi^j(\hat{\mathbf{x}})\}_{j \in J_\delta^\mu(\hat{\mathbf{x}})} \right\}. \tag{5.3.25}$$

∎

Note that the family of *m.a.c.d.f.* maps (5.3.22a) is also suitable for the case of $F = \phi$ in equation (5.1.4). According to (5.1.5), the algorithm will be equivalent to solving the following problems sequentially:

$$\min\{\, f(\mathbf{x}) \mid \Phi^j(\mathbf{x}) \leq 0\,\}\,, \quad j \in \underline{\mu}\,. \tag{5.3.26}$$

The algorithm will stop at the phase $\hat{k} \in \underline{\mu}$ while

$$0 \in \operatorname{co}\left\{ \partial f(\hat{\mathbf{x}}) \bigcup \{\partial \psi^j(\hat{\mathbf{x}})\}_{j \in J_0^\mu(\hat{\mathbf{x}})} \right\}, \tag{5.3.27}$$

is satisfied.

5.4 Concluding Remarks

This chapter has shown that by modifying the family of *c.d.f.* maps of the conventional Phase I - Phase II methods, it is possible to construct multi-Phase algorithms which solve the problems (5.1.1) (5.3.2) sequentially. In the j-th Phase, $j \in \underline{\mu}$, the modification ensures that the search direction vector associated with each family of *m.a.c.d.f.* maps is a descent direction for $\psi^j(\mathbf{x})$ if $\psi^j(\mathbf{x}) > 0$ and is a feasible descent direction for all $\psi^i(\mathbf{x})$ if $j > 1$ and $i \in \underline{j-1}$. The algorithms can be initialized at any point in $\mathbb{R}^n$. In the initial iterations, Algorithm 5.3.1 and Algorithm 5.3.2 (when $\pi = 1$) concentrate on decreasing $\psi^j(\mathbf{x})$ sequentially while not completely ignoring the cost function, whose effect becomes progressively more pronounced as the feasible set is approached. Also Algorithms 5.3.2 can deal with the problems which might have an empty feasible set.

CHAPTER 6

DATA STRUCTURES FOR MULTIVARIABLE SYSTEMS

6.1. Introduction

The purpose of this chapter is to transcribe the class of control systems described in Chapter 3 and the associated design specifications studied in Chapter 4 into a data representation system which is called the MIMO database. In DELIGHT.MIMO, the database system plays an important role in the performance of the whole system. It not only manages control-related data and information, but also provides a channel for communication between the user interface, the MIMO simulator, the optimization algorithm library in DELIGHT and other existing CACSD packages through an I/O handler, see Fig. 1.1.

We have therefore attempted to construct a database for MIMO which is efficient and easily accessible, as well as easily modifiable and extendible. The MIMO database introduced in this chapter has the configuration shown in Fig. 6.1. Since disk access times are much slower than main storage access times † , all necessary data required by simulator and optimization algorithms is stored in main storage. At the same time, all this data can be stored in disc files using ASCII format. The existence of these ASCII files improves the accessibility, modifiability and extendibility of the database system. The MIMO database system provides a compiler/translator which converts ASCII files into data structures stored in the main storage or vice versa.

The database supports a hierarchical descriptions of control design problems. In this chapter, we describe the database system in a bottom-up manner. In Section 6.2, data structures for various parameterized expressions are presented. Then, in Section 6.3 and 6.4, we introduce data structures for parameterized linear subsystems and input signal generators. In Section 6.5, the interconnection of linear subsystems and

† Typical disk access times range from about 400 milliseconds or more for a floppy disk on a micro computer to about 30 milliseconds or less for a large, "fast" disk on a large mainframe; main storage access is likely to be at least four or five orders of magnitude faster than disk access on any given system, see, for example, reference [Dat.1].

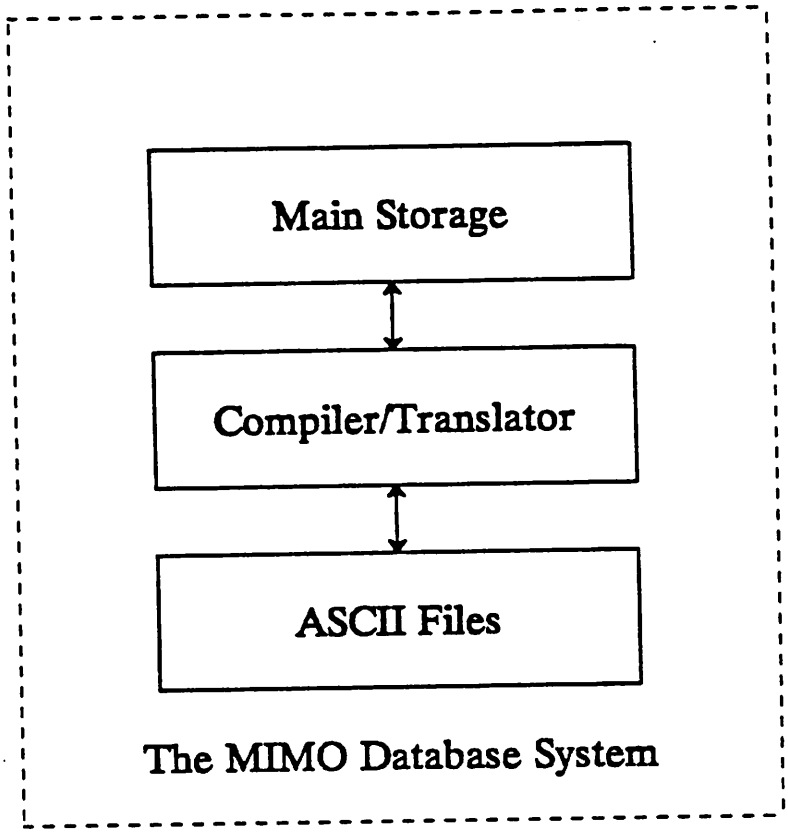


Fig. 6.1. The MIMO database system

input signal generators is described by a set of tables which give the geometry of the interconnection. In Section 6.6, data representations for various design constraints and objectives are described. Since the MIMO system is implemented in the C language, we use **"struct"** in C [Ker.1] to describe the data structures of the system.

6.2 Parameterized Expressions

The class of dynamic systems that will be considered in this dissertation are described by parameterized linear time-invariant ordinary differential equations and algebraic equations. As described in Chapter 2, the dynamics of each subsystem must be represented by the following state equations:

$$\dot{\mathbf{x}}_k(t) = \mathbf{A}_k(\mathbf{p})\mathbf{x}_k(t) + \mathbf{B}_k(\mathbf{p})\mathbf{u}_k(t), \tag{6.2.1a}$$

$$\mathbf{y}_k(t) = \mathbf{C}_k(\mathbf{p})\mathbf{x}_k(t) + \mathbf{D}_k(\mathbf{p})\mathbf{u}_k(t), \tag{6.2.1b}$$

where every component of the matrices $\mathbf{A}_k$, $\mathbf{B}_k$, $\mathbf{C}_k$ and $\mathbf{D}_k$ is a function in $\mathbf{p}$, the design parameter vector. In the simpliest case, each component of the matrices is a design parameter. For example, a subsystem with dimensions, 2 states, 1 input and 1 output, can be formulated as follows:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \mathbf{p}^1 & \mathbf{p}^2 \\ \mathbf{p}^3 & \mathbf{p}^4 \end{bmatrix}\mathbf{x}(t) + \begin{bmatrix} \mathbf{p}^5 \\ \mathbf{p}^6 \end{bmatrix}u(t), \tag{6.2.2a}$$

$$y(t) = [\mathbf{p}^7\ \mathbf{p}^8]\,x(t) + [\mathbf{p}^9]\,u(t), \tag{6.2.2b}$$

where $\mathbf{p} = [\mathbf{p}^1, \mathbf{p}^2, \cdots, \mathbf{p}^9]^T$ is the design parameter vector. For some examples, however, the components of the matrices may be multinomials (multivariable polynomials) or even rational functions which are defined as the ratio of two multinomials. For instance, a series electrical *RLS* network can be represented by the differential equation

$$Ri(t) + L\frac{di(t)}{dt} + \frac{1}{C}\int i(t)dt = v(t), \tag{6.2.3}$$

where R is the resistance, L the inductance, C the capacitance, $i(t)$ the current, and $v(t)$ the applied voltage. Equation (6.2.3) can easily be expressed as the state equation

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{1}{LC} & -\frac{R}{L} \end{bmatrix}\mathbf{x}(t) + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix}\mathbf{u}(t), \tag{6.2.4}$$

where $\mathbf{x}^1(t) = \int i(t)dt$, $\mathbf{x}^2(t) = \dot{\mathbf{x}}^1(t) = i(t)$ and the parameter vector $\mathbf{p} = [R, L, C]^T$. Therefore, some entries of the matrices in the equation (6.2.1a,b) may be rational functions in $\mathbf{p}$.

In this section, we are going to discuss the data structures of *design parameters*, *polynomials*, *multinomials* and *rational functions*.

6.2.1. Design Parameters

In MIMO, the design parameters are described by a linked list of data type, PARAMETER, defined as follows:

```
typedef struct Parameter {
    char *Name;
    double Value;
    struct Parameter *nextParam;
    } PARAMETER;
```

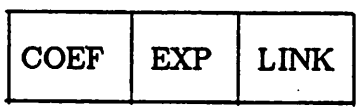
The name of the design parameter and its value are specified by **Name** and **Value**. The parameter list is global for all subsystems defined in MIMO, i.e, different subsystems may share the same parameters.

6.2.2 Polynomials and Multinomials

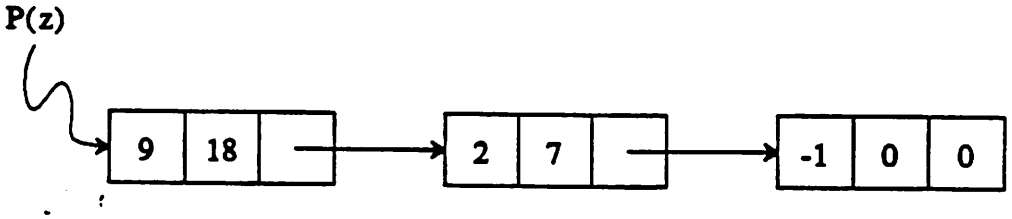
Our goal is to represent the polynomial

$$P(z) = a_m z^{e_m} + a_{m-1} z^{e_{m-1}} + \cdots + a_1 z^{e_1}, \tag{6.2.5}$$

where the $a_i \in \mathbb{R}$ are non-zero coefficients and the exponents $e_i \in \mathbb{Z}$ are such that $e_m > e_{m-1} > \cdots > e_1 \geq 0$. Easy manipulation of symbolic polynomials, such as the polynomial $P(z)$, is obtained by use of a linked list data structure [Hor. 1]. Each term is represented by a node which is of fixed size. The node has 3 fields: the coefficient a_i, the exponent e_i and a link pointer to the next term.



For instance, the polynomial $P(z) = 9z^{18} + 2z^7 - 1$ would be stored as



However, this data structure is not suitable for representing a multinomial such as

$$M(x,y,z) = x^{10}y^3z^2 + 2x^8y^3z^2 + 3x^8y^2z^2 + x^4y^4z + 6x^3y^4z + 2yz\,. \quad (6.2.6)$$

If we factor the multinomial in equation (6.2.6) as follows:

$$M(x,y,z) = ((x^{10} + 2x^8)y^3 + 3x^8y^2)z^2 + ((x^4 + 6x^3)y^4 + 2y)z\,, \quad (6.2.7)$$

we see that there are two terms in the variable z, $Cz^2 + Dz$, where $C = (x^{10} + 2x^8)y^3 + 3x^8y^2$ and $D = (x^4 + 6x^3)y^4 + 2y$ are multinomials in the variables x and y. Therefore, a multinomial $M(\mathbf{u},z)$ in the variables $(\mathbf{u},z)$, where $\mathbf{u}$ is a vector variable and z is a scalar variable, can be represented by the following equation

$$M(\mathbf{u},z) = M_m(\mathbf{u})z^{e_m} + M_{m-1}(\mathbf{u})z^{e_{m-1}} + \cdots + M_1(\mathbf{u})z^{e_1}, \quad (6.2.8a)$$

where the $M_i(\cdot)$ are multinomials in the variables $\mathbf{u}$ with exponents $e_i \in \mathbb{Z}$ such that $e_m > e_{m-1} > \cdots > e_1 \geq 0$. If $\mathbf{u} = (\mathbf{v},y)$, where $\mathbf{v}$ is a vector variable and y is a scalar variable, then (6.2.8a) can be reduced to the following recursive equation

$$M(\mathbf{u},z) = M_m(\mathbf{v},y)z^{e_m} + M_{m-1}(\mathbf{v},y)z^{e_{m-1}} + \cdots + M_1(\mathbf{v},y)z^{e_1}, \quad (6.2.8b)$$

Hence we can represent multinomials as a *generalized* linked list. Again, each term is represented by a node with the three fields as those in the nodes of the polynomial data structure. In the multinomial data structure, the coefficient field is a pointer to a multinomial coefficient. Since the multinomial in (6.2.8b) is recursively defined, each level has a data field which indicates the variable that has been factored out. The data structure of multinomials can be defined as follows:

```
typedef struct Polynomial {
    struct Parameter *Variable;
    struct PolynomialTerm *headTerm;
    } POLYNOMIAL;

typedef struct PolynomialTerm {
    double Coeff;
    struct Polynomial *CoeffPolyn;
    int Expon;
    struct PolynomialTerm *nextTerm;
    } POLYNOMIALTERM;
```

In the data structure, **PolynomialTerm**, if the **CoeffPolyn** is a null pointer, then the coefficient of this polynomial term is a scalar number stored in the data field **Coeff**. Hence, we can represent the multinomial (6.2.7) as shown in Fig. 6.2.

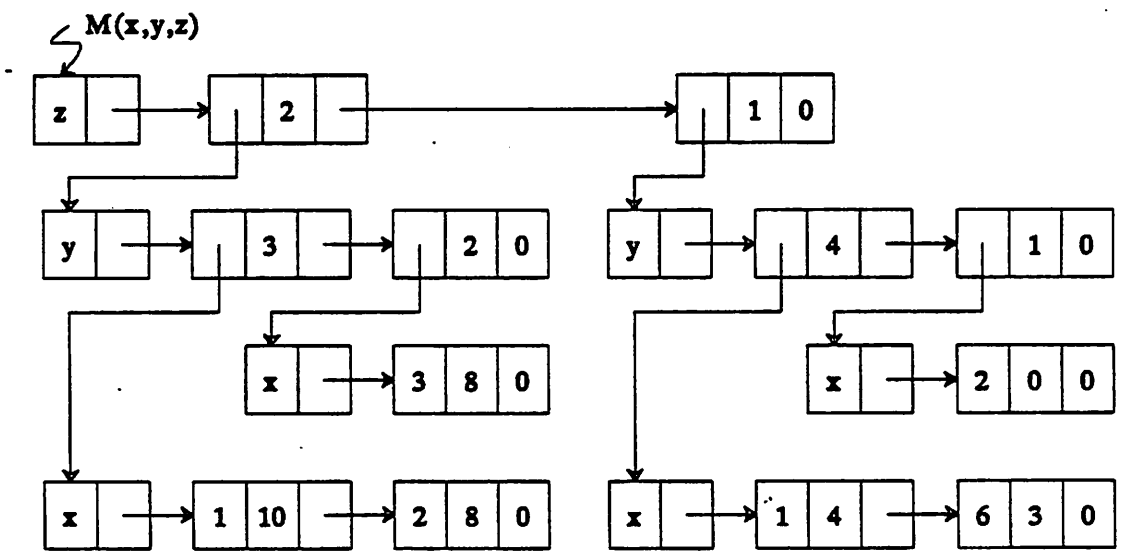


Fig. 6.2. data structure of a multinomial

6.2.3 Rational Functions

A rational function is defined as ratio of two multinomials:

$$Q(z) = \frac{M_n(z)}{M_d(z)}, \tag{6.2.9}$$

where $\mathbf{z}$ is a variable vector, and M_n, M_d are multinomials in the variable $\mathbf{z}$. The rational function Q can be described by the the data structure **RationalFunction**:

```
typedef struct RationalFunction {
    struct Polynomial *numPolyn;      /* numerator */
    struct Polynomial *denomPolyn;    /* denominator */
    } RATIONALFUNCTION;
```

where **numPolyn** and **denomPolyn** contain the pointers to the data structures which store the multinomial $M_n(\mathbf{z})$ and $M_d(\mathbf{z})$. If **numPolyn** is a null pointer, then the rational function is defined as $Q(\mathbf{z})=1/M_d(\mathbf{z})$. Conversely, if **denomPolyn** is a null pointer, then the rational function is a multinomial defined as $Q(\mathbf{z})=M_n(\mathbf{z})$.

6.2.4 Derivatives of Multinomials and Rational Functions

Sensitivity analysis is essential in the use of parametric optimization. When the parameter space becomes large, efficient simulation of sensitivity functions is very important.

Recall that equations (4.2.16d,e) and (4.3.66) involve the matrices, $\partial\mathbf{A}_k(\mathbf{p})/\partial\mathbf{p}^i$, $\partial\mathbf{B}_k(\mathbf{p})/\partial\mathbf{p}^i$, $\partial\mathbf{C}_k(\mathbf{p})/\partial\mathbf{p}^i$ and $\partial\mathbf{D}_k(\mathbf{p})/\partial\mathbf{p}^i$ for different design parameter values. For the purpose of efficient evaluation of sensitivity functions, we will store not only the parameterized matrices, $\mathbf{A}_k(\mathbf{p})$, $\mathbf{B}_k(\mathbf{p})$, $\mathbf{C}_k(\mathbf{p})$ and $\mathbf{D}_k(\mathbf{p})$, but also the matrices, $\partial\mathbf{A}_k(\mathbf{p})/\partial\mathbf{p}^i$, $\partial\mathbf{B}_k(\mathbf{p})/\partial\mathbf{p}^i$, $\partial\mathbf{C}_k(\mathbf{p})/\partial\mathbf{p}^i$ and $\partial\mathbf{D}_k(\mathbf{p})/\partial\mathbf{p}^i$ in a symbolic form.

If we assume that the entries of the matrices $\mathbf{A}_k(\mathbf{p})$, $\mathbf{B}_k(\mathbf{p})$, $\mathbf{C}_k(\mathbf{p})$ and $\mathbf{D}_k(\mathbf{p})$ are either multinomials or rational functions, then the symbolic form of each entry of these matrices can be stored in the data structures **POLYNOMIAL** or **RATIONALFUNCTION**. By the assumption, the entries of $\partial\mathbf{A}_k(\mathbf{p})/\partial\mathbf{p}^i$, $\partial\mathbf{B}_k(\mathbf{p})/\partial\mathbf{p}^i$, $\partial\mathbf{C}_k(\mathbf{p})/\partial\mathbf{p}^i$ and $\partial\mathbf{D}_k(\mathbf{p})/\partial\mathbf{p}^i$ are either multinomials or rational functions. Since the derivatives of these expressions with respect to *some* parameters $\mathbf{p}^i$ are zeros, we need to store those derivatives which are not zeros. We now have the following linked list data structures:

```
typedef struct DPolynomial {
    struct Parameter *dParam;          /* DPolynomial/dParam */
    struct Polynomial *Polyn;          /* result of differentiation */
    struct DPolynomial *nextDPolyn;
    } DPOLYNOMIAL;

typedef struct DRationalFunction {
    struct Parameter *dParam;
    struct RationalFunction *ratFunc;
    struct DRationalFunction *nextDRatFunc;
    } DRATIONALFUNCTION;
```

These data structures can be showns as the following diagram:

For instance, the derivatives of the multinomial in (6.2.6) would be stored as

$$\frac{\partial M(x,y,z)}{\partial x} \rightarrow \frac{\partial M(x,y,z)}{\partial y} \rightarrow \frac{\partial M(x,y,z)}{\partial z}$$

A symbolic differentiator is constructed using the data structures of multinomials and rational functions mentioned in this section. The details of the symbolic differentiator are described in Appendix A.

6.3 Linear Subsystems - State Space Description

The linear subsystems which will be considered in this dissertation are described by the following equations:

$$\frac{d}{dt}\begin{bmatrix} \mathbf{x}(t,\mathbf{p}) \\ \frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i} \end{bmatrix} = \begin{bmatrix} \mathbf{A}(\mathbf{p}) & 0 \\ \frac{\partial \mathbf{A}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{A}(\mathbf{p}) \end{bmatrix}\begin{bmatrix} \mathbf{x}(t,\mathbf{p}) \\ \frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i} \end{bmatrix} + \begin{bmatrix} \mathbf{B}(\mathbf{p}) & 0 \\ \frac{\partial \mathbf{B}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{B}(\mathbf{p}) \end{bmatrix}\begin{bmatrix} \mathbf{r}(t,\mathbf{p}) \\ \frac{\partial \mathbf{r}(t,\mathbf{p})}{\partial \mathbf{p}^i} \end{bmatrix}, \quad (6.3.1a)$$

$$\begin{bmatrix} \mathbf{y}(t,\mathbf{p}) \\ \frac{\partial \mathbf{y}(t,\mathbf{p})}{\partial \mathbf{p}^i} \end{bmatrix} = \begin{bmatrix} \mathbf{C}(\mathbf{p}) & 0 \\ \frac{\partial \mathbf{C}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{C}(\mathbf{p}) \end{bmatrix}\begin{bmatrix} \mathbf{x}(t,\mathbf{p}) \\ \frac{\partial \mathbf{x}(t,\mathbf{p})}{\partial \mathbf{p}^i} \end{bmatrix} + \begin{bmatrix} \mathbf{D}(\mathbf{p}) & 0 \\ \frac{\partial \mathbf{D}(\mathbf{p})}{\partial \mathbf{p}^i} & \mathbf{D}(\mathbf{p}) \end{bmatrix}\begin{bmatrix} \mathbf{r}(t,\mathbf{p}) \\ \frac{\partial \mathbf{r}(t,\mathbf{p})}{\partial \mathbf{p}^i} \end{bmatrix}. \quad (6.3.1b)$$

The minimum information which should be stored in the data structure consists of the *name* and the *dimension* of the system as well as the system matrices $\mathbf{A(p)}$, $\mathbf{B(p)}$, $\mathbf{C(p)}$, $\mathbf{D(p)}$, $\partial\mathbf{A(p)}/\partial\mathbf{p}^i$, $\partial\mathbf{B(p)}/\partial\mathbf{p}^i$, $\partial\mathbf{C(p)}/\partial\mathbf{p}^i$, and $\partial\mathbf{D(p)}/\partial\mathbf{p}^i$. For time domain simulation, the initial state vector $\mathbf{x}_0$ should be given as well. Each entry of the matrices, **A**, **B**, **C** and **D**, may be one of four data types: *scalar, parameter, multinomial* and *rational function.* Including the derivatives of the expressions of these four data types we may define a *union* data type [Ker.1] as follows:

```
typedef struct Polyn_and_Derv {
    struct Polynomial *Polyn;
    struct DPolynomial *dPolyn;        /* derivative of Polyn */
    } POLYN_AND_DERV;

typedef struct RatFunc_and_Derv {
    struct RationalFunction *RatFunc;
    struct DRationalFunction *dRatFunc;
    } RATFUNC_AND_DERV;

union u_elm {
    double Number;
    struct Parameter *Param;
    struct Polyn_and_Derv *polynDerv;
    struct RatFunc_and_Derv *ratfuncDerv;
    };
```

Now, the equations (6.3.1a,b) can be represented by the data structure, LINEARSYSTEM, defined as follows:

```
typedef struct Linearsystem  {
    char  *Name;
    int   States;
    int   Inputs;
    int   Outputs;
    union u_elm **A;
    union u_elm **B;
    union u_elm **C;
    union u_elm **D;
    int  **utypeA;
    int  **utypeB;
    int  **utypeC;
    int  **utypeD;
    union u_elm *Initialstate;
    int  *utypeInit;
    struct Constraints *Costs;
```

```
struct Constraints *Constraints;
struct Linearsystem *nextSubsys;
} LINEARSYSTEM;
```

utypeA[i][j] holds the data type of the element **A**[i][j]. The possible values for the data type are *SCALAR, PARAMETER, MULTINOMIAL, RATIONALFUNC.* The fields, **Costs** and **Constraints**, in the struct **Linearsystem** will be discussed in a later section. The **nextSubsys** field is a link pointer to the next linear subsystem. This linked list of the data structure **Linearsystem** forms a linear subsystem pool as shown in Fig.6.3.

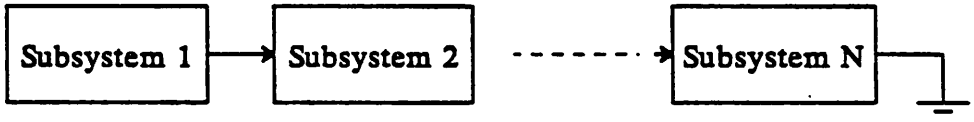


Fig. 6.3. Subsystem Pool

6.4 Input Signal Generators

Recall Section 2.5 that the class of inputs to the state equations are generated by a special class of linear systems which are excited by a polynomial function in the variable t (time). For multi-input signals, it is easier to define input signals channel by channel. Each channel can be described by a single-input single-output (SISO) state equation with a polynomial input function in t. The input signal generator for each channel can be represented by the following date structures:

```
typedef struct Channel  {
    int channelID;
    struct Linearsystem *subSystem;
    struct Polynomial *inputPolyn;
    struct Channel *nextChannel;
    } CHANNEL;
```

For instance, an exponential function $y(t)=1-e^{-\lambda t}$ can be generated by

$$\dot{x}(t) = -\lambda x(t)+1 \tag{6.4.1a}$$

$$y(t) = \lambda x(t) \tag{6.4.1b}$$

By making use of the **nextChannel** field in the struct **Channel**, we may form a linked list of input channels which form a multi-input signal system. Therefore, the multi-input signal system can be represented by the following data structure:

```
typedef struct InputSignal  {
    char *Name;
    int Channels;
    struct Channel *headChannel;
    struct InputSignal *nextInput;
    } INPUTSIGNAL;
```

Again, the linked list of the data structure **InputSignal** forms an input signal pool as shown in Fig. 6.4.

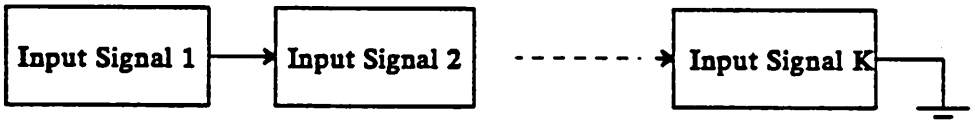


Fig. 6.4. Input Signal Pool

6.5 Interconnected Systems

We now describe a data structure for representing the interconnection of subsystems defined in Section 2.2. The interconnection of linear subsystems can be described by a set of tables which give the geometry of the interconnection. Most of this description is based on an example which demonstrates the features of these tables.

An example of interconnected systems is shown in Fig.6.5, in which the linear systems S_1 to S_8 may contain free parameters, and the systems R_1, R_2 and R_3 are

test input signals for the interconnection.

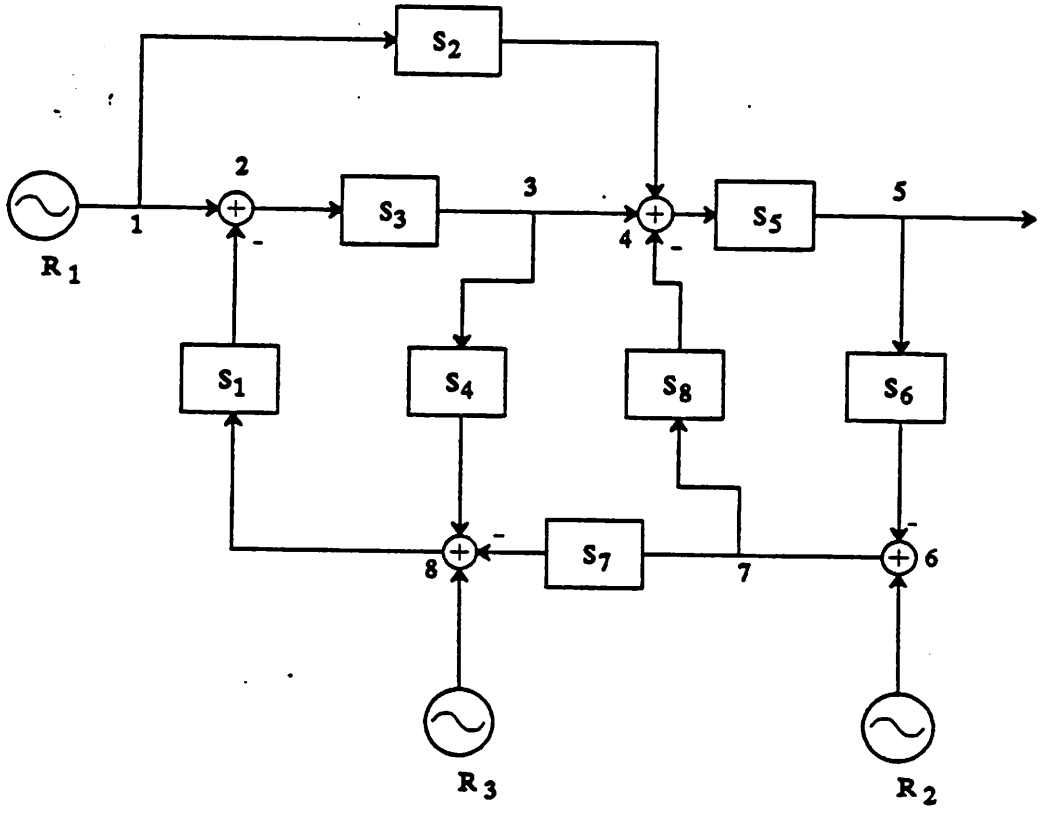


Fig. 6.5. An Interconnected System

Interconnections such as the one shown above can be described precisely by associating nodes with *summing junctions*, *branch points* and points *where two systems join in tandem*. A set of such node numbers is shown in the interconnection above. Tables can now be set up giving the geometry of the interconnection in terms of these node numbers.

In setting up these tables, it should be noted that associated with each of the systems S_1 to S_8, are a pair of nodes, namely an input node and an output node, and a transmission sign. Thus, for example, system S_4 has an input node 3 and output node 8, and its transmission sign is plus, while system S_8 has an input node 7 and output node 4, and its transmission sign is minus. This information can be summarized in the following *Link Table*:

Link Table			
System Name	From Node	To Node	Sign
S2	1	4	+
S3	2	3	+
S4	3	8	+
S1	8	2	-
S5	4	5	+
S8	7	4	-
S7	7	8	-
S6	5	6	-
unity	1	2	+
unity	6	7	+
unity	3	4	+

Note that each of the systems S_1 to S_8 may be a linear subsystem or an interconnected system. Also, the system "*unity*" is a special branch with unity gain.

Since input signal generators are not subject to inputs from any other systems, an output node is only associated with an input signal generator. For example, the output node associated with R_1 is 1, while the output node associated with R_2 is 6. This information can be described by the following *Input Table*:

Input Table		
Input Signal	To Node	Sign
R1	1	+
R2	6	+
R3	8	+

The link tables and input tables can be represented by the following data structures:

```
typedef struct LinkTable {
    struct Linearsystem *subSystem;
    struct InterConnectSys *interSys;
    int FromNode;
    int ToNode;
    char Sign;
    struct LinkTable *nextLinkBranch;
    } LINKTABLE;

typedef struct InputTable {
    struct InputSignal *inputSignal;
    int ToNode;
    char Sign;
    struct InputTable *nextInputBranch;
    } INPUTABLE;
```

In general, the information held in a link table and an input table are enough for the description of the geometry of interconnection. However, for the purpose of generating editing capability in a graphical mode, an extra table is added to hold the node number of the current existing nodes and the corresponding node type (*summing node* or *branch node*). This can be described by the struct **NodeTable**:

```
typedef struct NodeTable  {
    int NodeId;
    int NodeType;
    struct NodeTable *nextNode;
    } NODETABLE;
```

Now, the interconnected systems can be represented by the following data structure, **InterConnectSys**:

```
typedef struct InterConnectSys {
    char *Name;
    int InputNode;
    int OutputNode;
    struct LinkTable   *headLinkBranch;
    struct LinkTable   *unityLinkBranch;
    struct InputTable  *headInputBranch;
    struct NodeTable   *headNode;
    struct Constraints *Costs;
```

```
struct Constraints *Constraints;
struct InterConnectSys *nextInterSys;
} INTERCONNECTSYS;
```

The data structure **InterConnectSys** consists of 3 tables: the link table, the input table and the node table. The **unityLinkBranch** field is a pointer which points to the head unity branch in **LinkTable**. The fields, **InputNode** and **OutputNode**, specify the input and output nodes of the interconnected system, while the system is embedded in another interconnected system as a two port system branch. The **Costs** and **Constraints** fields in the data structure **InterConnectSys** will be discussed in the next section. Again, the linked list of the data structure **InterConnectSys** forms an interconnected system pool, as shown in Fig. 6.6.

6.6 Costs and Constraints

In Chapter 3, we have discussed the techniques involved in control system design via semi-infinite optimization. It has been shown that time and frequency domain specifications of control system designs can be formulated as semi-infinite inequality constraints or costs, such as in (3.3.7a,b), (3.3.11a,b,c) and (3.3.4).

In this section, we attempt to describe how these constraint and cost functions are represented. However, since cost functions and constraint functions have similar mathematical expressions, it is sufficient to describe only the data representations of the constraint functions.

6.6.1 Input/Output Expressions

Before a constraint is defined, one must specify the input/output of the system Σ on which a constraint is imposed. For instance, a time domain constraint is given as follows:

$$\mathbf{y}_c(t,\mathbf{p}) \leq b(t), \qquad \forall\ t \in [0,T]. \tag{6.6.1a}$$

where $\mathbf{y}_c$ stands for the output of the controller C. Sometimes, we want to impose constraints on the components of the vector function $\mathbf{y}_c$ as follows:

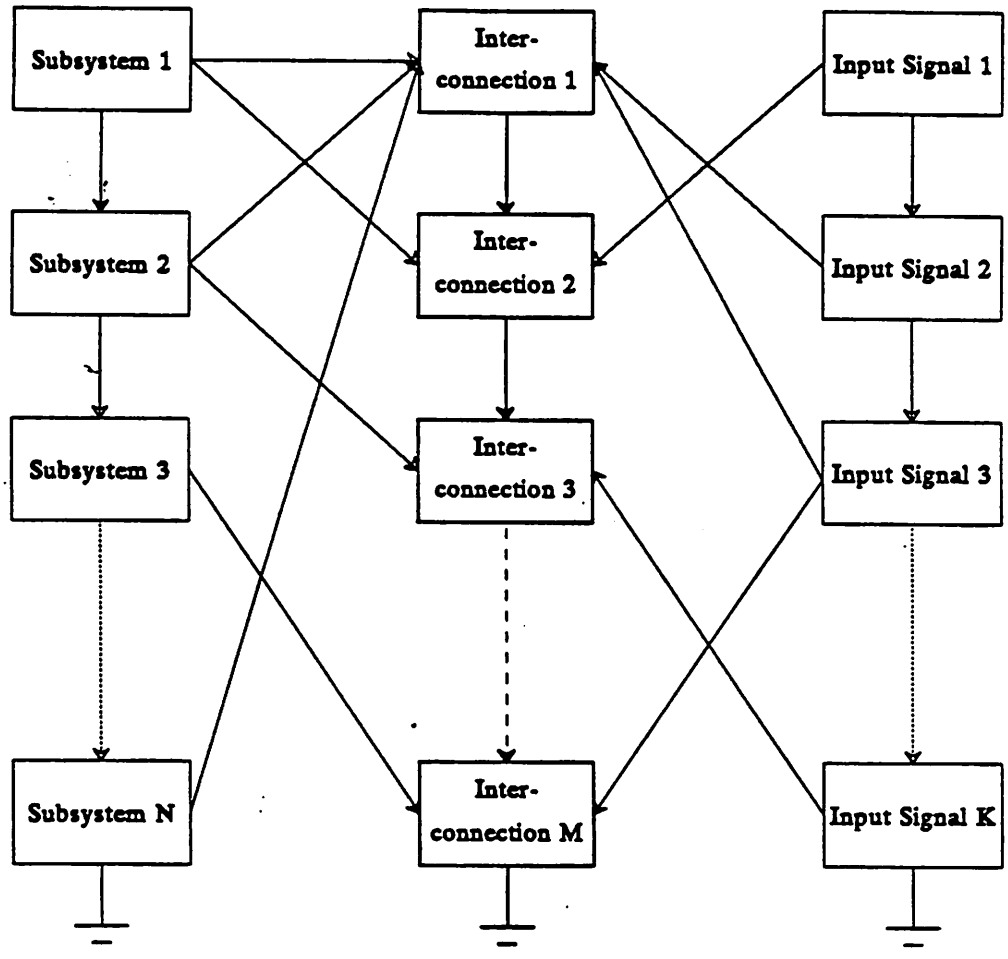


Fig. 6.6. Interconnected Systems Pool

$$\mathbf{y}_c^{i;j}(t,\mathbf{p}) \leq b(t), \qquad \forall\ t \in [0,T]. \tag{6.6.1b}$$

where $\mathbf{y}_c^{i;j}$ are the ith and jth components of $\mathbf{y}_c$. Therefore a data structure for representing the input/output expression is given as follows:

```
typedef struct mimoIO  {
    struct Linearsystem *subSystem;
    int IOSchoice;
    int *indexList;
    char *exprStr;
    } MIMOIO;
```

The field, **IOSchoice**, holds the values of *INPUT*, *OUTPUT* or *STATE*, which indicates the input, the output or the state of the linear subsystem being constrained. If only the components of the input, the output or the state vector are constrained, **indexList** stores a pointer to a list of these components.

6.6.2 Time Domain Constraints

There are three types of time domain constraints in DELIGHT.MIMO: *step* response constraints, *constant* constraints and *piecewise* linear constraints. The step responses constraints are specified in terms of *rise time, settling time, final time, peak amplitude, rise amplitude, settling amplitude ratio, and DC gain* as shown in Fig.6.7. This can be represented by the following data structure **StepConstr**.

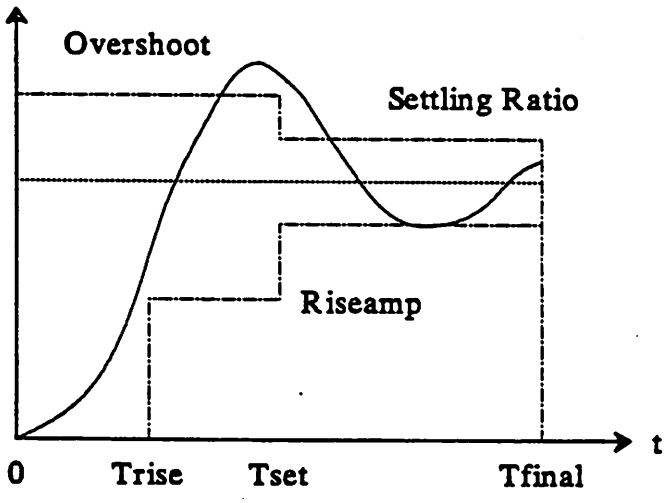


Fig. 6.7. Step Response Constraint

```
typedef struct StepConstr  {
    double Trise;
    double Tset;
    double Tfinal;
    double OshAmp;
    double RiseAmp;
    double SetRatio;
    double DCgain;
    } STEPCONSTR;
```

The constant constraints shown in Fig. 6.8 can be represented by the following data structure **ConstantConstr**.

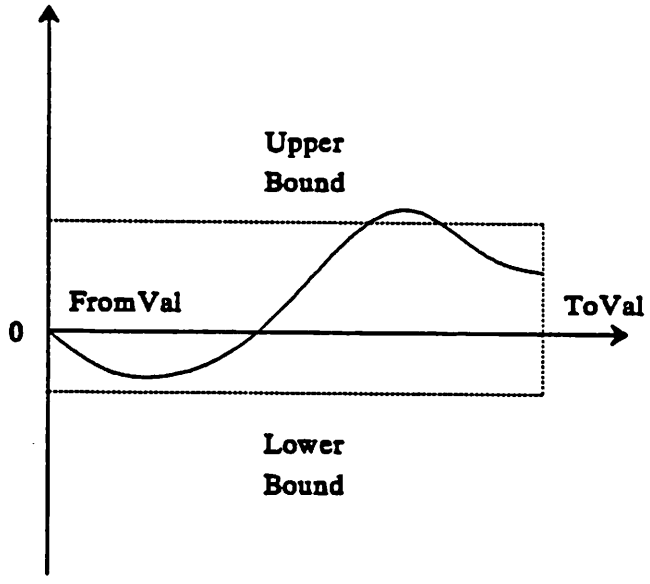


Fig. 6.8. Constant Constraint

```
typedef struct ConstantConstr {
    double UpperBound;
    double LowerBound;
    double FromVal;
    double ToVal;
    } CONSTANTCONSTR;
```

The piecewise linear constraints shown in Fig. 6.9 can be represented by the following data structure **PieceWiseConstr**.

```
struct XYPoint  {                          /* list of (x,y) points */
    double XValue;
    double YValue;
    struct XYPoint *nextXYPoint;
    };

typedef struct PieceWiseConstr {
    struct XYPoint *UheadXYPoint;    /* upper bound */
    struct XYPoint *LheadXYPoint;    /* lower bound */
    } PIECEWISECONSTR;
```

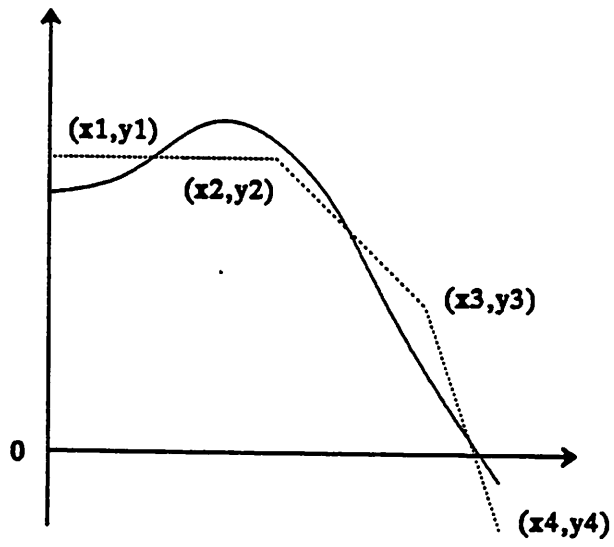


Fig. 6.9. Piecewise Linear Constraint

Now, we may represent the time domain constraints as the following data structure:

```
typedef struct TimeConstraint {
    struct mimoIO *ioExpr;
    int constrType;
    struct StepConstr *Step;
    struct ConstantConstr *Constant;
    struct PieceWiseConstr *PieceWise;
    struct TimeConstraint *nextTimeConstr;
    } TIMECONSTRAINT;
```

ioExpr stores the pointer to the input/output expression of the time domain constraint. **constrType** specifies the constraint type of the time domain constraint. The possible values for the constraint type are *STEP, CONSTANT* and *PIECEWISE*. **Step, Constant** and **PieceWise** are the pointers to the step response, constant and piecewise linear constraints respectively. According to the value of **constrType**, one of the pointers, **Step, Constant**, and **PieceWise** will be used for storing the constraint specifications.

6.6.3 Frequency Domain Constraints

There are two types of frequency domain constraints in DELIGHT.MIMO: *constant* constraints and *piecewise* linear constraints. The data structures of these two constraints are the same as time domain constraints. Therefore, the frequency domain constraints can be described by the following data structure, **FreqConstraint**:

```
typedef struct FreqConstraint {
    struct mimoIO *inputExpr, *outputExpr;
    int constrType;
    struct PieceWiseConstr *PieceWise;
    struct ConstantConstr *Constant;
    struct FreqConstraint *nextFreqConstr;
    } FREQCONSTRAINT;
```

6.6.4 Box Constraints on Design Parameters

There are physical constraints which must be taken into consideration in the implementation of real systems. The most common constraints are on the values of design parameters, which have the form of the following box constraints:

$$\underline{b}^i \leq p^i \leq \overline{b}^i, \qquad \text{for some } i. \tag{6.6.2}$$

where $\underline{b}^i$ and $\overline{b}^i \in \mathbb{R}$. The box constraints can be represented by the data structure **BoxConstraint**.

```
typedef struct BoxConstraint {
    struct Parameter *Param;
    double UpperBound;
    double LowerBound;
    struct BoxConstraint *nextBoxConstr;
    } BOXCONSTRAINT;
```

6.6.5 Stability Constraints

In DELIGHT.MIMO, we use the modified Nyquist criterion for a stability test in optimization-based design. In this section, a data structure is designed to hold necessary information for applying the modified Nyquist criterion. We first assume that the

S set in Theorem 3.2.4 is described in (3.2.24c) which can be represented by the following data structure:

```
typedef struct SSet {
    double DecayRate;
    double DampAngle;
    } SSET;
```

We also assume that the set $\mathbf{B} = \mathring{\mathbb{C}}_+$. Then, the modified Nyquist criterion has the form of (3.2.27a,b,c,d,e) in Example 3.2.3. The equations (3.2.27a,b,c) can be represented by **BoxConstraint** and the equation (3.2.27d) can be described by the following data structure

```
struct QuadraticConstr {                    /* ax^2+ by+ c<0 */
   struct Parameter *xParam, *yParam;
   double aCoeff, bCoeff, cCoeff;
   struct QuadraticConstr *nextQuadConstr;
   };
```

Now, we have the following data structure which represents the modified Nyquist stability constraint

```
typedef struct NyqstConstraint {
    int denomDegree;
    struct Parameter **denomParam;
    struct BoxConstraint   *headBoxConstr;
    struct QuadraticConstr *headQuadConstr;
    struct SSet *eignConstr;
    } NYQSTCONSTRAINT;
```

where **denomParam** stores values of the parameter vector **q** in (3.2.27e).

Now, to summarize all the data structures of various constraints, we have the following data structure **CONSTRAINTS**.

```
typedef struct Constraints {
    struct TimeConstraint  *headTimeConstr;
    struct FreqConstraint  *headFreqConstr;
    struct NyqstConstraint *nyquistConstr;
    struct EigenConstraint *eignConstr;
```

```
struct BoxConstraint    *headBoxConstr;
} CONSTRAINTS;
```

Note that both **Linearsystem** and **InterConnectSys** data structures contain pointers, **Costs** and **Constraints**, which point to the data structure **CONSTRAINTS**.

CHAPTER 7

USER INTERFACE

7.1. Introduction

Designing a human interface to a computer system is more art than science. It requires taking into account human factors in computing systems, as well as the differences between physical and conceptual aspects of the interface, see references [Fol.1, Mon.1]. The physical hardware (keyboard, display screen, tablet stylus, mouse and so on) affects the way in which the system behind the interface may be used. It does not affect conceptual power of the interface, however. From our point of view, a user interface is a mediator which handles transactions between users and application programs. The major tasks of the MIMO user interface are database access, query and update, screen I/O construction and CAD computing environment control.

In this chapter we discuss the user interface which we designed for the MIMO system. The plan of this chapter is as follows. We begin with an overview of existing user interface features and a summary of general design principles for friendly user interfaces in Section 7.2. By examining the needs of various users to be supported by the MIMO system, Section 7.3 results in a set of design criteria for the MIMO user interface. Section 7.4 then surveys the MIMO user interface features which help to meet these design criteria.

7.2. Overview of User Interface Design

Since it is difficult to survey existing user interfaces in all application areas, we restrict ourselves to a few MIMO related areas. We first review the application in the area of database systems. A wide variety of computer applications falls into the category of *Interactive Information Systems* (IIS) [Row.3], sometimes they are called transaction processing systems. These applications allow the user to access, query and update data stored in a database. The applications do not involve much computation,

but they do involve significant user interaction with the application. Typical examples of on-line IIS are: an airline reservations system, a retail banking system, or an accounting system. A survey of the tools available for developing IIS is given in reference [Row.1]. It concludes that form-based interfaces or structure editors are easier for users to learn.

A form, sometimes called a screen, consists of a collection of fields in which the application displays data and the user enters or edits data. Basically, the form-based interface is a screen definition system which defines forms that are displayed on a terminal screen with fields where data can be entered by the user or displayed by the application program. There are three ways to define forms: a textual specification, a question and answer dialogue, and a what-you-see-is-what-you-get (WYSIWYG) form editor.

A textual specification uses a language to define the fields and literals. The user edits the definition of a form with a conventional text editor. A compiler translates this specification into a run-time representation that is loaded with the program and used by the screen I/O run-time system.

The second approach is to engage the application user in a question and answer session that defines the form. The advantage of this approach is that the user is prompted for the information needed to define a form; he or she does not have to remember the syntax of the specification language. This approach makes changing a previously defined form difficult, however, and the question and answer paradigm can be tedious.

The last approach is a WYSIWYG form editor that allows the user to edit the form definition while it is displayed on the terminal. Interfaces of this type not only allow a user to *browse* through a database *visually* † , but also to navigate the user through the database, making ad-hoc changes. Contrast this approach with the compiled textual language approach, where the user makes several changes, compiles the

† A simple illustration of a browsing program is a full screen, visual text editor [Joy.1].

form, runs a program, and then discovers that the form does not look right. Now he or she has to go through the edit-compile-run cycle again to fix the problem. Switching back and forth between the text editor and the program displaying the form is slow, so the user is less productive than if he or she could see the effect of his or her changes immediately. However, the user may tire of using a WYSIWYG form editor to define some repeated data format. Nevertheless, a WYSIWYG form editor is generally the best approach to problem definition.

The interaction between the user and the computer system can be carried out in two basic *driven modes*: command driven mode and menu driven mode. The command driven mode uses some specialized user-interaction language which is called a *command language*, while the menu driven mode lists a set of operations that the user can execute. In some application areas, command language can be easily extended or enhanced by using an macro or soft key, and menu operations can be modified or *opened* is such a way that one can create new menu items or redefine an old menu item.

We now address several issues involved in managing terminal display while running application programs. In some application areas, such as VLSI design, the data or information needed to be displayed is enormous. Screen I/O constructions become important in the user interface. In order to make computers easier to use and more versatile, many user interfaces try to explore the use of multiple windows on a single screen and multiple coordinated screens in a single work station displaying linked or related information. Detailed window layouts, in standard positions on the terminal screen, outlined with boxes, and having specialized functions, improve clarity. The user interaction process does not totally monopolize the terminal display. The computational and graphical results produced by application programs are as important as the user interaction process. A secondary window may temporarily appear for some of the menu selection processes, and once selection has occurred, this window disappears so that application programs may use the screen space. Multiple overlapped window layouts which display various results produced by application programs can be easily defined by users.

There are various general design principles that lead to friendly user interfaces, see for example [Fri.1]. We now conclude this section by briefly enumerating these principles:

1. A minimal effort should be required of the user to operate the system. The user should do only operations that are absolutely essential.
2. The user should be required to memorize as little information as possible while using the system. The system should provide memory aids for the user.
3. The system design should lead to as little user frustration as possible. This is particularly important for advanced users.
4. The system should take maximal advantage of user habits and patterns. Consistent organization of the terminal display and functional key assignments is important here.
5. The system should have maximal tolerance for human differences among users.
6. The system should be tolerant of changes in the hardware and software system environment.
7. When problems occur, the user should be promptly notified. Problems should be anticipated wherever possible.
8. The user should be given maximum control of CAD computing tasks.
9. Support for user tasks should be provided in the form of on-line structured documentation.

7.3. Design Criteria for the MIMO User Interface

The general design goal for the MIMO user interface is to allow both novice and experienced users to make efficient use of MIMO's facilities. The interface should provide enough instructions and information for beginners, while allowing advanced users to bypass unnecessary procedures during the interaction with the computer systems.

In this section, we summarize the previous section and characterize the needs of a DELIGHT.MIMO user by listing the set of design criteria for the MIMO user interface.

- The mixing of different user interaction styles with the MIMO system.
- The importance of a clear, easy-to-use description language for user interaction.
- The consistency of the various interfaces.
- Management of the terminal display during the user interaction process.
- User on-line HELP facilities for the MIMO system.
- The need to provide high level graphics tablet facilities to support recognition of symbols drawn by the user at the tablet.
- Transportability of the user interface processor to different computing environments.

In the next section we survey various features of the MIMO user interface which help the interface to meet these design criteria.

7.4. MIMO User Interface Features

We have considered various features in some existing user interfaces and identified some tradeoffs among these features in Section 7.2. In this section, we survey features in the MIMO user interface that help to meet the design goal and criteria given in Section 7.3.

7.4.1. User Interaction Styles

One of the most important aspects of a user command processor is the variety of user interaction styles supported by the processor. Not only should each user be able to find a style that he is comfortable with, but also users should be able to choose

among a variety of styles. The generality and interchangeability of user interaction styles are particularly important. Different methods of communicating with the computer are appropriate for different tasks. The MIMO user interface attempts to widen the communication bandwidth and to fill these needs by providing a variety of devices for command input, consistent and clear interaction concepts, and the ability to mix the application program dialogue with complete on-line HELP facilities for the MIMO system.

Two interactive modes are provided by the MIMO user interface. One is the *Command* driven mode and the other is the *Menu/Form* driven mode. The MIMO system allows the user to move between these two modes freely.

7.4.1.1. Input Devices Supported by the User Interface

There are three input devices supported by the MIMO user interface. We now describe the functions and usages of these devices in detail.

Alphanumeric Keyboard

The most general input device supported by most terminals is the alphanumeric keyboard. At startup, the MIMO system is in command driven mode and waiting for the alphanumeric keyboard input. A hierarchical structure of command language has been designed based on the alphanumeric keyboard input. The language will be described in the next section.

Mouse/Tablet and Stylus Input

Since *MENU* selection is essentially a *pick* operation (in GKS terminology - see [Hop.1]), a pointing device is the most convenient method of selecting *MENU* entries. The graphics mouse/tablet stylus may be used for such operations. At startup time, the user must inform the interface that the mouse/tablet stylus will be used for user interaction. Then, the MIMO system will initialize the screen and enter the menu/form driven mode. Once in the menu/form driven mode, the terminal graphics

cursor will track the position of the mouse/tablet stylus. To digitize a point or select a *MENU* item on the screen, the mouse/tablet stylus is moved so that the graphics cursor is at the desired screen position or is within the desired *MENU* pushbutton. Then, the mouse/tablet stylus is pressed to digitize or to select the *MENU* item.

Thumbwheels/Graphical Cursor Movement Keys

The thumbwheels/graphical cursor movement keys may be used as a substitute for a tablet and stylus/mouse on terminals that lack tablets. However, pointing operations using the thumbwheels/graphical cursor movement keys (*UP, DOWN, LEFT, RIGHT,* and, *CURSOR FAST*) are considerably more cumbersome than operations using the tablet stylus/mouse. Furthermore, once the graphical cursor is correctly positioned using the thumbwheels/cursor movement keys, the user must type the *PRESS* command to digitize that point and activate the MIMO system.

Basically, the alphanumeric keyboard will be the major input device in the command driven mode. While in the menu/form mode, the graphical input devices will be the dominate input devices.

7.4.1.2. Mixing Interaction Modes

As described before, the MIMO system provides two interactive modes: the command driven mode and the menu/form driven mode. The MIMO system allows the user to move between these two modes freely. At startup, the MIMO system is in the command mode. To enter the menu/form mode, one should first declare the graphical terminal type; this is only done once. Exiting the menu/form driven mode will always return to the command driven mode. These two modes share the same database system. We intend to design the interface in such a way that almost every operation which can be done in the menu driven mode can be done equivalently in the command driven mode.

7.4.1.3. Mixing Problem Definition Styles

One of the most important tasks of the MIMO user interface is to define application problems such as linear subsystems, input signals or interconnected systems. As described in Section 7.2, there are three ways to define problems: a textual specification, a question and answer dialogue, and a what-you-see-is-what-you-get (WYSIWYG) menu/form editor. The mixing of these three interaction styles for problem definition is allowed in both command mode and menu/form mode. In the command mode, a user can specify the interaction style by adding the option (-f[ile] or -p[rompt]) in the command line. In the menu mode, menu options are available for the user to choose appropriate interaction styles.

7.4.1.4. Error Handling and Conflict Resolution

In all computer systems, users make errors while performing actions which are not recognized by the computer system. It is especially true when a user interface has mixing interaction styles. From the statistics of the *UNIX* command usage [Kra.1], users made errors on 10% of the commands they issued. Error rates were very uneven across commands, ranging from less than 3% to over 50%. These figures do not include the errors that users made, while the command line data is correct, however the system response doesn't match the user's intention. For example, users may print a wrong file. In menu mode, some conflicts and ambiguities may occur as well. For example, users may select a wrong *MENU* entry, or a digitized point may be outside a *MENU* item or exactly in between two such items.

An ideal computer system gives sufficient help and feedback so that once users make an error, they can easily correct it on the next attempt. In the MIMO user interface, there are several design principles for error handling:

- When errors occur, the current action is ignored, and the system responds with an error message. For some cases in menu/form mode, there are no error messages. By default, the system will ring the bell when errors occur. Users may turn off the error bell.

- Pressing an alphanumeric key has no effect when the system is waiting for pointing device input, and vice-versa.
- When using the mouse/tablet stylus to select a *MENU* entry, a digitized point may be outside a *MENU* pushbutton or exactly in between two such pushbuttons. In these cases, no error message is displayed, but the user interface will wait for the user to digitize a valid point to select a *MENU* entry.
- An *undo* facility or *redefine* capability is provided.

7.4.1.5. Mixed Interface Operations and Help Facilities

The MIMO user interface provides on-line documentation for the user. The details of its implementation and usage are described in Appendix C. This documentation facility is invoked by using the *HELP* command. Whenever a user is confused at some point in the input process corresponding to a particular position in the command tree (see below), the user can invoke the *HELP* command to obtain documentation or explanations of what the application program is expecting. We also establish a procedure for easily generating documentation.

7.4.2. Command Interface Design

The generic user interface in the *UNIX* operating system is a command-based interface which provides a successful interactive environment. This command specification approach has been recommended for applications having a large number of possible facilities and for experienced users. However, there is much more to remember and users tend to forget the commands available or their syntax. Spelling mistakes and typing errors can also lead to problems. Therefore, we try to alleviate these problems in the design of the MIMO command interface.

One of the major tasks of the MIMO user interface is to *load* the user problems into the MIMO database system easily. Since the MIMO database described in chapter 6 supports hierarchical data models, we intend to design our command

language with a similar hierarchical structure so that the user can easily browse through the database and display data of interest or make ad-hoc changes of data. A sub-tree of the MIMO command structure is shown in Fig. 7.1.

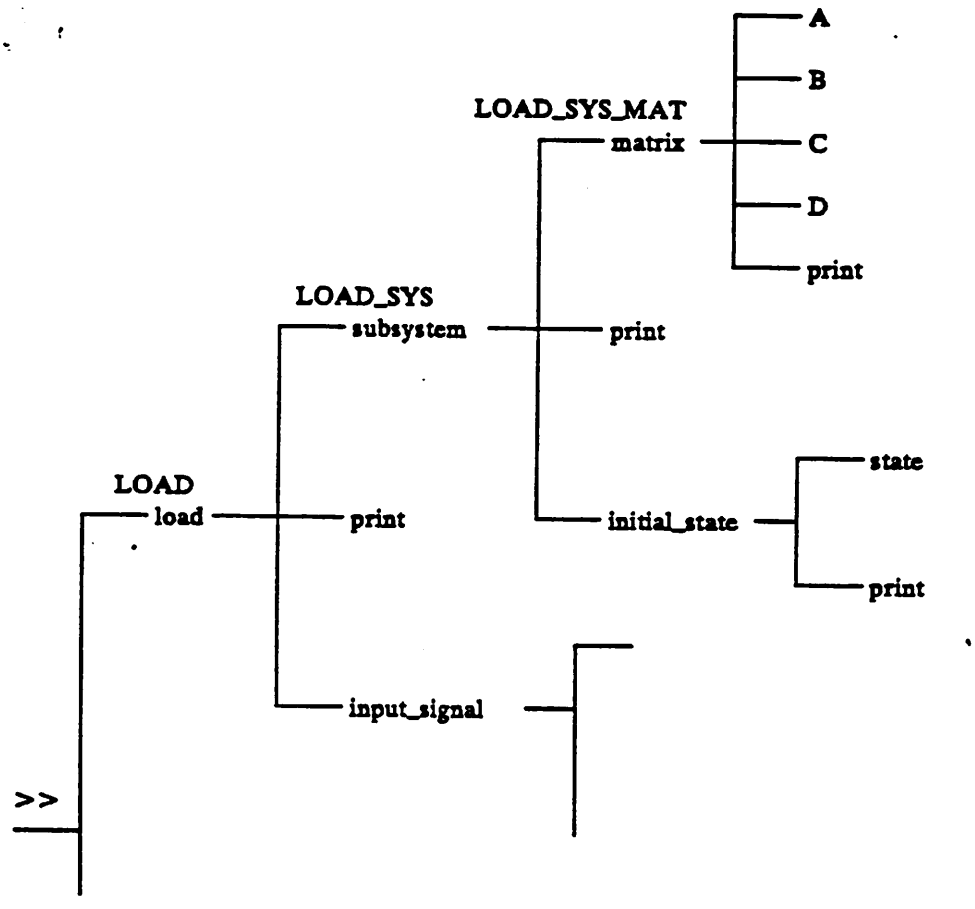


Fig. 7.1. A hierarchical structure for the command language

In summary, the command language assumes the following format:

branch_command [data] {branch_command [data]}

There are some basic design principles behind this command tree. We will discuss these principles in turn.

- *Decompose the Command Data Lines*

A typical example of a MIMO command line is shown as follows:

>> load subsystem "Plant" 3 2 2 matrix A(1,1) = x2+y**

where, according to the command tree shown in Fig.7.1, **load**, **subsystem**, **matrix**, and **A** are branch commands, and **["Plant" 3 2 2]** and **[(1,1) = x**2+y]** are the data of branch commands **subsystem** and **A** respectively. As mentioned before, spelling mistakes and typing errors often occur in the typing of long command lines. Therefore, MIMO allows the command line to be decomposed. For example, the previous command line can be decomposed as the following short commands:

load
subsystem "Plant" 3 2 2
matrix
A(1,1) = x2+y**

or

load subsystem "Plant" 3 2 2
matrix A(1,1) = x2+y**

These short commands have exactly the same effect as the long commands.

- *Mix System Prompts with Command Language*

In the MIMO system, we intend to mix system *prompts* with the command data line. The system prompts have the following format:

head-string>> **tail-string**

where **head-string** and **tail-string** are character strings containing the prompt information. At startup, the system is awaiting keyboard input, and the **head-string** and **tail-string** are null strings, therefore the system prompt is just ">>". Then, the prompt will be modified after each *branch command* is entered. The new **head-string** will be formed by appending the *command string* to the tail of the old **head-string**. The command string, denoted by **command-string**, is an abbreviated form of the

branch command, e.g., the command string of the branch command **subsystem** and **matrix** are **SYS** and **MAT** respectively. Therefore, the modification of the **head-string** can be formulated as follows:

If the old prompt string is a null string, then

head-string = command-string,

else

head-string = head-string_command-string.

The **tail-string** is either a null string or the name of the subsystem, input signal or interconnected system with which the user is currently dealing. For instance, if the commands shown in the previous example were typed in, the system prompt would undergo the following modifications:

```
>> load
LOAD>> subsystem "Plant" 3 2 2
LOAD_SYS>> matrix                                    Plant
LOAD_SYS_MAT>> A(1,1) = x**2+y                       Plant
LOAD_SYS_MAT>>                                       Plant
```

where, **LOAD, SYS** and **MAT** are the abbreviated forms of the branch commands **load, subsystem** and **matrix**. Notice that the following two commands have the same effect.

```
>> load subsystem "Plant" 3 2 2
LOAD>> subsystem "Plant" 3 2 2
```

The prompts not only improve readability, but also can save much typing efforts.

- *Locate and Present the Data by System Prompts*

Locating a specific item of data in the database and presenting it to the user are major objectives in considering the design of the MIMO user interface. Just as the *UNIX* pathname indicates the current working directory, the MIMO prompt strings, **head-string** and **tail-string**, indicate the current working location in the database tree. For example, the following prompt strings:

LOAD_SYS_MAT≫ **Plant**

show that the current working subsystem is "**Plant**", which is in the pool of MIMO subsystems. The current working location is at the matrix sub-trees of the subsystem database tree. Therefore, the user can either display or modify the system matrices information by the following commands:

LOAD_SYS_MAT≫ print A **Plant**

LOAD_SYS_MAT≫ A(1,1) = x2+y** **Plant**

- *Hierarchical Help Facility*

The prompt in the MIMO system not only shows the current working location in the database tree, but also indicates the current *node* location at the command language tree. There is only a set of **branch_command** available at each node. For instance, at the node **LOAD_SYS**, the available commands are **matrix**, **print** and **initial_state**; at the node **LOAD_SYS_MAT**, another set of commands, **A**, **B**, **C**, **D** and **print**, is available. There is much more to remember when the command tree is large. Also users tend to forget the commands available at each level and node. Therefore, the *HELP* facility mentioned in Section 7.3.5., is designed so that the user can display the set of available commands at any node of the command tree.

7.4.3. MENUS/FORMS User Interface Design

In contrast with the command user interface, the menu user interface avoids most of the drawbacks associated with the command driven environment, e.g., typing errors, syntax errors, etc. The menu interface can be used even more efficiently, if it is employed in conjunction with graphics techniques making use of window/viewport displays.

The two major tasks of the MIMO user interface are problem formulations and control of the CAD computing environment. As discussed in Section 7.2, the WYSIWYG structure editors or form-based editors are probably the best way to enter problem descriptions. By using graphic displays and a menu interface, the user may

easily browse through the database, modifying input/output through forms or structure editors. Interaction with the MIMO simulator can also be accomplished easily with the menu interface. These are the design goals for the MIMO menu interface.

The purpose of this section is to describe the screen layouts used in MIMO for the menu user interface. Then, the various form-based, structure editors are constructed in this menu driven environment. These help the user to set up a design problem easily. Finally, an interactive simulation environment is constructed so that the user can easily carry out manual tuning of his design.

7.4.3.1 MENU/FORM Layout

The MIMO system has five kinds of *MENUS*; the static menu, dynamic menu, pop-up menu, symbolic menu and color/fill pattern/line style menu. To minimize the time and effort for redrawing on some terminals, these menu viewports are arranged so that they do not overlap each other. A large, central viewport is available for form displays and application program output; the lower righthand corner is reserved for dialogue and error messages. Fig. 7.2 shows the detailed viewport layouts on the terminal screen.

We now describe the functions of each viewport in turn.

- *Static Menu Viewport*

The static menu viewport occupies the top line of the screen. The menu items shown in this viewport are usually unchanged (static) during interaction with the MIMO system. Two auxiliary items are also shown in the static menu viewport. They are the *level status* and the *quit window* of the current screen layout. The level status which is shown at the right side of the static menu viewport gives the level (name) of the current screen layout; the quit window, which is located at the left side of the static menu viewport, provides an exit for the current screen layout (similar to the MacIntosh).

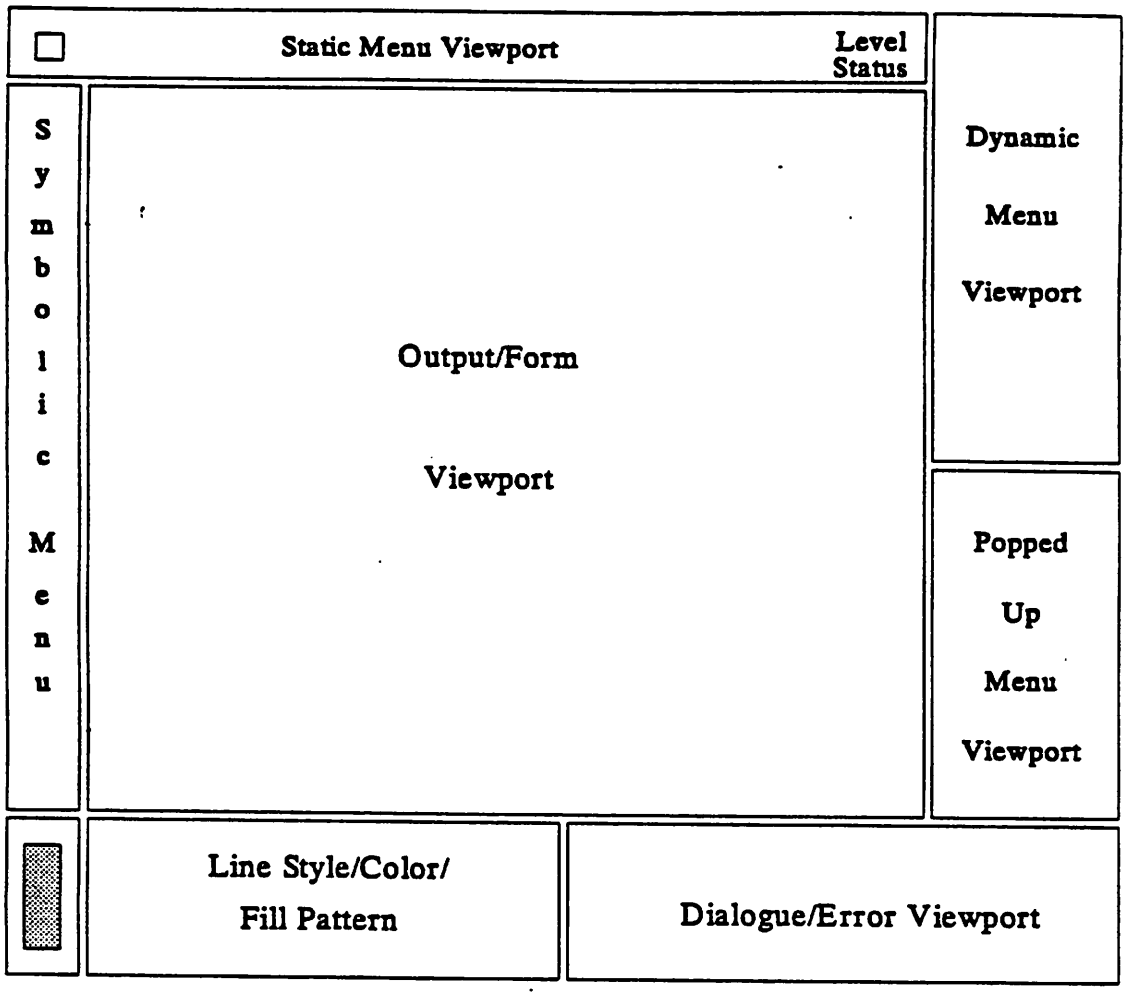


Fig. 7.2. Viewport layouts on terminal screen

- *Dynamic Menu Viewport*

The dynamic menu viewport occupies the upper righthand corner of the screen. The menu items in this viewport are changed dynamically according to the static menu item selected by the user. It is clear that the function of the dynamic menu could be implemented as a *pull-down* type of menu. However, this would require advanced features of graphics terminals that do not exist for many currently existing terminals. Therefore, in order to avoid viewport redraws, a viewport with a fixed size and location is used for the dynamic menu.

- *Pop-Up Menu Viewport*

The pop-up menu viewport occupies the middle section of the rightmost side of the screen. The menu items in this viewport are popped up, when it is necessary, according to the dynamic menu item selected by the user.

- *Symbolic Menu Viewport*

The symbolic menu viewport occupies the leftmost side of the screen. Its menu items, box, circle, line, node, etc, are shown symbolically in the viewport. This set of menus is designed for the block diagram editor which will be described in Section 7.5.2.

- *Color/Fill Pattern/Line Style Menu Viewport*

The color, fill pattern, or line style menu viewport is located at the left bottom corner of the screen. For a color graphics display, the menu items are color choices. For a non-color terminal, a set of fill patterns and line styles is shown in the viewport.

- *Dialogue/Error Viewport*

The dialogue viewport occupies the right bottom corner of the screen. Prompt streams, keyboard input data and error messages are displayed in the dialogue/error viewport.

- *Output/Form Display Viewport*

A large, central viewport is reserved for form/structure editors and application program output. There is a number of graphics editors available in the MIMO system, such as the block diagram editor, subsystem editor, input signal editor and costs/constraints editor. The graphics displays from these editors are shown in this viewport. In conjunction with a window manager, various output from the MIMO simulator can be displayed in this viewport.

7.4.3.2 Structure Editors

Along with a variety of tools for defining, editing and managing application problems, a development environment should provide a structure on these tools so that the user can easily model his application problems. In MIMO, users need to define subsystems, input signals and interconnected systems or to formulate design objectives and constraints for a given subsystem or interconnected system. By using graphics techniques and the menu interface, several structure editors with the so-called what-you-see-is-what-you-get feature are designed to achieve these purposes. We now describe these structure editors in turn.

- *Subsystem and Matrix Editor*

The subsystem editor is a form-based editor driven by the menu interface. It appears as shown in Fig. 7.3. The name and dimensions of the subsystem and the components of matrices are entered by combining menu choices and keyboard input. For creating and manipulating large matrices of complex data, a matrix editor can be invoked. The matrix editor can zoom into a submatrix of a large matrix and edit each component of the matrix with the help of the pointing device. The screen layout of the matrix editor is shown in Fig. 7.4.

- *Input Signal Editor*

The input signal editor is a form-based editor driven by the menu interface. Its screen layout is shown in Fig. 7.5. The user may define the input signal type, dimension and parameter values by menu choices and keyboard input.

- *Block Diagram Editor*

The easiest way to define an interconnected system is by drawing a block diagram. Ultimately, one desires the ability to freely draw the interconnected system block diagrams on the tablet with the stylus or mouse, using standard block diagram symbols for the block diagram elements. This can be accomplished with the block

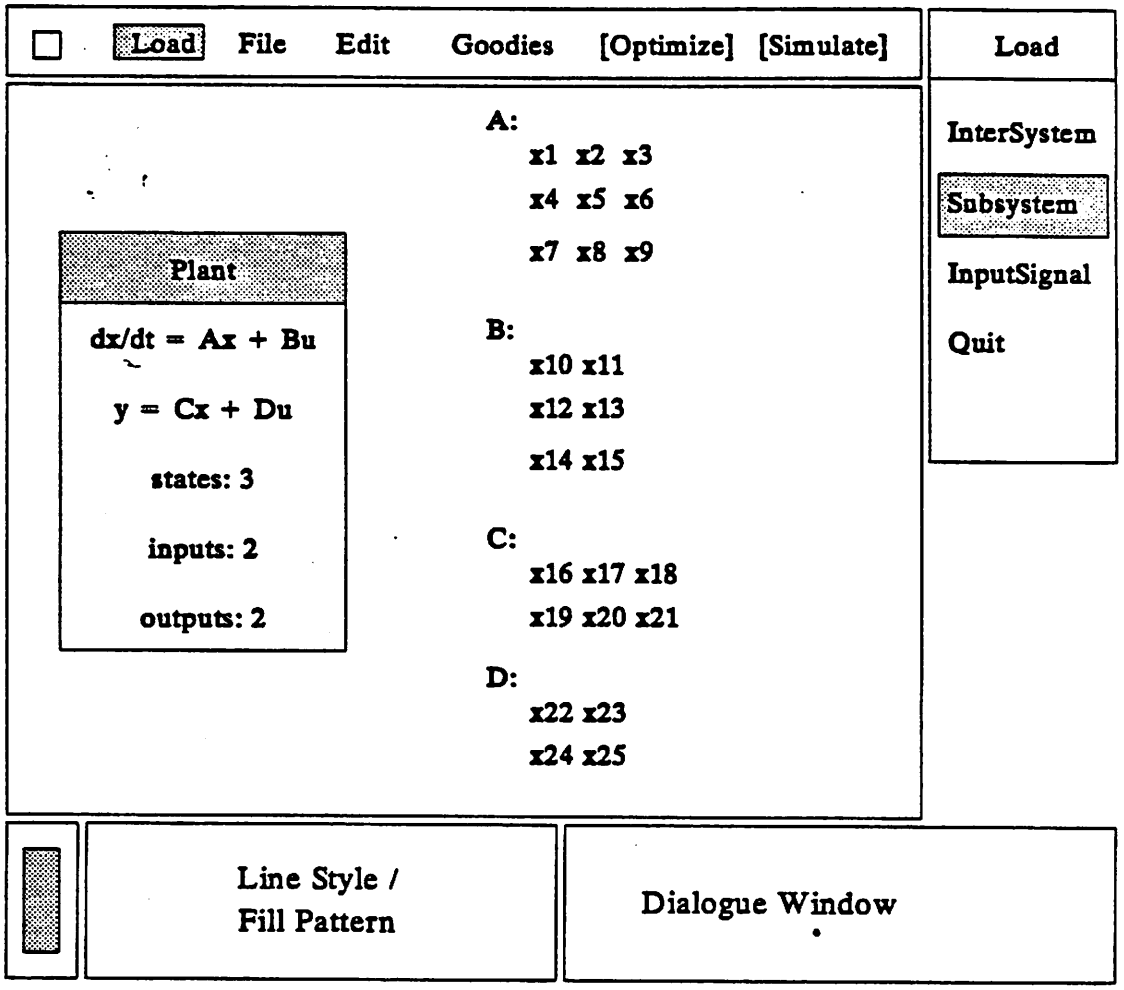


Fig. 7.3. Subsystem editor

diagram editor. A set of symbolic menu items including box, circle, node, summing node, summing sign and line, is shown on the lefthand side of the screen. The block diagram editor provides a symbol recognition facility for the symbols drawn in such diagrams, and is able to translate such drawn symbols into exact symbol drawings or into numerical parameters which can be read into the MIMO database. The editor also provides *undo* and *center* capabilities. In conjunction with other editors such as the subsystem editor and the input signal editor, the block diagram editor provides both *top-down* and *bottom-up* capabilities for the construction of the interconnected

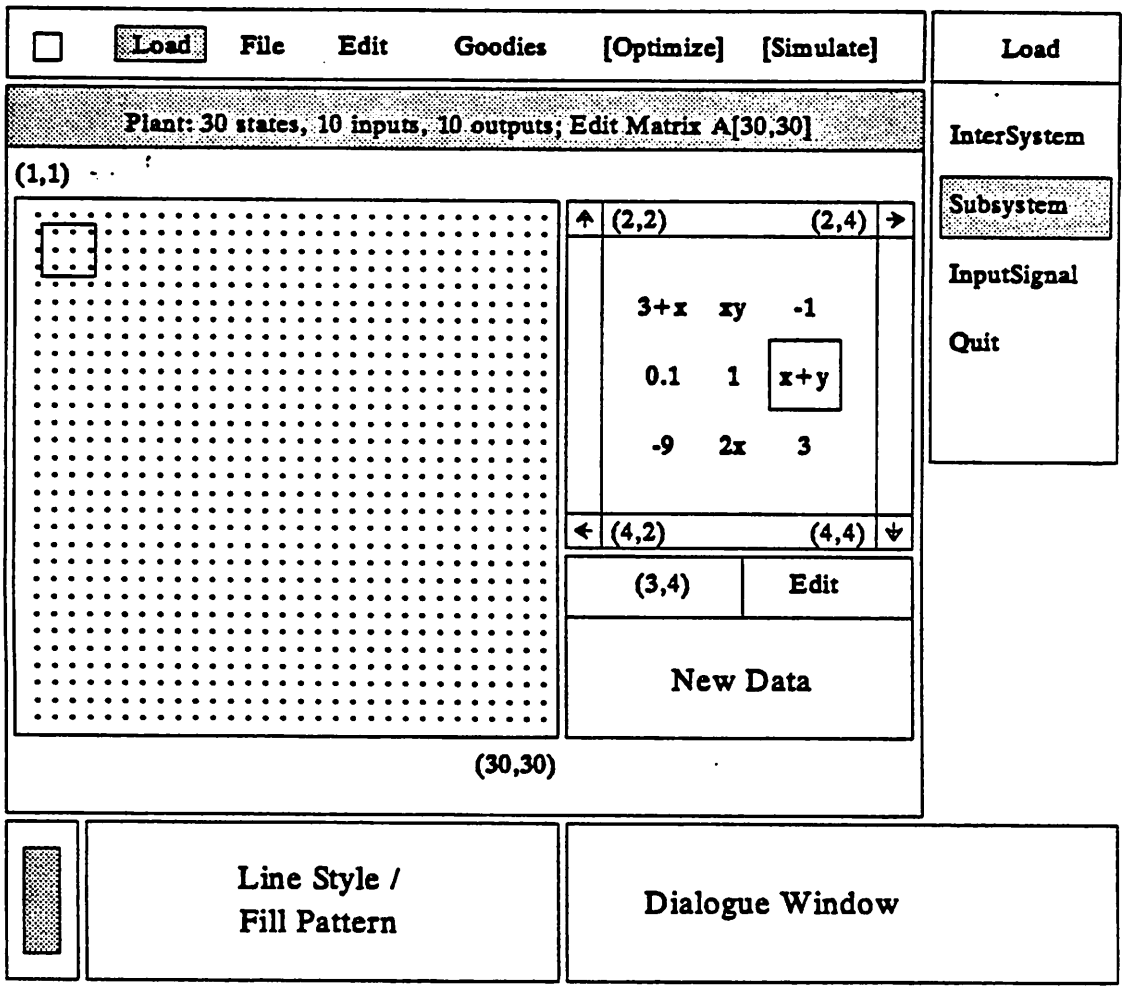


Fig. 7.4. Matrix editor

system. The screen layout for the block diagram editor is shown in Fig. 7.6.

- *Costs/Constraints Editor*

In an optimization-based design environment, a costs/constraints editor is required for defining design objectives and specifications. In Chapter 3, we have formulated various types of costs and constraints in equations (3.3.4), (3.3.7a,b), (3.3.11a,b,c) and (3.3.12). It is difficult to transcribe such mathematical formulations into a database. In MIMO, this is accomplished in two steps. First, the block

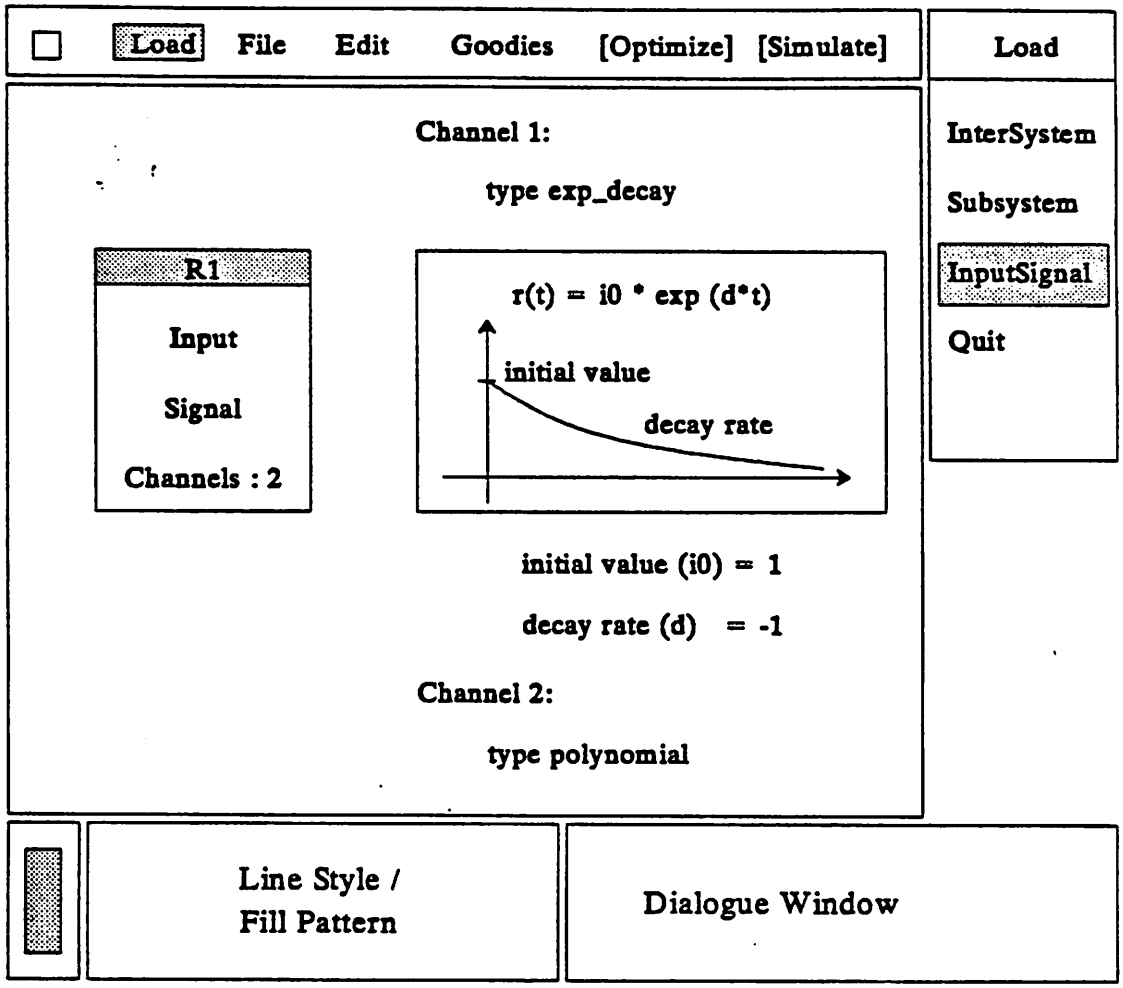


Fig. 7.5. Input signal editor

diagram of the interconnected system is zoomed out to the upper right corner of the output/form display viewport. By using the pointing device, the user can mark the location on the block diagram which specifies either a time domain response or a frequency domain response. Secondly, a form window which contains descriptions of the costs/constraints specifications is popped up. The user may use the menu interface and keyboard input to enter data for design specifications. The screen layout for the costs/constraints editor is shown as in Fig. 7.7.

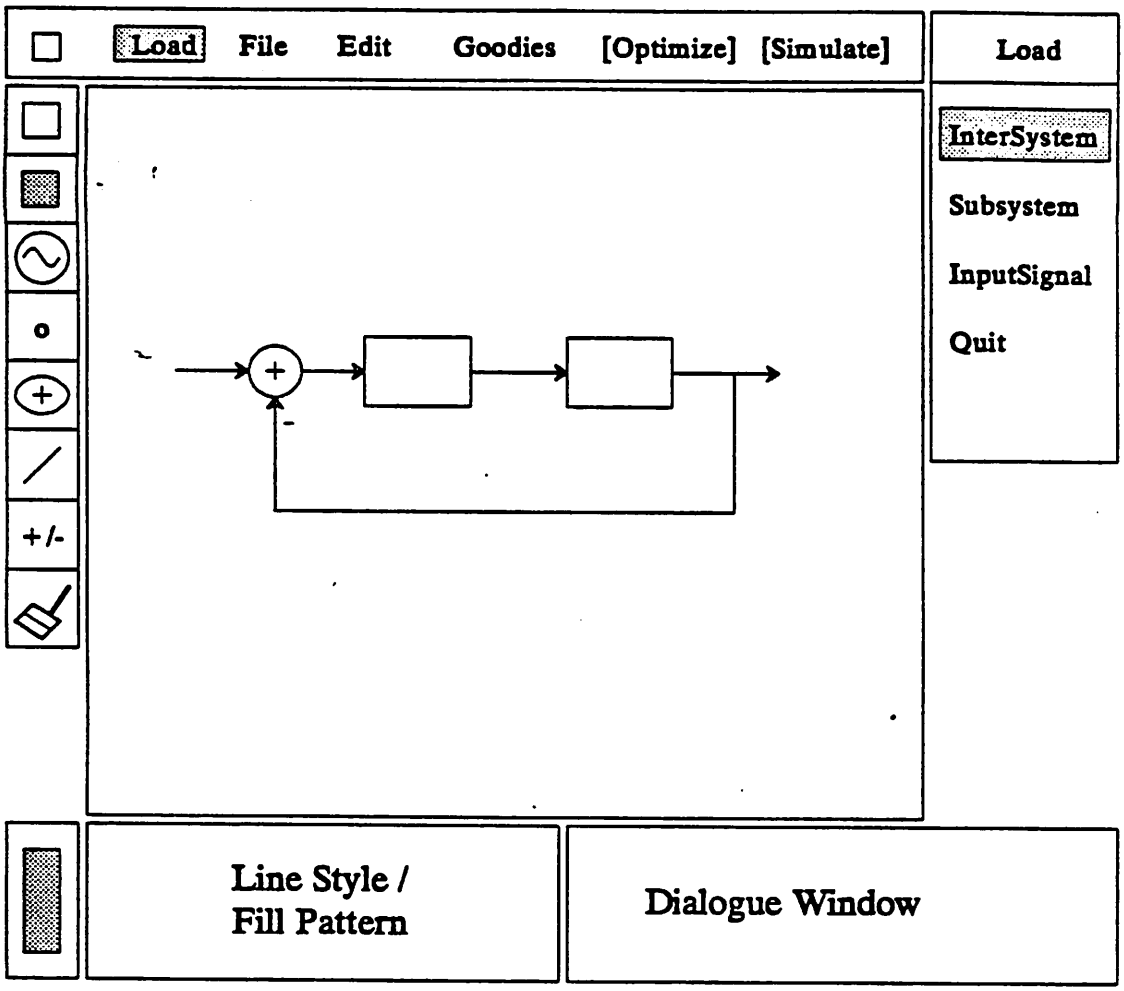


Fig. 7.6. Block diagram editor

7.4.3.3 Menu Interface for Simulation

A friendly simulation environment is specially important for initial design, analysis and verification. In the design of the menu interface for MIMO control system simulation, two important features are included to enhance the interactive capabilities of the system. First, a window/viewport manager is constructed so that the simulation outputs can be placed at any desired area in the output/form display viewport. Secondly, a manual tuning capability is included. This gives a user the power to design the system by tuning the design parameter manually. We now

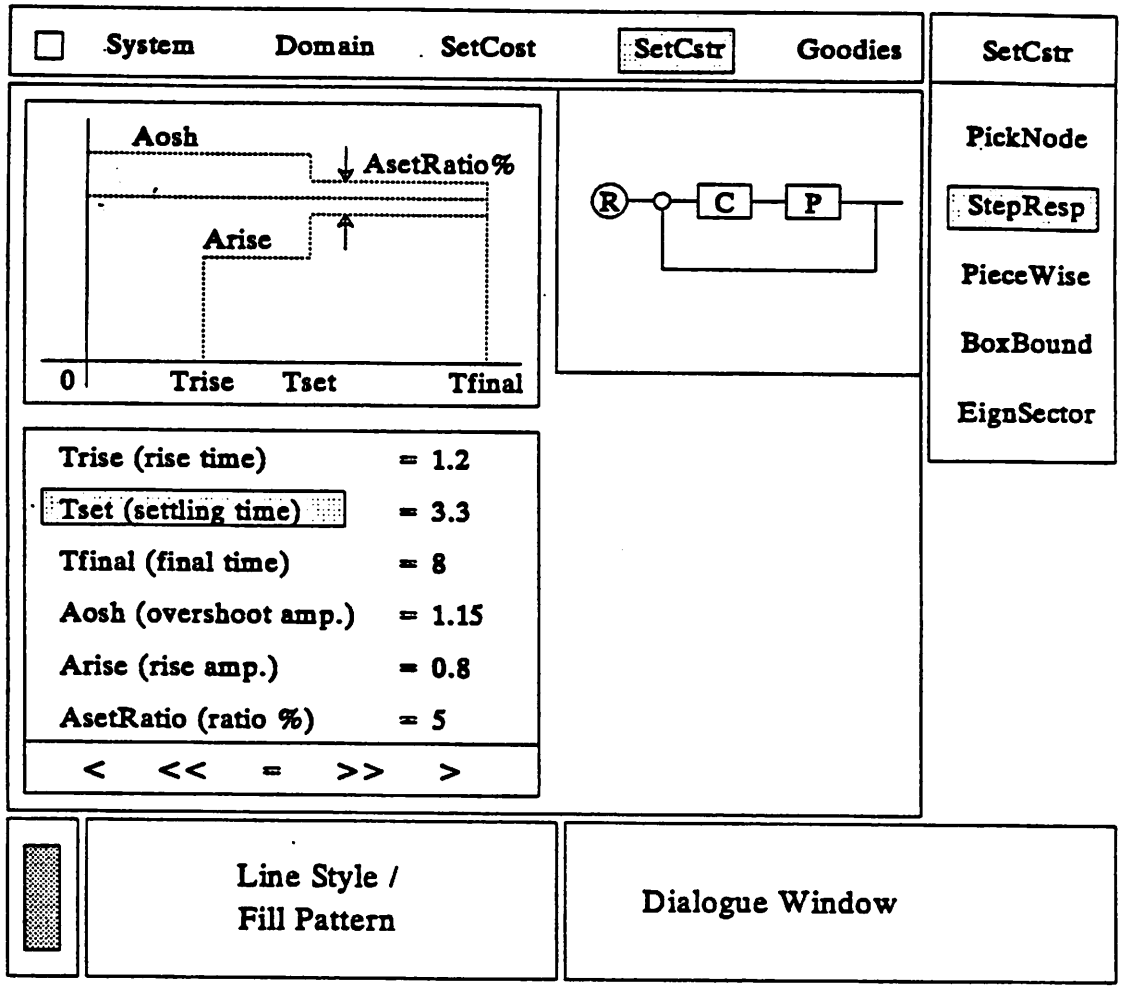


Fig. 7.7. Costs/Constraints editor

describe these features in detail.

- *Output Display (Window/Viewport) Manager*

A window is an area in a world coordinate system that contains objects to be displayed. A viewport is the area on the graphics display in which the user views the contents of a window. In other words, the window defines the objects generated from the simulator to be displayed, and a viewport defines where to display the objects.

During the simulation process, various outputs need to be displayed. The MIMO window/viewport manager allows the user to open, move, delete and redraw windows in the output/form viewport for output displays. The output display window/viewport is divided into two parts: the header of the window, and the graphics output from the simulator. The header, which occupies the top line of the window, contains the name of the interconnected system and the name of the time or frequency response being displayed in the central and lower part of the window. The layout of several windows is illustrated in the Fig. 7.8.

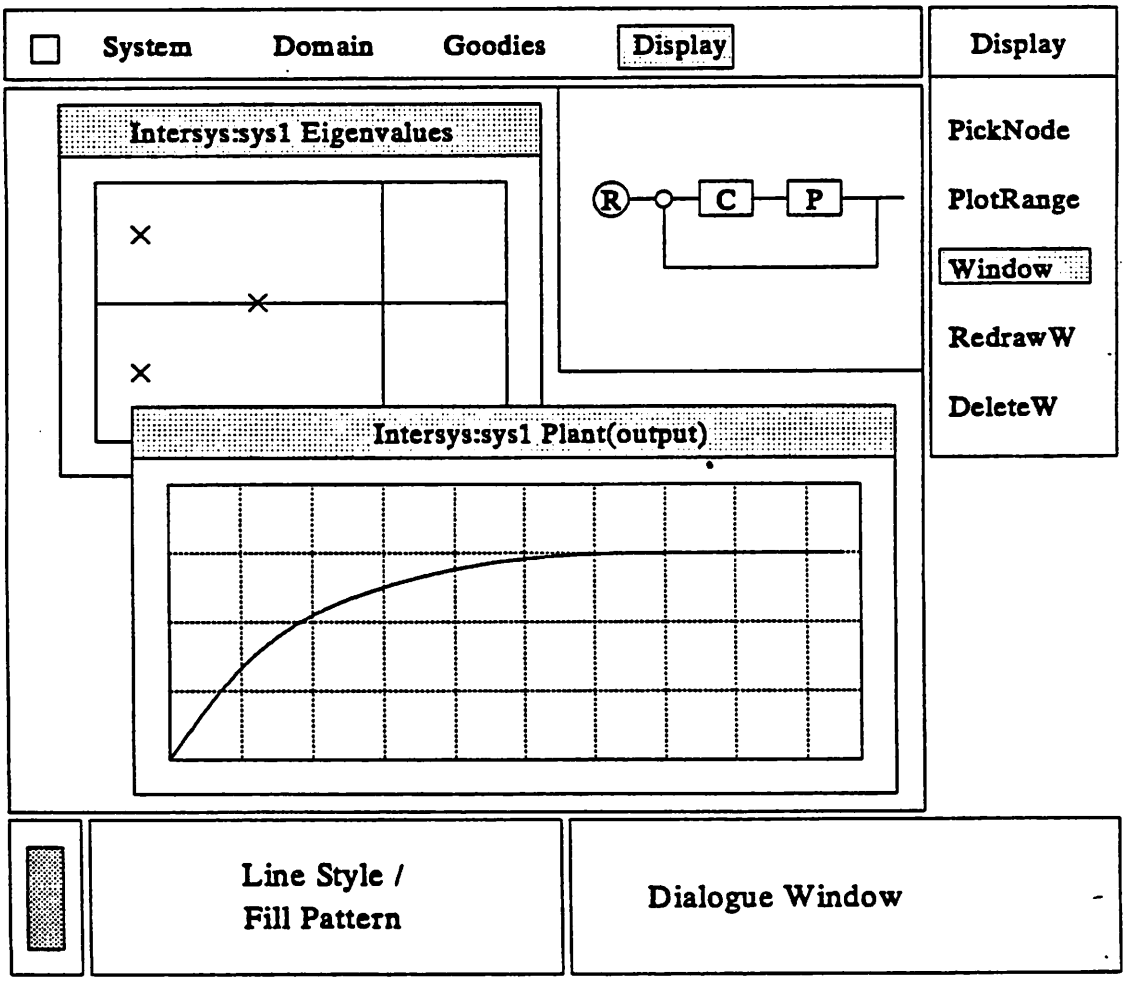


Fig. 7.8. Multi-window system for simulation output displays

- *Manual Design via Parameter Tuning*

All simulation outputs displayed in opened windows correspond to current values of the design parameters. Manual design tuning, based on sensitivity calculations, can be performed by the user. This is accomplished in three steps. First, a list of design parameters is shown in the popped-up menu viewport. Second, a set of windows can be activated for use during the parameter tuning process. Finally, the value of the design parameters can be changed (tuned) by using of a set of menu commands. Meanwhile, outputs shown in the activated windows are updated with each new set of parameter values. The layout of the screen for manual design is illustrated in the Fig. 7.9.

7.4.4. Textual Descriptions of MIMO Control Systems

There are four main reasons to have textual problem descriptions for MIMO control systems being designed. First, it saves the main storage memory. Second, it keeps the problem descriptions in files or disk so that it can be reused later. Third, it provides a way for more advanced users to bypass unnecessary and tedious use of the user interface. Fourth, the textual descriptions could possibly be created automatically by another (preprocessor) program.

A textual specification uses a language to describe objects such as subsystems, input signals, etc. The user edits the definition of a form with either a conventional text editor or the structure editors available in the MIMO system. Then, a compiler translates this specification into the run-time data representation described in Chapter 6. MIMO also provides a translator which can write the data structures into the textual problem description files.

In MIMO, there are six textual descriptions for a MIMO control system. They are the file descriptions for the linear subsystem, input signal, interconnected system, constraints, costs and design parameters. These textual specifications are described in detail in [Wuu. 1].

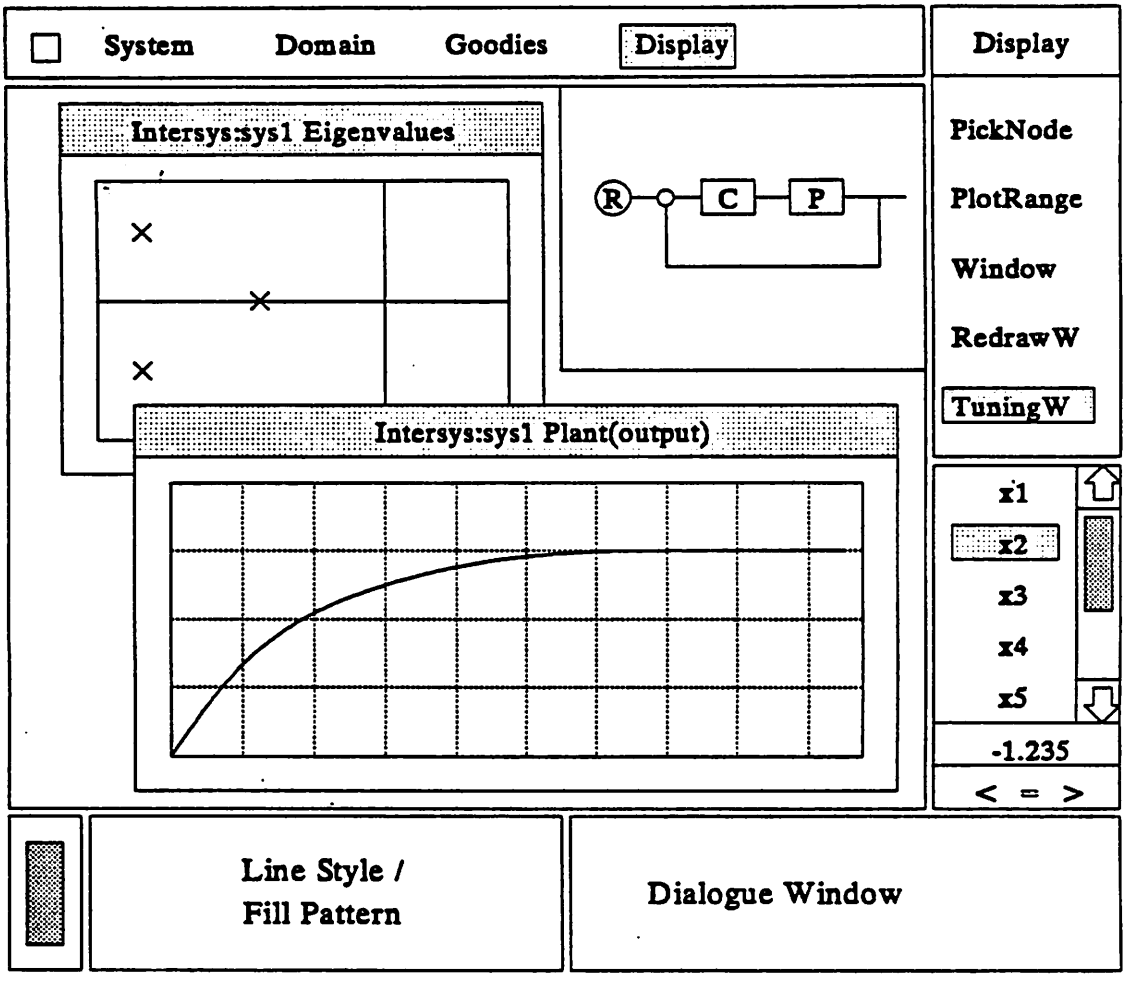


Fig. 7.9. Screen layout for manual design

7.5. Transportability of the MIMO User Interface

A final goal for the MIMO user interface is transportability. First, the *C* programs that implement the MIMO user interface should be easily transported to other computer systems besides *UNIX*. Secondly, the graphics package which we are using for the MIMO implementation is based on the *Graphical Kernel System* (GKS), see Appendix B. In June 1982, the International Organization for Standardization (ISO) announced that GKS was adopted as a Draft International Standard. This increases the transportability of the MIMO user interface. Finally, it should be reasonably easy

to add support to the MIMO system for different kinds of CAD work stations. In the current GKS implementation, described in Appendix B, we use the Model Frame Buffer (MFB) package [Bil.1] which provides a virtual graphics interface for frame buffer devices. It performs the terminal dependent task of encoding and decoding graphics code, thereby allowing the user to write graphics programs to run on almost any graphics device. A device description database, *MFBCAP*, see reference [Bil.2], is used by the MFB system, which is evolved from the concepts of a UNIX *termcap* database. Therefore, if a new display device is acquired, it can be run with the GKS routines when the appropriate device description is added to the MFBCAP database.

7.6. Concluding Remarks

A user interface has been constructed to allow both novice and experienced users to make efficient use of MIMO's facilities. Both alphanumeric and graphical/mouse inputs are available in the DELIGHT.MIMO system so that it can match the capabilities of various terminals. Implementation of the graphical part of the package is based on the Graphical Kernel System (GKS) standard which improves the portability of the system. The user interface provides both command driven and menu driven interactive environments. A graphics editor for defining linear subsystems, input signals and interconnected systems, allows both top-down and bottom-up problem formulations. To display multiple outputs from various systems, a window manager has been developed to direct the outputs to a set of user defined windows. A manual design tuning capability, based on sensitivity calculations, has been developed. A set of windows can be activated for use during the tuning process. An hierarchical on-line "Help" facility has also been included.

CHAPTER 8

DESIGN EXAMPLES

8.1. Introduction

As was mentioned in Chapter 3, the scope of applications used in this study to demonstrate the proposed methodology is limited to the design of linear time-invariant multivariable systems. Two design examples are illustrated in this chapter. The first example illustrates the use of our modified Nyquist stability criterion in control system design. The second example shows a design with time and frequency domain constraints.

8.2. Example 1: CH-47 Tandem Rotor Helicopter

Consider the design of a control system, with configuration shown in Fig. 8.1, for a CH-47 tandem rotor helicopter. The control system controls two measured outputs, the vertical velocity and pitch attitude, by manipulating collective and differential collective rotor thrust commands. An abstracted, nominal plant model, denoted by P, for the dynamics relating these variables at 40 knot airspeed is given in [Doy.1]

$$\dot{\mathbf{x}}_2 = \mathbf{A}_2\mathbf{x}_2 + \mathbf{B}_2\mathbf{u}_2, \tag{8.2.1a}$$

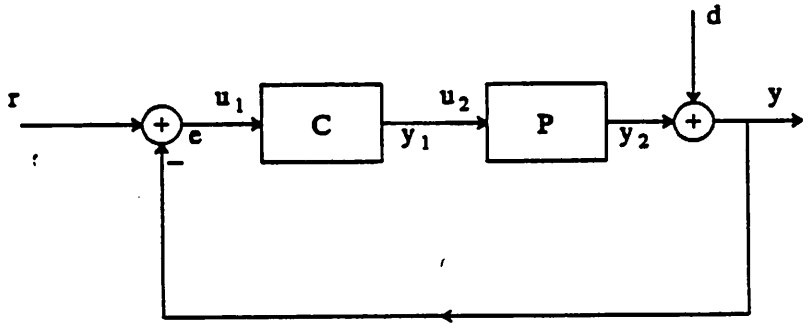
$$\mathbf{y}_2 = \mathbf{C}_2\mathbf{x}_2 + \mathbf{D}_2\mathbf{u}_2, \tag{8.2.1b}$$

where

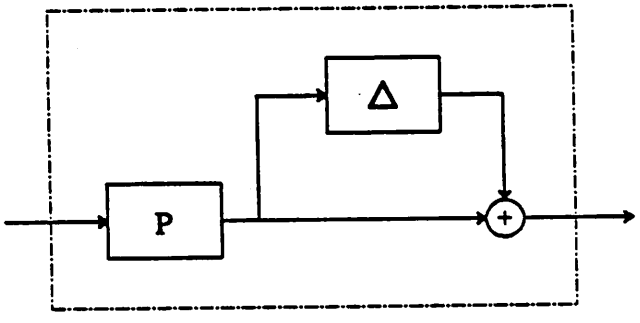
$$\mathbf{A}_2 = \begin{bmatrix} -0.02 & 0.005 & 2.4 & -32.0 \\ -0.14 & 0.44 & -1.3 & -30.0 \\ 0.0 & 0.018 & -1.6 & 1.2 \\ 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}, \qquad \mathbf{B}_2 = \begin{bmatrix} 0.14 & -0.12 \\ 0.36 & -8.6 \\ 0.35 & 0.009 \\ 0.0 & 0.0 \end{bmatrix}, \tag{8.2.1c}$$

$$\mathbf{C}_2 = \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 57.3 \end{bmatrix}, \qquad \mathbf{D}_2 = \begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}. \tag{8.2.1d}$$

The eigenvalues of the plant are -2.22787, 0.0652232, and 0.491325 $\pm$ 0.415134i.



(a)



(b)

Fig. 8.1. Control system configuration

We assume that we are required to design a compensator which stabilizes the closed loop system, reduces sensitivity to output disturbances and avoids plant saturation by disturbances. We shall given a mathematical expression for these requirements shortly. First, we chose a compensator C, with a state space representation of the form

$$\dot{\mathbf{x}}_1 = \mathbf{A}_1(\mathbf{p})\mathbf{x}_1 + \mathbf{B}_1(\mathbf{p})\mathbf{u}_1 \tag{8.2.2a}$$

$$\mathbf{y}_1 = \mathbf{C}_1(\mathbf{p})\mathbf{x}_1 + \mathbf{D}_1(\mathbf{p})\mathbf{u}_1 \tag{8.2.2b}$$

with

$$\mathbf{A}_1(\mathbf{p}) = \begin{bmatrix} \mathbf{p}^1 & \mathbf{p}^2 & \mathbf{p}^3 & \mathbf{p}^4 \\ \mathbf{p}^5 & \mathbf{p}^6 & \mathbf{p}^7 & \mathbf{p}^8 \\ \mathbf{p}^9 & \mathbf{p}^{10} & \mathbf{p}^{11} & \mathbf{p}^{12} \\ \mathbf{p}^{13} & \mathbf{p}^{14} & \mathbf{p}^{15} & \mathbf{p}^{16} \end{bmatrix}, \qquad \mathbf{B}_1(\mathbf{p}) = \begin{bmatrix} \mathbf{p}^{17} & \mathbf{p}^{18} \\ \mathbf{p}^{19} & \mathbf{p}^{20} \\ \mathbf{p}^{21} & \mathbf{p}^{22} \\ \mathbf{p}^{23} & \mathbf{p}^{24} \end{bmatrix}, \tag{8.2.2c}$$

$$\mathbf{C}_1(\mathbf{p}) = \begin{bmatrix} \mathbf{p}^{25} & \mathbf{p}^{26} & \mathbf{p}^{27} & \mathbf{p}^{28} \\ \mathbf{p}^{29} & \mathbf{p}^{30} & \mathbf{p}^{31} & \mathbf{p}^{32} \end{bmatrix}, \qquad \mathbf{D}_1(\mathbf{p}) = \begin{bmatrix} \mathbf{p}^{33} & \mathbf{p}^{34} \\ \mathbf{p}^{35} & \mathbf{p}^{36} \end{bmatrix}. \tag{8.2.2d}$$

and $\mathbf{p} = [\mathbf{p}^1, \mathbf{p}^2, \cdots, \mathbf{p}^{36}]$.

• *Closed-Loop System Stability Requirement:*

We begin with the most fundamental requirement: that of exponential stability of the nominal closed loop system. Let $\mathbf{B} = \mathring{\mathbb{C}}_+$, $\mathbf{S} = \mathring{\mathbb{C}}_-$, and let

$$D(s,\mathbf{q}) \triangleq \prod_{i=1}^{4}(s^2 + a_i s + b_i), \tag{8.2.3}$$

where $\mathbf{q} \triangleq [a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4]$. By Theorem 3.2.4, exponential stability of the *nominal* closed-looped system will be ensured if

$$-\mathbf{q}^i + \epsilon \leq 0, \qquad \forall\ i = 1, 2, \cdots, 8. \tag{8.2.4a}$$

$$-\mathrm{Re}[\chi(j\omega, \mathbf{p})/D(j\omega, \mathbf{q})] + \epsilon \leq 0, \qquad \forall\ \omega \in [0, \infty). \tag{8.2.4b}$$

where $\epsilon > 0$.

• *Stability Robustness*

Major unstructured uncertainties associated with our helicopter model are due to neglected rotor dynamics and unmodeled rate limit nonlinearities. These are discussed at length in [Ste.1]. For the purpose of our design example, it suffices to note that the modeling uncertainties are the same in both control channels and that if the actual plant model is expressed in the form $P(s)(I + \Delta(s))$, then we may assume that

$$\bar{\sigma}[\Delta(j\omega)] < b(\omega) \qquad \forall\ \omega \geq 0, \tag{8.2.5}$$

with $\bar{\sigma}$ denoting the maximum singular value of the matrix in question and $b(\omega) > 1$ for all $\omega > 10$ rad/sec. Hence (see [Doy.1]) to ensure that not only the *nominal* design, but also the worst case design is exponentially stable, we require that

$$\bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{yr}}(j\omega,\mathbf{p})] \leq 1/b(\omega) \qquad \forall\ \omega \geq 0. \tag{8.2.6}$$

- *Plant Saturation Avoidance*

To ensure avoidance of plant saturation by disturbances over the frequency range [0.01 , 100], we impose the condition:

$$\bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{u_2r}}(j\omega,\mathbf{p})] \leq 6.0,\ \forall\ \omega \in [0.01, 100]. \tag{8.2.7}$$

- *Desensitization to Output Disturbances in System Bandwidth*

We assume that the desired bandwidth of the closed loop system is [0 , 2.0]. We shall desensitize the system to output disturbances within this frequency interval by introducing an appropriate cost function. To ensure that the disturbances are not excessively amplified outside the closed loop system bandwidth, we require that

$$\bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})] \leq 2.40,\ \forall\ \omega \in [\,2.0, 1000.0\,]. \tag{8.2.8}$$

- *Objective : Desensitization to Output Disturbances*

Finally, we model the requirement of output disturbance rejection over the frequency interval [0.01 , 2.00], as a cost function $f^0 : \mathbb{R}^{36} \rightarrow \mathbb{R}$, i.e.:

$$f^0(\mathbf{p}) \triangleq \max\{\, \bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})] \mid \omega \in [\,0.01, 2.00\,] \,\}. \tag{8.2.9}$$

The initial values for the compensator were chosen arbitrarily as follows:

$$\mathbf{A}_1(\mathbf{p}) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -3 & -10 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -20 & -8 \end{bmatrix}, \qquad \mathbf{B}_1(\mathbf{p}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{8.2.10a}$$

$$\mathbf{C}_1(\mathbf{p}) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \qquad \mathbf{D}_1(\mathbf{p}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{8.2.10b}$$

The resulting closed-loop eigenvalues were -10.1505, -7.6577, -0.101678, -1.85382, –0.0943453 ± 1.16857i, and 0.3505 ± 4.52988i. The initial values of $\mathbf{q}$ for $D(s, \mathbf{q})$ were chosen so that the zeros of the polynomial $D(s, \mathbf{q})$ matched the stable closed-loop eigenvalues, above. Hence we set $\mathbf{q} = [$ 17.8282, 77.9079, 1.9755, 0.2081, 0.2087, 1.3764, 2.7724, 22.4414]. The modified Nyquist diagram for the initial values is shown in Fig.8.2.

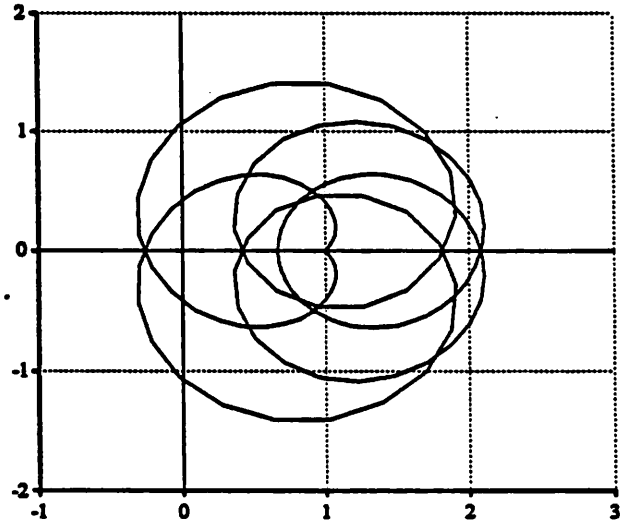


Fig. 8.2. Modified Nyquist diagram (Initial design)

Design via semi-infinite optimization must be carried out in two stages. First a stabilizing compensator must be computed, then the remainder of the optimization is carried out while maintaining closed loop stability.

After 14 iterations of the phase I - phase II, feasible directions method [Gon.1], the eight inequalities in (8.2.4a) and one functional inequality (8.2.4b), ensuring stability, were satisfied. The stabilizing compensator had the following matrices:

$$\mathbf{A}_1(\mathbf{p}) = \begin{bmatrix} -0.0597886 & 1.02787 & -0.00941522 & -0.00805203 \\ -3.00538 & -9.93141 & 0.00238179 & -0.053201 \\ -0.118403 & -0.00252507 & -0.154035 & 1.48117 \\ 0.00701371 & -0.00863074 & -20.0169 & -7.95165 \end{bmatrix}, \qquad (8.2.11a)$$

$$\mathbf{C}_1(\mathbf{p}) = \begin{bmatrix} 1.00561 & 0.00432885 & 1.00029 & -0.0159385 \\ -0.076972 & 0.984892 & -0.17239 & 0.850142 \end{bmatrix}, \tag{8.2.11b}$$

$$\mathbf{B}_1(\mathbf{p}) = \begin{bmatrix} 0.993688 & -0.100487 \\ -0.151619 & 0.996535 \\ 0.863102 & -0.206853 \\ -0.23829 & 0.980223 \end{bmatrix}, \quad \mathbf{D}_1(\mathbf{p}) = \begin{bmatrix} -0.111091 & 0.286772 \\ -1.48049 & -0.110133 \end{bmatrix} \tag{8.2.11c}$$

The resulting closed-loop eigenvalues at this point were $-13.4488 + 0i$, $-9.1549 + 0i$, $-0.0531313 + 0i$, $-0.686026 + 0i$, $-0.632693 \pm 1.83429i$, and $-3.68045 \pm 5.36757i$.

The vector $\mathbf{q}$ at iteration 14, was $\mathbf{q} = [\,17.8781,\ 77.9102,\ 3.81775,\ 3.52904,\ 3.86632,\ 4.98543,\ 6.1543,\ 29.4078\,]$. The modified Nyquist diagram for the stabilized system is shown in Fig. 8.3a. The state of the other requirements is shown in Fig. 8.3b,c,d.

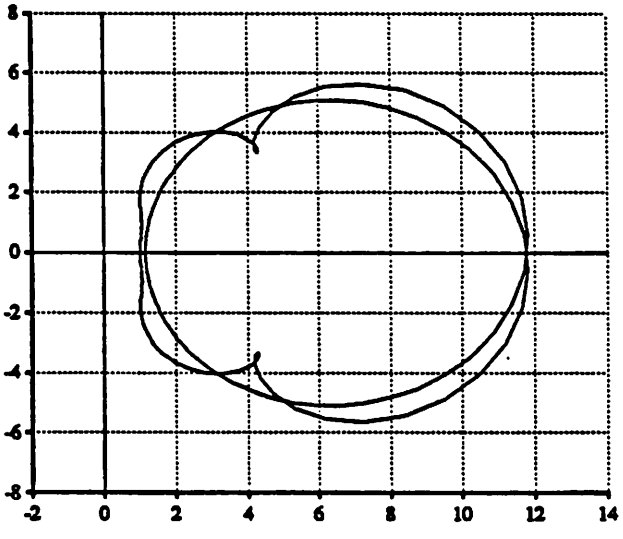


Fig. 8.3a. Modified Nyquist diagram for the stabilized system

After 63 iterations, all the design constraints were satisfied and the cost $\max_{\omega \in [0.01,2.0]} \{\, \bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})] \,\} = 0.3625$. The final compensator is described by the

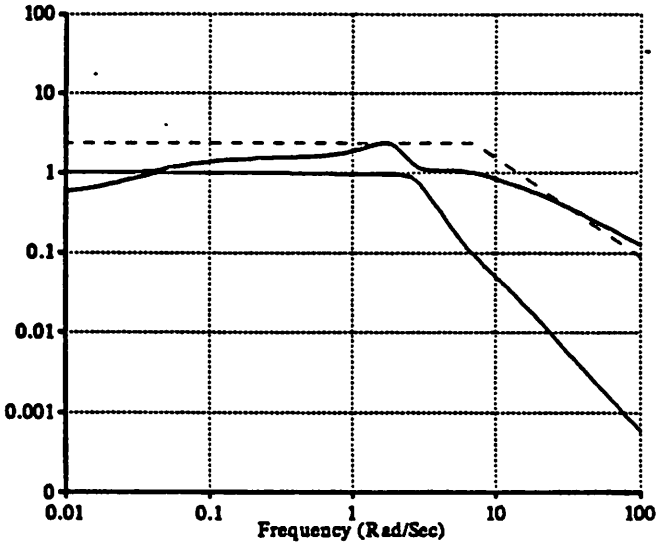


Fig. 8.3b. Singular value response for $\hat{\mathbf{H}}_{\mathbf{yr}}(j\omega,\mathbf{p})$

following matrices:

$$\mathbf{A}_1(\mathbf{p}) = \begin{bmatrix} 0.469015 & 0.858198 & 0.0985659 & -0.291412 \\ -2.95312 & -9.96053 & -0.071059 & -0.17516 \\ -0.110506 & 0.031315 & 0.0878297 & 1.20987 \\ 0.0129985 & -0.0674458 & -8.01688 & -3.74451 \end{bmatrix}, \tag{8.2.12a}$$

$$\mathbf{C}_1(\mathbf{p}) = \begin{bmatrix} 0.761554 & 0.229028 & -1.60218 & 0.213464 \\ 0.127167 & 0.654161 & -0.760495 & -0.278236 \end{bmatrix}, \tag{8.2.12b}$$

$$\mathbf{B}_1(\mathbf{p}) = \begin{bmatrix} -0.379027 & -0.319741 \\ -0.0424007 & 0.37653 \\ 0.260686 & 1.77696 \\ 0.238893 & -0.337195 \end{bmatrix}, \quad \mathbf{D}_1(\mathbf{p}) = \begin{bmatrix} -0.902073 & 2.25228 \\ -0.877849 & 0.233123 \end{bmatrix} \tag{8.2.12c}$$

The results of the final design are shown in Fig. 8.4a,b,c in solid lines, which can be compared with the results of the initial design shown in dashed lines.

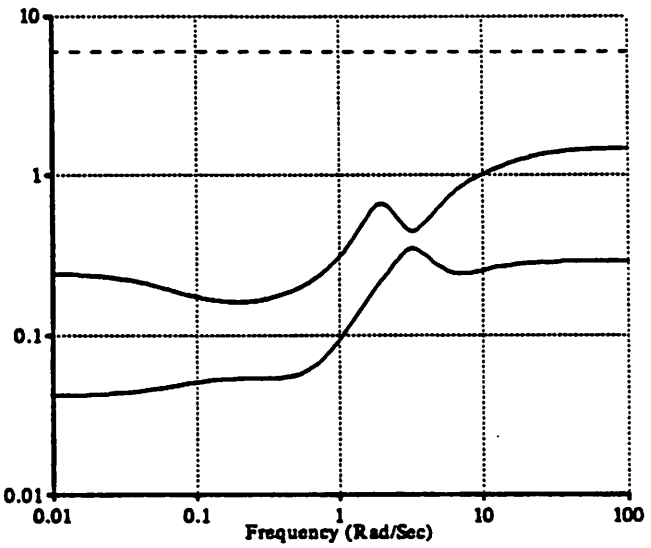


Fig. 8.3c. Singular value response for $\hat{\mathbf{H}}_{u_2 r}(j\omega,\mathbf{p})$

8.3. Example 2: A Design with Time and Frequency Domain Constraints

Consider the unity-feedback configuration shown in Fig. 8.1. We assume that the plant P is given as in (8.2.1a), (8.2.1b), with

$$\mathbf{A}_2 = \begin{bmatrix} -3 & -4 & -2 \\ 1 & 0 & 0 \\ 0 & -2 & -4 \end{bmatrix}, \qquad \mathbf{B}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix},$$

$$\mathbf{C}_2 = \begin{bmatrix} 1 & 4 & 3 \\ 0 & 2 & 3 \end{bmatrix}, \qquad \mathbf{D}_2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{8.3.1}$$

The compensator C is assumed to be the form (8.2.2a), (8.2.2b), with

$$\mathbf{A}_1(\mathbf{p}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \mathbf{B}_1(\mathbf{p}) = \begin{bmatrix} \mathbf{p}^1 & \mathbf{p}^2 \\ \mathbf{p}^3 & \mathbf{p}^4 \end{bmatrix},$$

$$\mathbf{C}_1(\mathbf{p}) = \begin{bmatrix} \mathbf{p}^5 & \mathbf{p}^6 \\ \mathbf{p}^7 & \mathbf{p}^8 \end{bmatrix}, \qquad \mathbf{D}_1(\mathbf{p}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{8.3.2}$$

where $\mathbf{p} = [\mathbf{p}^1, \mathbf{p}^2, \cdots, \mathbf{p}^8]$ is the design vector.

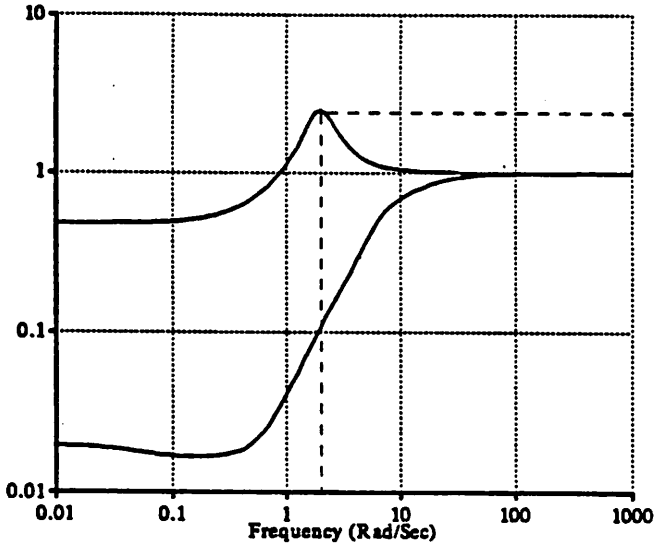


Fig. 8.3d. Singular value response for $\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})$

Our design constraints and objective are as follows:

- *Time Domain Constraints:*

The step response constraints on plant output $\mathbf{y}(t) = (\mathbf{y}^1(t),\mathbf{y}^2(t))$, corresponding to an input $\mathbf{r}(t) = (\mathbf{r}^1(t),\mathbf{r}^2(t))$, $\mathbf{r}^i(t) = 1$, for $i = 1,2$, were specified in terms of the following numbers:

rise time	1.000
settling time	3.000
final time	10.000
peak amplitude	1.090
rise amplitude	0.800
settling amplitude ratio	0.040

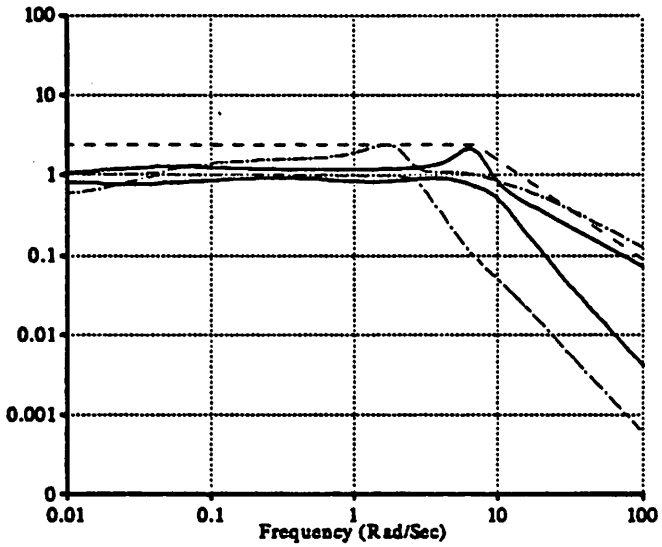


Fig. 8.4a. Singular value response for $\hat{\mathbf{H}}_{\mathbf{yr}}(j\omega,\mathbf{p})$

These constraints lead to the following pair of semi-infinite inequalities:

$$\underline{b}(t) \le \mathbf{y}^i(t,p) \le \overline{b}(t)\,,\ \forall\, t \in [\,0,16.0\,]\,,\ i = 1,2. \tag{8.3.3a}$$

where

$$\underline{b}(t) = \begin{cases} 0, & \text{for all } t \in [\,0,1.00\,] \\ 0.8, & \text{for all } t \in [\,1.00,3.0\,] \\ 0.96, & \text{for all } t \in [\,3.0,10.0\,] \end{cases} \tag{8.3.3b}$$

$$\overline{b}(t) = \begin{cases} 1.09, & \text{for all } t \in [\,0,3.0\,] \\ 1.04, & \text{for all } t \in [\,3.0,10.0\,] \end{cases} \tag{8.3.3c}$$

- *Frequency Domain Constraints:*

(1) Noninteraction constraint:

$$|\,\hat{\mathbf{H}}_{\mathbf{y}_i\mathbf{r}_j}(j\omega,\mathbf{p})\,| \le 0.15,\ \forall\, \omega \in [\,0.01,200\,]\,,\ (i,j) \in \{\,(1,2),(2,1)\,\}\,. \tag{8.3.4}$$

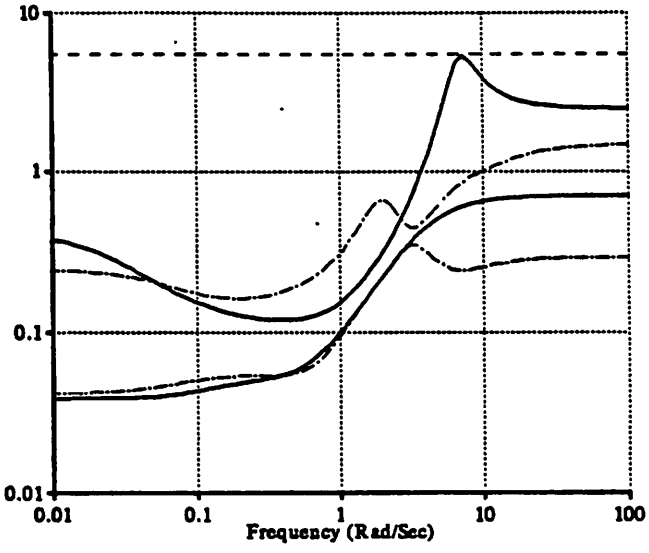


Fig. 8.4b. Singular value response for $\hat{\mathbf{H}}_{u_2r}(j\omega,\mathbf{p})$

(2) Plant saturation by disturbance avoidance constraint:

$$\bar{\sigma}[\hat{\mathbf{H}}_{u_2r}(j\omega,\mathbf{p})] \leq 5.0, \ \forall \ \omega \in [\,0.01, 200\,]. \tag{8.3.5}$$

(3) Output disturbance desensitization constraint:

$$\bar{\sigma}[\hat{\mathbf{H}}_{yd}(j\omega,\mathbf{p})] \leq 1.05, \ \forall\, \omega \in [\,1.0, 1000.0\,]. \tag{8.3.6}$$

- *Exponential Stability Constraints:*

To apply the modified Nyquist criterion, we define $\mathbf{B} = \mathring{\mathbb{C}}_+$ and let

$$D(s,\mathbf{q}) \triangleq (s + a_0)\prod_{i=1}^{2}(s^2 + a_i s + b_i), \tag{8.3.7}$$

where $\mathbf{q} \triangleq [a_0, a_1, a_2, b_1, b_2]$, and we require that

$$-\mathbf{q}^i + \epsilon \leq 0\,, \qquad \forall \ i = 1, 2, \cdots, 5\,. \tag{8.3.8a}$$

$$-\mathrm{Re}[\chi(j\omega,\mathbf{p})/D(j\omega,\mathbf{q})] + \epsilon \leq 0, \ \forall\, \omega \in [0,\infty), \tag{8.3.8b}$$

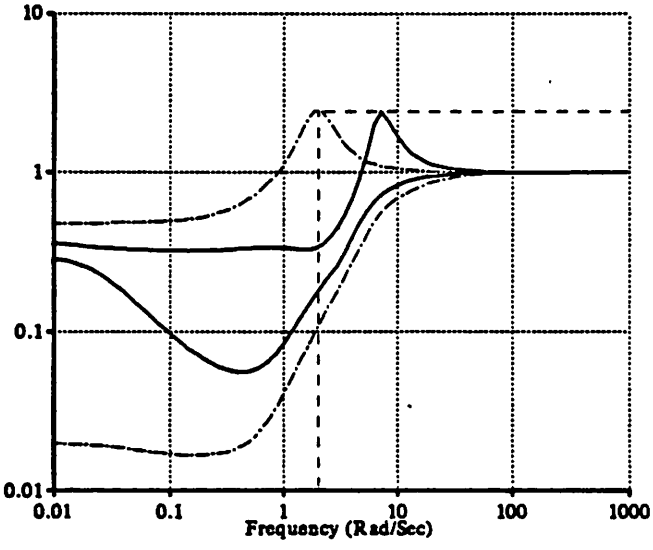


Fig. 8.4c. Singular value response for $\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})$

where $\epsilon > 0$.

- *Cost function:*

We propose to achieve good output disturbances rejection by defining it as our cost. This leads to the cost function $f^0:\mathbb{R}^8 \rightarrow \mathbb{R}$ defined by:

$$f^0(\mathbf{p}) \triangleq \max\{\, \bar{\sigma}[\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})] \mid \omega \in [\,0.001\,,1.00\,]\,\}. \tag{8.3.9}$$

First a stabilizing compensator was obtained by means of the semi-infinite optimization algorithm [Gon.1], which satisfied the stability constraints (8.3.8a,b) :

$$\mathbf{A}_1(\mathbf{p}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \mathbf{B}_1(\mathbf{p}) = \begin{bmatrix} 1.001 & -1.000 \\ -0.499 & 1.000 \end{bmatrix},$$

$$\mathbf{C}_1(\mathbf{p}) = \begin{bmatrix} 1.001 & -0.999 \\ 0.100 & 1.500 \end{bmatrix}, \qquad \mathbf{D}_1(\mathbf{p}) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{8.3.10}$$

The system responses corresponding to this compensator are shown in Fig.8.5a,b,c,d,e,f in dashed lines. We see that some of the design constraints are violated. After 12 iterations of the Polak-Wardi algorithm [Pol.2], all the constraints are satisfied. After another 59 iterations, during which the cost function $f^0(\mathbf{p})$ was minimized without constraint violation, we obtained a compensator defined by the matrices

$$\mathbf{B}_1(\mathbf{p}) = \begin{bmatrix} 9.4709 & -4.2233 \\ -2.9109 & 10.5181 \end{bmatrix}, \qquad \mathbf{C}_1(\mathbf{p}) = \begin{bmatrix} 2.60501 & -3.895 \\ 0.49817 & 1.38818 \end{bmatrix}. \quad (8.3.11)$$

the matrices $\mathbf{A}_1$ and $\mathbf{D}_1$ remained as in (8.3.10).

The final design can be evaluated by inspecting the system responses in Fig.5.5a,b,c,d,e,f in solid lines.

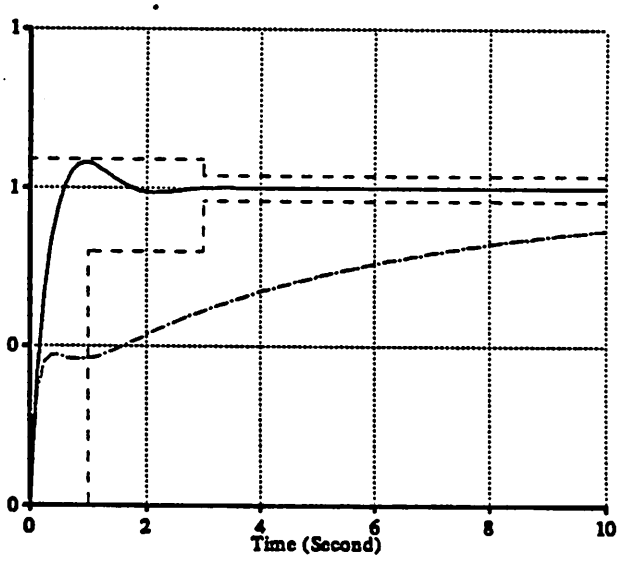


Fig. 8.5a. Step response for $\mathbf{y}^1(t,\mathbf{p})$

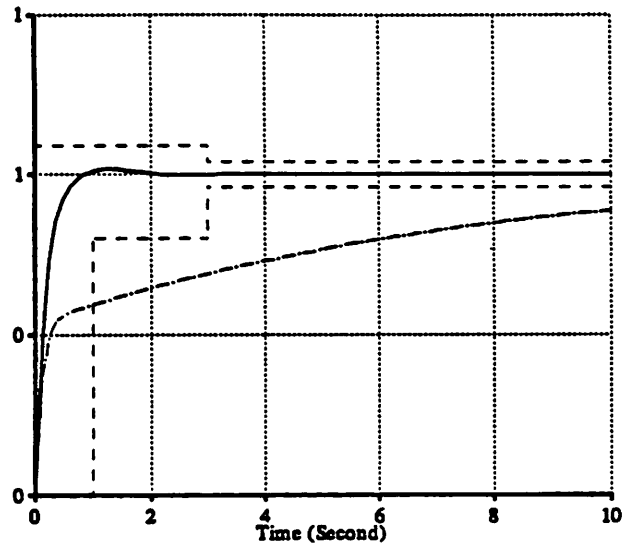


Fig. 8.5b. Step response for $\mathbf{y}^2(t,\mathbf{p})$

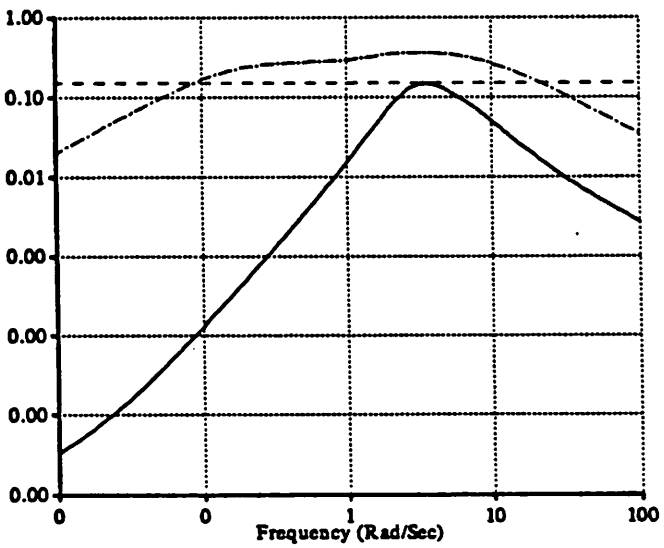


Fig. 8.5c. Absolute value response for $\hat{\mathbf{H}}_{\mathbf{y}^1\mathbf{r}^2}(j\omega,\mathbf{p})$

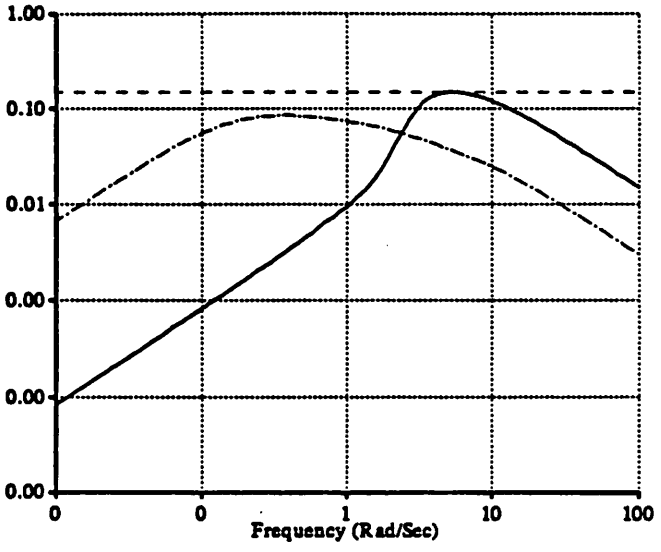


Fig. 8.5d. Absolute value response for $\hat{\mathbf{H}}_{y2_r1}(j\omega,\mathbf{p})$

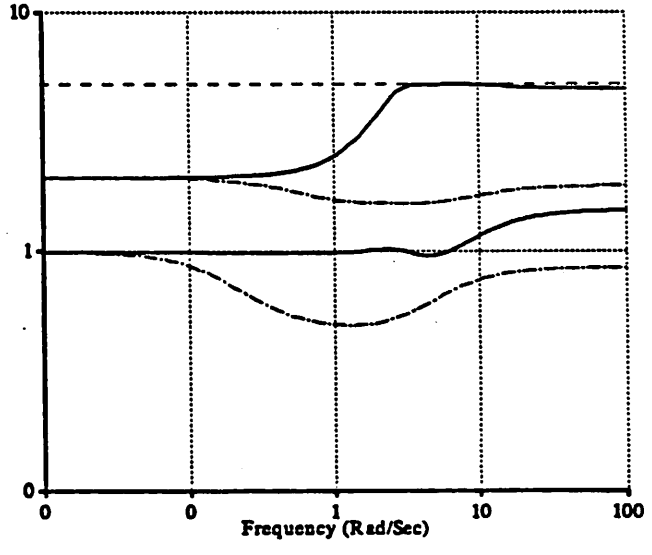


Fig. 8.5e. Singular value response for $\hat{\mathbf{H}}_{u_2r}(j\omega,\mathbf{p})$

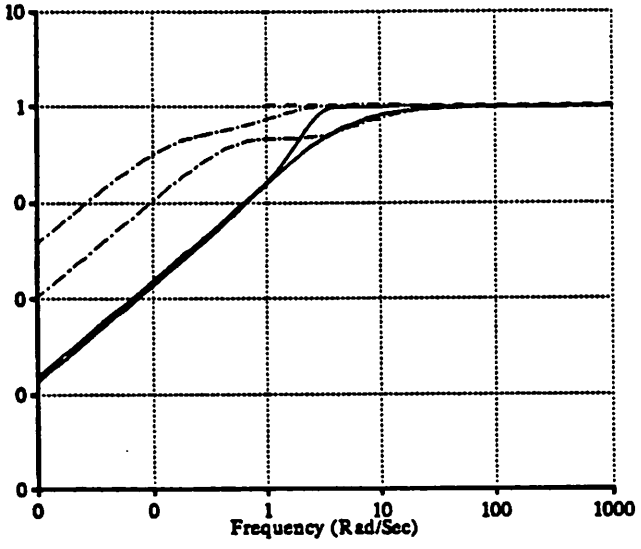


Fig. 8.5f. Singular value response for $\hat{\mathbf{H}}_{\mathbf{yd}}(j\omega,\mathbf{p})$

CHAPTER 9

CONCLUSIONS AND FUTURE RESEARCH

We have designed and implemented the optimization-based computer-aided control system design package, DELIGHT.MIMO. The DELIGHT.MIMO system provides control engineers with a powerful new design tool with which they can optimize the performance of their designs. In Chapter 3, a new computational stability criterion was presented which has proved its usefulness in the design of stabilizing compensators for linear time invariant multivariable control systems. Also, in the same chapter, various design specifications were formulated in the form of discrete or semi-infinite programming problems. In Chapter 4, an efficient simulator, MIMO, for control systems was constructed. A formula for the time domain response sensitivities was presented which eliminated the need for numerical integration and led to computational simplification. In Chapter 5, a multi-phase algorithm was presented for solving design problems sequentially. Chapter 6 described a hierarchical database which not only managed control-related data and information, but also provided a channel for communication between the user interface, the MIMO simulator, the optimization algorithm library in DELIGHT and other existing CACSD packages through an I/O handler, see Fig. 1.1. In Chapter 7, a user interface was constructed to allow both novice and experienced users to make efficient use of MIMO's facilities. Two design examples were illustrated in Chapter 8 to demonstrate the usefulness of the system in control system design.

There are still a number of enhancements or areas of future research that can further improve the DELIGHT.MIMO system. Some suggestions for enhancements and areas of future research are listed as follows:

(a) Since several shortcomings have been shown in the current algorithms, emphasis should be placed on superlinearly convergent algorithms for nondifferentiable semi-infinite optimization problems.

(b) A related enhancement is that the algorithm model in the current DELIGHT optimization library needs to be extended to or replaced by a more general

algorithm model [Pol.4] which solves nondifferentiable optimization problems.

(c) In Chapter 1, we have mentioned the difficulties of using optimization algorithms in solving engineering problems. In order to further overcome these difficulties, the user interface should be extended so that optimization algorithms can be used under the current menu/window environment.

(d) There are quite a few computer-aided design packages in the control area, see in [Jam.1]. Most of them are focused on particular aspects of design problems. Therefore, to facilitate the use of other packages, a communication means among software packages, called an *inter-package communication*, should be further studied.

REFERENCE:

[Ath.1] M. Athans, "The Role and Use of Stochastic Linear-Quadratic Gaussian Problem in Control System Design," *IEEE Trans. Automat. Contr.*, vol. AC-16, no.6, 1971.

[Aus.1] M. A. Austin, K. S. Pister, S. A. Mahin, "A Methodology for Computer-Aided Design of Earthquake-Resistant Steel Structures", *Memo No. UCB/EERC-85/13,* Earthquake Engineering Research Center, University of California Berkeley, California, December 1985

[Bal.1] R. J. Balling, K. S. Pister, E. Polak, "DELIGHT.STRUCT: A Computer-Aided Design Environment for Structural Engineering", *Memo No. UCB/EERC-81/19,* Earthquake Engineering Research Center, University of California Berkeley, California, December 1981

[Bav.1] C. A. Bavely, G. W. Stewart, "An Algorithm for Computing Reducing Subspaces by Block Diagonalization", *SIAM J. Num. Anal.*, 16, pp.359-367, 1979.

[Bec.1] R. G. Becker, "Linear System functionals via diagonalization", Imperial College, CCD report No: 79/10, Oct. 1979.

[Bil.1] G. Billingsley, K. Keller, "Model Frame Buffer (MFB) Manual", University of California, Berkeley, CAD Group, Dept. of EECS, 1982.

[Bil.2] G. Billingsley, K. Keller, "Graphics Terminal Capability Data Base (MFBCAP) Manual", University of California, Berkeley, CAD Group, Dept. of EECS, 1982.

[Bog.1] R. A. Bogen, *MACSYMA Reference Manual,* Version 6, Laboratory for Computer Science, M.I.T., Cambridge, Mass., 1977.

[Boy.1] S. Boyd, "Subharmonic Functions and Performance Bounds on Linear Time-Invariant Feedback Systems," *UCB/ERL Memo.* M84/51, June 1984.

[Bra.1] R. K. Brayton, R. Spence, *Sensitivity and Optimization,* Elsevier Scientific Publishing Company, Amsterdam, Netherlands, 1980.

[Cal.1] F. M. Callier, C. A. Desoer, *Multivariable Feedback Systems,* Springer-Verlag, 1982.

[Che.1] C. T. Chen, *Introduction to Linear System Theory,* Holt Rinehart and Winston, New York, 1970, pp. 369-371.

[Che.2] M. J. Chen and C. A. Desoer, "Necessary and Sufficient conditions for Robust Stability of Linear Distributed Feedback Systems," *Int. J. Control,* Vol.35, No.2, pp.255-267, 1982.

[Cla.1] F. H. Clarke, *Optimization and Nonsmooth Analysis,* Wiley-Interscience, New York, N.Y. 1983.

[Cru.1] J. B. Cruz, *System Sensitivity Analysis,* - Benchmark papers in Electrical Engineering and Computer Science, Dowden Hutchinson & Ross, Inc. 1971.

[Dat.1] C. J. Date, *An Introduction to Database Systems, Volume I,* Fourth Edition, Addison Wesley, 1986.

[Dav.1] E. J. Davison and I. J. Ferguson, "The Design of Controllers for the Multivariable Robust Servomechanism Problem Using Parameter Optimization Methods," Systems Control Rep. 8002, University of Toronto, Feb. 1980.

[Den.1] M. J. Denham, C. J. Benson, "SLICE: A Subroutine Library for Control System Design," Internal Report 01/82, School of EECS, Kingston Polytechnic, U.K., 1982.

[Des.1] C. A. Desoer, *Notes for A Course on Linear System.*

[Doy.1] J. C. Doyle, G. Stein, "Multivariable Feedback Design: Concepts for a Classical/Modern Synthesis", *IEEE* Trans. on Control, Vol. AC-26, No.1, pp.4-16, 1982.

[Fra.1] P. M. Frank, *Introduction to System Sensitivity Theory,* Academic Press, 1978

[Fol.1] J. D. Foley, V. L. Wallace, P. Chan, "The Human Factors of Computer Graphics Interaction Techniques", *IEEE CG&A,* pp.13-48, Nov. 1984.

[Fri.1] L. Fried, "Nine Principles for Ergonomic Software," *Datamation* Vol.28, No.12, pp.163-166, Nov. 1982.

[Gan.1] F. R. Gantmacher, *The Theory of Matrices,* Vol I, II, Chelsea Publ. Co., New York, 1959.

[Gon.1] C. Gonzaga, E. Polak, and R. Trahan, "An Improved Algorithm for Optimization Problems with Functional Inequality Constraints," *IEEE Trans. Automat. Contr.,* vol.AC-25, no.1, 1980.

[Gon.2] C. Gonzaga and E. Polak, "On Constraint Dropping Schemes and Optimality Functions for a Class of Outer Approximations Algorithms," *J. SIAM Contr. Optimiz.*, vol. 17, 1979.

[Hal.1] J. K. Hale, *Ordinary Differential Equations* Robert E. Krieger Publishing Company, Inc., 1980.

[Hea.1] A. Hearn, *REDUCE 2 User's Manual,* University of Utah, Salt Lake City, 1973.

[Hof.1] K. Hoffman, R. Kunze, *Linear Algebra,* Prentice Hall, 1971.

[Hop.1] F. R. A. Hopgood, D. A. Duce, J. R. Gallop, D. C. Sutcliffe, *Introduction to the Graphical Kernel System (GKS),* Computer Science Press, Inc. 1981.

[Hor.1] E. Horowitz, S. Sahni, *Fundamentals of Data Structures,* Academic Press, 1983.

[Hor.2] I. M. Horowitz, *Synthesis of Feedback Systems,* New York: Academic Press, 1963.

[Hsu.1] C. H. Hsu, C. T. Chen, "A Proof of the Stability of Multivariable Feedback Systems", *Proceedings of the IEEE,* pp. 2061-2062, Nov. 1968

[Jam.1] M. Jamshidi and C. J. Herget, *Computer-Aided Control Systems Engineering,* edited by M. Jamshidi and C.J.Herget, North-Holland, 1985.

[Joy.1] William Joy, "An Introduction to Display Editing with Vi", *Unix Reference Manual*

[Kar.1] J. S. Karmarkar and D. D. Siljak, "Maximization of Absolute Stability Regions by Mathematical Programming Methods," *Regelungtechnik,* no.2, 1975.

[Kat.1] T. Kato, *Perturbation Theory of Matries,* J. Assoc. Comp. Mach., 1958.

[Kat.2] T. Kato, *A Short Introduction to Perturbation Theory for Linear Operators,* Springer-Verlag, 1982.

[Ker.1] B. W. Kernighan, D. M. Ritchie, *The C Programming Language,* Prentice-Hall, Inc., 1978.

[Kok.1] P. V. Kokotovic, R. S. Rutman, "Sensitivity of Automatic Control Systems (Survey)," *Auto. Remote Cont.,* No.4, p727-749, 1965.

[Kok.2] P. V. Kokotovic, "Method of Sensitivity Points in the Investigation and Optimization of Linear Control Systems," *Auto. Remote Cont.,* No.12, p1512-1517, 1964.

[Kra.1] R. E. Kraut, S. J. Hanson, J. M. Farber, "Command Use and Interface Design," *CHI'83 Proceeding,* pp.120-124, Dec., 1983.

[Kwa.1] H. Kwakernaak and R. Sivan, *Linear Optimal Control Systems,* New York: Wiley-Interscience, 1972.

[Lan.1] P. Lancaster, "On Eigenvalues of Matries Dependent on a Parameter", *Numerische Mathematik,* Vol.6, pp. 377-387, Dec. 1964.

[Lau.1] A. J. Laub, "Efficient Multivariable Frequency Response Computations", *IEEE* AC-26, p407, April, 1981.

[Lau.2] A. J. Laub, "Numerical Linear Algebra Aspects of Design Computations", *IEEE* AC-30, p97-108, Feb., 1985.

[Lau.3] A. J. Laub, M. T. Heath, "Core Numerical Library for CASCADE", in *Issues in The Design of A Computer-Aided Systems and Control Analysis and Design Environment (CASCADE),* ORNL/TM9038, 1984.

[Lue.1] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming,* Reading, MA: Addison Wesley, 1975.

[Mar.1] J.E. Marsden, *Elementary Classical Analysis,* Freeman, 1974.

[Mar.2] J. E. Marsden, *Basic Complex Analysis,* W. H. Freeman & Company, 1973.

[Mol.1] C. Moler, C. Van Loan, "Nineteen Dubious Ways to Compute the Exponential of a Matrix", *SIAM Review,* p801-836, Oct. 1978.

[Mon.1] A. Monk, "Fundamentals of Human-Computer Interaction," *Academic Press,* 1984.

[Nob.1] B. Noble, J. W. Daniel, *Applied Linear Algebra,* Prentice-Hall, 1977

[Nye.1] W. T. Nye, A. L. Tits, "An Application-Oriented, Optimization-Based Methodology for Interactive Design of Engineering Systems", to appear in

International Journal of Control, 1986

[Nye.2] W. T. Nye, "The Helper Facility", manuscript, 1984.

[Nye.2] W. T. Nye, "DELIGHT: An Interactive System for Optimization-Based *Ph.D. Dissertation, U. C. Berkeley,* Engineering Design" June, 1983.

[Nyq.1] H. Nyquist, "Regeneration Theory", *Bell Syst. Tech. J.* vol. 2, pp. 126-147, Jan. 1932.

[Oet.1] W. Oettli, "The Method of Feasible Directions for Continuous Minimax Problems," *Proc. 9the Intl. Math. Prog. Symp.* Budapest, pp. 505, 1976.

[Par.1] B. N. Parlett, K. C. Ng, "Development of An Accurate Algorithm for EXP(BT)", *Center for Pure and Applied Mathematics,* PAM-294, University of California, Berkeley.

[Par.2] B. N. Parlett, "A Recurrence Among the Elements of Functions of Triangular Matrices", *Linear Algebra Appl.,* 14, pp.117-121, 1976.

[Pet.1] P. H. Petkov, N. D. Christov, M. M. Konstantinov, "A Program Package for Computer-Aided Design of Digital Computer Control Systems," Preprint SOCOCO82, pp. 217-220, Madrid, Spain, 1982.

[Pol.1] E. Polak, D. Q. Mayne, D. M. Stimler, "Control System Design via Semi-infinite Optimization: A Review", *UCB/ERL,* M84/35, April 1984.

[Pol.2] E. Polak, Y. Wardi, "A Nondifferentiable Optimization Algorithm for the Design of Control Systems subject to Singular Value Inequalities over a Frequency Range", *Automatica,* Vol. 18, No. 3, pp. 267-283, 1982.

[Pol.3] E. Polak, "Notes on the Mathematical Foundations of Nondifferentiable Optimization in Engineering Design", *UCB/ERL* M84/15, 2 Feb. 1984.

[Pol.4] E. Polak, "On the Mathematical Foundations of Nondifferentiable Optimization in Engineering Design", *UCB/ERL,* Memo 85/17, Feb. 1985.

[Pol.5] E. Polak, R. Trahan, D.Q. Mayne, "Combined Phase I-Phase II Methods of Feasible Direction", *Math. Prog.* Vol.17, pp.61-73, 1979.

[Pol.6] E. Polak, "A Modified Nyquist Stability Test for Use in Computer Aided Design", *IEEE* AC-29, No.1, pp.91-93, Jan. 1984.

[Pol.7] E. Polak and R. Trahan, "An Algorithm for Computer Aided Design of Control Systems", *Proc. IEEE Conf. on Dec. and Control,* 1976.

[Pol.8] E. Polak, "Algorithms for a Class of Computer-Aided Design Problems: A Review", *Automatica,* Vol.15, pp.795-813, September 1979.

[Pol.9] E. Polak and D. M. Stimler, "On the Design of Linear Control Systems with Plant Uncertainty via Nondifferentiable Optimization ", *IFAC 9-th World Congress,* Budapest, July 1984.

[Pol.10] E. Polak, D. Q. Mayne and D. M. Stimler, "Control System Design via Semi-Infinite Optimization: A Review", *Proceedings of the IEEE,* pp 1777-1795, December 1984.

[Pol.11] E. Polak, "Semi-Infinite Optimization in Engineering Design", in *Lecture Notes in Economics and Mathematical Systems,* Vol. 215, *Semi-Infinite Programming and Applications,* A. V. Fiacco and K. O. Kortanek, Eds. New York, Springer-Verlag, 1983.

[Pol.12] E. Polak, D. Q. Mayne, "An Algorithm for Optimization Problems with Functional Inequality Constraints," *IEEE Trans. Automat. Contr.,* vol. AC-21, no.2, 1976.

[Pol.13] E. Polak, "On the Use of Optimization in the Design of Linear Systems," *UCB/ERL* Memo M377, 1973.

[Pol.14] E. Polak, *Computational Methods in Optimization: A Unified Approach.* New York: Academic Press, 1971.

[Row.1] L. A. Rowe, "Tools for Developing OLTP Applications", *Datamation,* Vol.31, No.15, pp.73-82, Aug., 1985.

[Row.2] L. A. Rowe, K. A. Shoens,"FADS - A Form Application Development System", *ACM SGMOD Int. Conf. on Mgt of Data,* June 1982.

[Row.1] L. A. Rowe, "Fill-in-the-Form Programming", *VLDM,* 1985.

[Sal.1] Septimiu E. Salcudean, "Algorithms for Optimal Design of Feedback Compensators", Ph.D. Dissertation, Dec. 1986.

[San.1] N. R. Sandel, "Robust Stability of Systems with Applications to Singular Value Perturbations," *Automatica,* Vol.15, 1979.f

[Smi.1] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, C. B. Moler, *Matrix Eigensystem Routines - EISPACK Guide,* Springer-Verlag, 1976.

[Ste.1] G. Stein, J.C. Doyle, "Singular values and feedback: Design Examples," *Proc. Allerton Conf.,* Urbana, IL, 1978.

[Tra.1] R. Trahan, E. Polak, "A Derivative Free Algorithm for a Class of Infinitely Constrained Optimization Problems," *IEEE Trans. on Automat. Contr.,* Vol. AC-25, No.1, 1979.

[Wra.1] A. Wragg, C. Davies, "Computation of the Exponential of a Matrix, Part I: Theoretical Considerations", *JIMA,* 11, pp.369-375, 1973; and "Computation of the Exponential of a Matrix, Part II: Practical Considerations", *JIMA,* 15, pp.273-278, 1975.

[Wuu.1] T. S. Wuu, "DELIGHT.MIMO Manual", in preparation.

[Zak.1] V. Zakian and L. Al-Naib, "Design of Dynamical and Control Systems by the Method of Inequalities," *Proc. Inst. Elec. Eng.,* vol.120, no.11, 1973.

[Zou.1] G. Zoutendijk, *Methods of Feasible Directions,* Elsevier, Amsterdam, 1970.

APPENDIX A

SYMBOLIC DIFFERENTIATOR

A.1. Introduction

In numerical mathematics there are many instances where it is necessary to differentiate a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ one or more times. If a package that performs analytical differentiation is not available, the user must either carry out the differentiation by hand, a tedious and error-prone process, or compute a numerical approximation to the derivatives. In many instances both alternatives are unacceptable.

Several packages for performing symbolic computation are currently available, MACSYMA [Bog.1] and REDUCE [Hea.1] are among the best known. However, these packages are to a large degree *isolated* in the sense that they do not readily interface with programs written in a language of the user's choice.

In the current application, the functions which are of interest in the MIMO system are multivariable polynomials (multinomials) and rational functions. The data structures of multinomials and rational functions were described in Chapter 6. This Appendix describes a simple library of procedures which perform analytic differentiation based on those data structures.

A.2. Procedures for Symbolic Differentiation

The procedures that perform symbolic differentiation of multinomials and rational functions are described in this section. The procedure headings are given for a C language implementation. The appropriate type declarations for any variables and procedures are described in Chapter 6.

- **Multivariable Polynomials:**

The procedure **exprtopolyn**, which has a heading of the form

```
POLYNOMIAL *exprtopolyn(expr, ierr)      /* expression to polynomial */
char *expr;
int *ierr;
```

takes a string representation of a multinomial as input. A pointer to the data structure representation of the multinomial is returned. If there is a syntax error in the multinomial string, then **ierr** is set to **YES**.

Once the data structure representation of a multinomial has been created, it may be evaluated by the procedure **evalpolyn**, which has a heading of the form

```
double evalpolyn(polyn)                  /* evaluate polynomial */
POLYNOMIAL *polyn;
```

This procedure takes as input a pointer to a multinomial and returns the value of the multinomial.

The procedure **polyntoexpr**, which has a heading of the form

```
char *polyntoexpr(polyn)                 /* polynomial to expression */
POLYNOMIAL *polyn;
```

reverses the process of **exprtopolyn** and converts a multinomial data structure into a character string ready for printing.

The procedure **polynderv**, which has a heading of the form

```
POLYNOMIAL *polynderv(polyn, param)      /* derivative of polynomial */
POLYNOMIAL *polyn;
PARAMETER *param;
```

takes a pointer to a multinomial data structure and a pointer to a parameter. It creates a multinomial data structure of the derivative of the input multinomial with respect to the parameter and returns a pointer to it.

The procedure **polyoppoly**, which has a heading of the form

POLYNOMIAL *polyoppoly(apolyn, bpolyn, op)
*POLYNOMIAL *apolyn, *bpolyn;*
char op;

creates a pointer to a multinomial data structure which is the result of two multinomials, *apolyn* and *bpolyn*, combined by the given binary operator. There are three operators, +, -, and *, which are allowed for multinomial operations.

- **Rational Functions:**

The procedure **exprtoratfn**, which has a heading of the form

RATIONALFUNCTION *exprtoratfn(expr, ierr)
*char *expr;*
*int *ierr;*

takes a string representation of a rational function as input. A pointer to the data structure representation of the rational function is returned. If there is a syntax error in the rational function string, then **ierr** is set to **YES**.

Once the data structure representation of a rational function has been created, it may be evaluated by the procedure **evalratfn**, which has a heading of the form

double evalratfn(ratfn)
*RATIONALFUNCTION *ratfn;*

This procedure takes as input a pointer to a rational function and returns the value of the rational function.

The procedure **ratfntoexpr**, which has a heading of the form

char *ratfntoexpr(ratfn)
*RATIONALFUNCTION *ratfn;*

reverses the process of **exprtoratfn** and converts a rational function data structure into a character string ready for printing.

The procedure **ratfnderv**, which has a heading of the form

RATIONALFUNCTION *ratfnderv(ratfn, param)
*RATIONALFUNCTION *ratfn;*
*PARAMETER *param;*

takes a pointer to a rational function data structure and a pointer to a parameter. It creates a rational function data structure of the derivative of the input polynomial with respect to the parameter and returns a pointer to it.

The procedure **ratfopratf**, which has a heading of the form

RATIONALFUNCTION *ratfopratf(aratfn, bratfn, op)
*RATIONALFUNCTION *aratfn, *bratfn;*
char op;

creates a pointer to a rational function data structure which is the result of two rational functions, *aratfn* and *bratfn*, combined by the given binary operator. There are four operators, +, -, *, and /, which are allowed for rational function operations.

APPENDIX B

GKS IMPLEMENTATION

B.1. GKS - A Device Independent Graphics Package

one of the most important objectives in constructing a graphical package is to set up a standard so that the production and manipulation of pictures does not depend on the computer or graphics device being used. In this section, an international graphics standard known as the *Graphical Kernel System* (GKS) is introduced. The implementation of the GKS package is discussed.

B.1.1. Introduction and Overview

Standardization in computer graphics has a long period of history in which many regional standards have existed and no international standards have evolved. The rapid changes in both hardware of devices and pattern of usage are two of the most important reasons which cause difficulties in setting up an international standard in computer graphics. This can be traced back to the early history of graphics that different graphics packages were provided by manufacturers to capitalize on the features of each device.

Nonetheless, standards in computer graphics must be established in order to produce software which facilitates fast and logically controlled graphics. In particular, any software that has to run on multiple devices, or has to be transported to another device or machine, must be able to easily build up a standard interface to each device or machine based on the adopted graphics standards.

In response to this need, a joint graphics working group was formed to deal with the standardization of computer graphics in September 1978. After years of work, the graphics working group has combined some existing graphics standards such as the American standard (CORE system), German standard, and the Norway standard working toward a common specification of a core graphics system known as the *Graphical Kernel System* (GKS). In June 1982, the International Organization for

Standardization (ISO) announced that GKS was adopted as a Draft International Standard.

The GKS is a device independent graphics system that provides an interface between user-level software and graphics input and output devices. A uniform set of functions is defined for a wide variety of graphics devices, which imposes a minimal penalty for differences in devices, [Hop.1].

B.1.2. Organization of the GKS Package

GKS is a comprehensive graphics system containing most of the features required by the application programmer. As a result, GKS is quite large, and for some application, there will be a great deal of unnecessary complexity in GKS. For our purpose, only a subset of the facilities in GKS is required and implemented. These routines can be segregated into groups described as follows:

(1) Control Functions : These routines are used to initialize the internal data structures and graphics devices and to control which device is active.

(2) Output Functions and Attributes : These routines draw simple display items on the device displays, and also change what those items look like.

(3) Coordinate Systems and Transformation Functions : This group of routines defines the coordinates systems used in GKS. It gives the user more control over area of display in the device and regions of the user's coordinate space being viewed.

(4) Input Functions : These functions control different types of input devices.

(5) Inquiry Functions : These fucntions inquire about certain values held within the graphics package.

(6) Error Handling : These routines help users catch, display and recover from errors.

(7) Utility Functions

Having outlined the different aspects of a graphics package, each one will be considered in more depth in Section B.2.

B.1.3. The Device Drive Interface via The MFB System

The current GKS package is primarily for use with frame buffer (raster) devices. Therefore, the low level routines used to control input and output, to and from the device tend to reflect this. Our GKS makes calls to these low level routines via a *virtual graphics interface.* The most general form of this interface is provided by use of the Model Frame Buffer (MFB) package of routines [Bil.1].

The MFB system provides a virtual graphics interface for frame buffer devices. Its purpose is to perform the terminal dependent task of encoding and decoding graphics code, thereby allowing the user to write graphics programs to run on almost any graphics device.

The MFB package is written in C and was originally intended to run under the UNIX operating system; however, an attempt to run it under the VMS has been made. The package depends on the operating system to control the environment in which it has to run. Also, the operating system suspends any interrupts from external processes while in the graphics mode of operation.

A device description database, *MFBCAP* (see reference [Bil.2]) is used by the MFB system. This contains descriptions of each individual device's capabilities and control syntax. Thus, the GKS system calls the MFB package to initialize the routines for control of a certain type of device. If a new display device is acquired, it can be run with the GKS routines when the appropriate device description is added to the MFBCAP database. This means no new routines have to be written to cope with a particular display device, unless an operation that is not contained within the MFB package is required. In such a situation it would be necessary to expand the MFB subroutine libraries.

Most workstations such as VAXstation, SUN, Mathcomp, etc have their own graphics libraries. The portability to these workstations can be easily achieved by writing a set of interface subroutines either between the MFB library and the workstation libraries or between the GKS library and the workstation libraries. The portability of the current GKS implementation is shown in the Fig. B.1.

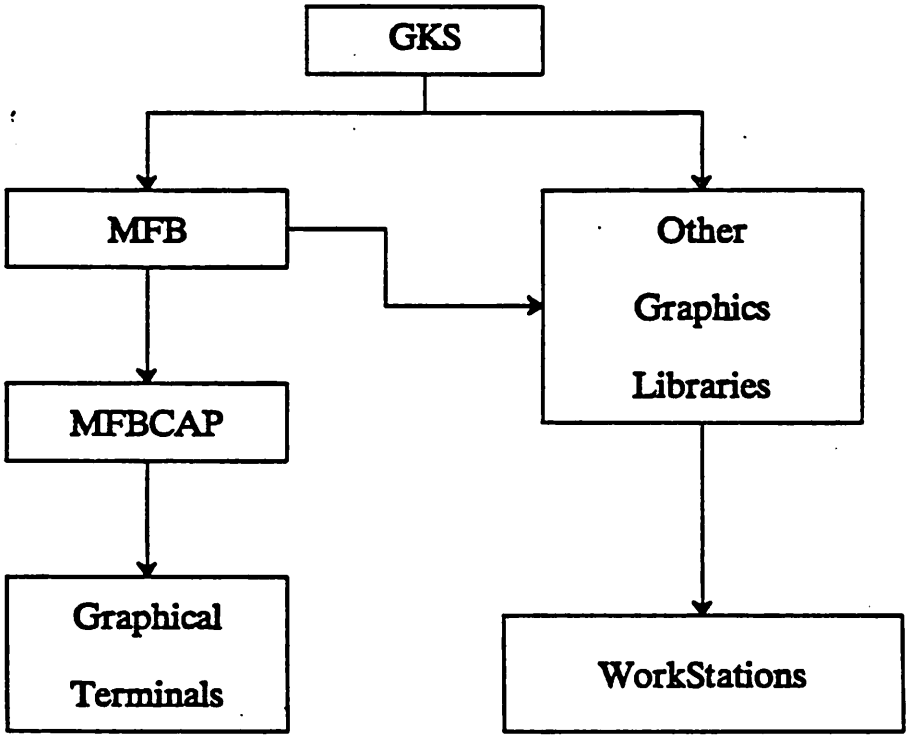


Fig. B.1. GKS Implementation

B.2 GKS Subroutine Users Manual

This section describes the GKS routines and the calling sequences necessary to invoke them. Because the GKS subroutine package is written in C, the names and necessary parameters for C program calls are given. In order to permit the routines to be called in FORTRAN, the arguments of the C subroutines are passed by address. A macro preprocessor file must be included during the compilation of the application programs to make the names of GKS subroutines compatible in both C and FORTRAN calls.

The routines have been segregated into groups relating to the different aspects of operation and control of the graphical display. If default values are set up at initialization time without a program having to make explicit calls to set them up, this is indicated below the subroutine description.

B.2.1 Control Functions for Workstations

Devices being used through GKS are called *workstations*. Before a workstation is accessed, it is necessary for GKS to be initialized. This is accomplished by the routine **OpenGKS**, which creates and initializes all internal data structures, and moves GKS from the GKS closed (*GKSCLOSE*) state into the GKS open (*GKSOPEN*) state.

Every workstation must be opened before it may be used. Once GKS has been opened, it is possible to open workstations by the function **OpenWorkstation**. It creates the data structures necessary to reference the requested workstation, and establishes a connection between the workstation and the computer. Furthermore, if GKS is in the *GKSOPEN* state, it moves into the workstation open (*WSOPEN*) state.

Before an opened workstation can receive output, it must be activated. This is done by the routine **ActivateWorkstation** which, initializes the workstation. Also, if GKS is in the *WSOPEN* state, it moves into the workstation active (*WSACTIVE*) state. This implementation of GKS has no internal limits on the number of workstations open at any time. However, system limitations (e.g. the maximum number of open i/o streams a process may have) will affect the total number of workstations capable of being activated. In this GKS version, we only allow one workstation to be activated at a time.

The function **DeactivateWorkstation** deactivates a workstation. As the program executes, workstations may be activated and deactivated so that one can direct output to various workstations. When the active workstation is deactivated, GKS returns to the *WSOPEN* state.

When the user has no further need for a workstation, the connection between the computer and the workstation should be closed down by **CloseWorkstation**. Active

workstations may not be closed until they have been deactivated. When the last open workstation is closed, GKS returns to the *GKSOPEN* state.

Finally, the function **CloseGKS** closes GKS and returns GKS to the *GKSCLOSE* state. It is necessary to deactivate and close all workstations before GKS is closed.

In summary, the GKS has the following four operating states:

(1) *GKSCLOSE* GKS is closed.

(2) *GKSOPEN* GKS is open, but no workstations are open.

(3) *WSOPEN* GKS is open, and there are workstations open but none is active.

(4) *WSACTIVE* GKS is open, and at least one workstation is active.

The functions which control these operating states are described as follows:

Open GKS - start working with GKS.

C **OpenGKS(errorfile)**
*char *errorfile;*

FORTRAN **gopks(errfil)**
character(*) errfil*

The function initializes the GKS package. It is only called once for an invocation of GKS. The character string *errorfile* specifies the name of the file that will be used by GKS to return error messages to the application program.

Close GKS - Stop working with GKS.

C **CloseGKS()**

FORTRAN **gclks**

This call terminates the GKS package. The error file is closed. All memory allocated for use in GKS is freed and any other housekeeping functions required by the implementation are performed. Note that it is necessary to deactivate and close all workstations that are open prior to closing GKS down.

Open Workstation - Create a connection between the specified workstation and GKS.

C **OpenWorkstation(stationId,connectId,typeId)**
*int *stationId, *connectId, *typeId;*

FORTRAN **gopwk(wkid,conid,wtype)**
integer wkid, conid, wtype

The first parameter, *stationId*, specifies the number to be used by the program to identify the workstation. *connectId* specifies the device channel that the operating system will use for communication with the workstation. *typeId* defines the workstation type of the channel specified by *connectId*. GKS maintains a table of valid types. The parameter *typeId* and the associated workstation name can be accessed via GKS Inquiry Functions: **InquireListWorkstationType** and **InquireWorkstationName**.

Close Workstation - Release the connection between the specified workstation and GKS.

C **CloseWorkstation(stationId)**
*int *stationId;*

FORTRAN **gclwk(wkid)**
integer wkid

This call will ensure that all output to the workstation *stationId* has been transferred to the device and that a disconnection is made from the channels in use. Note that it is necessary to deactivate the workstation *stationId* before it is closed.

Activate Workstation - Output is routed to the specified workstation.

C **ActivateWorkstation(stationId)**
*int *stationId;*

FORTRAN **gacwk(wkid)**
integer wkid

A workstation, specified by *stationId* when the workstation was opened, is activated. All output is sent to the device for display. In current implementation, only one workstation is allowed to be active. If the workstation is active, this call is ignored.

Deactivate Workstation - Output is no longer routed to the specified workstation.

C **DeactivateWorkstation(stationId)**
*int *stationId;*

FORTRAN **gdawk(wkid)**
integer wkid

Deactivate a workstation which is specified by *stationId* when the workstation was opened. If the workstation is not active, this call is ignored.

B.2.2 Output Functions and Attributes

In GKS, pictures are constructed from a number of basic building blocks, or primitives as they are called. These primitives are of a number of types each of which can be used to describe a different component of a picture. The four main primitives in GKS are:

(1) *polyline* - draws a sequence of connected line segments.

(2) *polymarker* - marks a sequence of points with the same symbol.

(3) *fill area* - displays a specified area.

(4) *text* - draws a string of characters.

Additional data are necessary to describe the actual appearance of the primitives. (e.g. color, line style, fill pattern, and etc.) These additional data are known as *attributes* which represent features of the primitives.

In the rest of this subsection, we describe the usage of GKS routines for these four primitive types of GKS given above and their associated attributes. The positions of points used in the output primitives are specified in (user) world coordinates which will be explained in the subsection B.2.3.

Polyline - Generates a polyline defined by points in world coordinates.

C **Polyline(numpts,xpoints,ypoints)**
*int *numpts;*
*double *xpoints, *ypoints;*

FORTRAN **gpl(npts,xp,yp)**
integer npts
double precision xp(npts), yp(npts)

The polyline generated consists of *numpts-1* line segments joining *numpts* points starting with the first points, (*xpoints(1)*, *ypoints(1)*), and ending with the last,

(*xpoints*(*numpts*), *ypoints*(*numpts*)).

Polymarker - Generates markers of a given type at specified points in world coordinates.

C **Polymarker(numpts,xpoints,ypoints)**
*int *numpts;*
*double *xpoints, *ypoints;*

FORTRAN **gpm(npts,xp,yp)**
integer npts
double precision xp(*npts*), *yp*(*npts*)

This places centered markers at specified points in world coordinates. The arguments are the same as for the **Polyline** function.

Text - Generates a text string at the given position in world coordinates.

C **Text(xcoord,ycoord,text)**
*double *xcoord, *ycoord;*
*char *text;*

FORTRAN **gtx(xc,yc,text)**
double precision xc, yc
*character *(*) text*

Generates a text string at the given position, (*xcoord,ycoord*), in world coordinates. The parameter *text* stores the text string.

Fill Area - Generates a filled polygon of a given pattern or type.

C **FillArea(numpts,xpoints,ypoints)**
*int *numpts;*
*double *xpoints, *ypoints;*

FORTRAN **gfa(npts,xp,yp)**
integer npts
double precision xp(*npts*), *yp*(*npts*)

Generates a polygon which may be filled with a color, a pattern, or which may be hollow. The arguments are the same as for the **Polyline** function. If the area is not closed (i.e. the first point is not the same as the last point), the boundary is the polyline defined by the points but extended to join the last point to the first point.

Set Polyline Index - Selects a bundle index for polylines.

C **SetPolylineIndex(lineId)**
*int *lineId;*

FORTRAN **gspli(lindex)**
integer lindex

This sets the polyline index to the value *lineId* for subsequent polylines. The polyline index *lineId* points into a bundle table which is defined by the GKS routine **SetPolylineRepresentation**.

Set Line Type - Sets the line type independently for use.

C **SetLineType(ltype)**
*int *ltype;*

FORTRAN **gsln(ltype)**
integer ltype

This sets the polyline type index to the value *ltype* for subsequent polylines. By default, zero specifies the solid line.

Set Polyline Color Index - Sets the polyline color index.

C **SetPolylineColorIndex(colorId)**
*int *colorId;*

FORTRAN **gsplci(coli)**
integer coli

This function sets the polyline color index *colorId* which is defined by the GKS routine **SetColorRepresentation**.

Set Polymarker Index - Selects a bundle index for polymarkers.

C **SetPolymarkerIndex(markId)**
*int *markId;*

FORTRAN **gspmi(markid)**
integer markid

This sets the polymarker index to the value *markId* for subsequent polymarkers. The polymarker index *markerId* points into a bundle table which is defined by the GKS routine **SetPolymarkerRepresentation**.

Set Polymarker Color Index - Sets the polymarker color index.

C **SetPolymarkerColorIndex(colorId)**
*int *colorId;*

FORTRAN **gspmci(coli)**
integer coli

This function sets the polymarker color index *colorId* which is defined by the GKS routine **SetColorRepresentation**.

Set Text Color Index - Sets the text color index.

C **SetTextColorIndex(colorId)**
*int *colorId;*

FORTRAN **gstxci(coli)**
integer coli

This function sets the text color index *colorId* which is defined by the GKS routine **SetColorRepresentation**.

Set Fill Area Index - Selects a bundle index for fill area.

C **SetFillAreaIndex(areaId)**
*int *areaId;*

FORTRAN **gsfai(areaid)**
integer areaid

This sets the fill area index to the value *areaId* for subsequent fill areas. The fill area index *areaId* points into a bundle table which is defined by the GKS routine **SetFillAreaRepresentation**.

Set Fill Area Color Index - Sets the fill area color index.

C **SetFillAreaColorIndex(colorId)**
*int *colorId;*

FORTRAN **gsfaci(coli)**
integer coli

This function sets the fill area color index *colorId* which is defined by the GKS routine **SetColorRepresentation**.

Set Polyline Representation - Defines the representation of polylines on the specified workstation.

C **SetPolylineRepresentation(stationId,lineId,ltype,lwidth,colorId)**
*int *stationId, *lineId, *ltype, *colorId;*
*double *lwidth;*

FORTRAN **gsplr(wkid,pli,ltype,lwidth,coli)**
integer wkid, pli, ltype, coli
double precision lwidth

For each polyline index value *lineId*, the function defines a representation on a particular workstation *stationId*. *ltype* specifies the line type (e.g. solid line, dashed line, dotted line, etc). *lwidth* specifies the line width scale factor. *colorId* specifies the polyline color index.

Set Polymarker Representation - Defines the representation of polymarkers on the specified workstation.

C **SetPolymarkerRepresentation(station,mark,type,scale,color)**
*int *station, *mark, *type, *color;*
*double *scale;*

FORTRAN **gspmr(wkid,pmi,mtype,mszsf,coli)**
integer wkid, pmi, mtype, coli
double precision mszsf

For each polymarker index value *mark*, the function defines a representation on a particular workstation *station*. *type* specifies the marker type which defines the form of the marker to be displayed. *scale* specifies the marker size scale factor. *color* specifies the polymarker color index.

Set Fill Area Representation - Defines the representation of fill area primitives on the specified workstation.

C **SetFillAreaRepresentation(station,area,interior,style,color)**
*int *station, *area, *interior, *style, *color;*
*double *scale;*

FORTRAN **gsfar(wkid,fai,ints,styli,coli)**
integer wkid, fai, ints, styli, coli

For each fill area index value *area*, the function defines a representation on a particular workstation *station*. *interior* specifies the fill area interior style. The possible values for the fill area interior style are *HOLLOW, SOLID, PATTERN, HATCH*. *style* which is defined by the GKS routine **SetPatternRepresentation** specifies the fill area style index. It only has meaning for the interior styles *PATTERN* and

HATCH. *color* specifies the fill area color index.

Set Pattern Representation - Defines the pattern to be associated with a pattern index (i.e. a fill area style index) on the specified workstation.

C **SetPatternRepresentation(station,pattern,xsize,ysize,patternAry)**
*int *station, *pattern, *xsize, *ysize, patternAry* [ ];

FORTRAN **gspar(wkid,pai,nx,ny,patary)**
integer wkid, pai, nx, ny, patary(nx,ny)

pattern specifies the pattern index associated with the entry to be defined on the workstation *station*. *xsize* and *ysize* specify the number of cells of the pattern in the X and Y directions while the *patternAry*, which is a *xsize* by *ysize* FORTRAN array, specifies the color index table for each cell. In current GKS implementation, *xsize* and *ysize* have been fixed to number 8.

Set Color Representation - Defines the color to be associated with a color index on the specified workstation.

C **SetColorRepresentation(stationId,colorId,red,green,blue)**
*int *stationId, *colorId;*
*double *red, *green, *blue;*

FORTRAN **gscr(wkid,coli,red,green,blue)**
integer wkid, coli
double precision red, green, blue

For a color display, this function defines (redefines) the entries in the color look up table pointed at by the color index *colorId*. Each entry in the table consists of the intensity values of red, green and blue colors. *red, green* and *blue* specify the respective intensities which are in the range 0 to 1.

B.2.3 Coordinate Systems and Transformation Functions

GKS provides three Cartesian coordinate systems at all times, which are sufficient to allow complete freedom in conversion from drawing primitives to their display. These coordinate systems are described as follows:

World World Coordinates (WC) are those in which the graphical output is described to GKS. These coordinate systems are defined by the user

for various applications.

Device The Device Coordinate (DC) system, which varies significantly from one device to another, is implemented by a workstation's designers. For a frame buffer (raster) device, the coordinates will be defined by the pixel resolution of the display.

Normalized The Normalized Device Coordinate (NDC) system is a virtual device coordinates system which defines a display surface visible in the range 0 to 1 in both the X and Y directions on the virtual device. In other words, NDC space is in the unit square. It is therefore independent of both user coordinates and device coordinates.

A *transformation* is a way of converting the units in one coordinate system into those of another. GKS uses transformations to convert World Coordinates to Normalized Device Coordinates and from Normalized Device Coordinates to Device Coordinates.

Window is a rectangular area defined in world coordinate space, and *viewport* defines in which this area will appear on the normalized device (specified by the NDC system). GKS uses the rectangular window and viewport pairs to specify how WC are transformed to NDC. The transformation of coordinates from WC to NDC defined by the window to viewport mapping is often called a *normalization transformation* in GKS.

Workstation window defines a rectangle area of NDC space which will be seen at workstation and *workstation viewport* defines where this area will appear on the display screen. The workstation viewport is specified in the appropriate device coordinates. The transformation of coordinates from NDC to DC defined by the workstation window to workstation viewport is a so-called *workstation transformation* in GKS. Unlike normalization transformations, there is only one workstation transformation per workstation which is used for the whole picture.

The usage of GKS routines for manipulating these coordinate systems and transformations are described as follows:

Set Window - Sets the window in world coordinates of the specified normalization transformation.

C **SetWindow(tranId,xmin,xmax,ymin,ymax)**
*int *tranId;*
*double *xmin, *xmax, *ymin, *ymax;*

FORTRAN **gsw(tranid,xmin,xmax,ymin,ymax)**
integer tranid
double precision xmin, xmax, ymin, ymax

Sets the *tranId*th window in world coordinates (*xmin, xmax, ymin, ymax*). The associated viewport to the *tranId*th window is specified by the GKS routine **SetViewport**.

Set Viewport - Sets the viewport in normalized device coordinates of the specified normalization transformation.

C **SetViewport(tranId,xmin,xmax,ymin,ymax)**
*int *tranId;*
*double *xmin, *xmax, *ymin, *ymax;*

FORTRAN **gsvw(tranid,xmin,xmax,ymin,ymax)**
integer tranid
double precision xmin, xmax, ymin, ymax

Sets the *tranId*th viewport in normalized device coordinates (*xmin, xmax, ymin, ymax*). The associated window to the *tranId*th viewport is specified by the GKS routine **SetWindow**.

Set Viewport Input Priority - Sets the input priority of the specified viewport for locator input.

C **SetViewportInputPriority(TranId,referTranId,priority)**
*int *TranId, *referTranId, *priority;*

FORTRAN **gsvwip(tranid,rtrani,relpri)**
integer tranid, rtrani, relpri

In some cases where viewports overlap, a *viewport input priority* must be associated with each normalization transformation so that an input device can select the transformation with the highest viewport input priority to perform the transformation back to world coordinates. This function defines the viewport input priority of the transformation *TranId* to be higher or lower than *referTranId* depending on the value (*HIGHER* or *LOWER*) of parameter *priority*.

Select Normalization Transformation - Selects a normalization transformation for output.

C **SelectNormalizationTransformation(tranId)**
*int *tranId;*

FORTRAN **gselnt(tranid)**
integer tranid

For multi-viewport graphical systems, a normalization transformation must be specified with a particular output primitive. This specifies that the normalization transformation *tranId* (the *tranId*th window to the *tranId*th viewport mapping) is selected to be used with output primitives.

Set Clipping Indicator - Sets the clipping indicator for the current normalization transformation.

C **SetClippingIndicator(clipInd)**
*int *clipInd;*

FORTRAN **gsclin(clipid)**
integer clipid

Sets the clipping indicator for the current normalization transformation. The parameter *clipInd* may take the values *CLIP* and *NOCLIP* as appropriate. Note that the clipping indicator is associated with the currently selected normalization transformation. There is not a separate clipping switch for each normalization transformation.

Set Workstation Window - Sets the workstation window in normalized device coordinates.

C **SetWorkstationWindow(stationId,xmin,xmax,ymin,ymax)**
*int *stationId;*
*double *xmin, *xmax, *ymin, *ymax;*

FORTRAN **gswkw(wkid,xmin,xmax,ymin,ymax)**
integer wkid
double precision xmin, xmax, ymin, ymax

This sets the workstation window in normalized device coordinates (*xmin, xmax, ymin, ymax*). This specifies that area of the normalized device coordinates (NDC) space in the range 0 to 1 which will be output to the workstation *stationId*. The associated workstation viewport is specified by the GKS routine **SetWorkstationViewport**.

Set Workstation Viewport - Sets the workstation viewport in device coordinates.

C **SetWorkstationViewport(stationId,xmin,xmax,ymin,ymax)**
*int *stationId;*
*double *xmin, *xmax, *ymin, *ymax;*

FORTRAN **gswkvw(wkid,xmin,xmax,ymin,ymax)**
integer wkid
double precision xmin, xmax, ymin, ymax

This function sets the workstation viewport in device (raster) coordinates (*xmin, xmax, ymin, ymax*). It defines the location on the display screen where the view of NDC space will appear. The associated workstation window is specified by the GKS routine **SetWorkstationWindow**.

Evaluate Transformation Matrix - Evaluates a transformation matrix for a set of given information.

C **EvaluateTransformationMatrix(fx,fy,dx,dy,angle,sx,sy,coor,mat)**
*double *fx, *fy, *dx, *dy, *angle, *sx, *sy, *mat;*
*int *coor;*

FORTRAN **gevtm(fx,fy,dx,dy,angle,sx,sy,coor,mat)**
double precision fx, fy, dx, dy, angle, sx, sy, mat(2,3)
integer coor

The transformation specified by fixed point (*fx,fy*), shift vector (*dx,dy*), rotation angle *angle* (in radians) and scale factors (*sx,sy*) is evaluated and the result is returned in the output 2x3 transformation matrix *mat*. The coordinate switch, *coor*, can take the value WC or NDC.

B.2.4 Input Functions

Request Locator - Requests position in world coordinates and normalization transformation number from the specified locator device.

C **RequestLocator(stationId,locatorId,status,tranId,xpos,ypos)**
*int *stationId, *locatorId, *status, *tranId;*
*double *xpos, *ypos;*

FORTRAN **grqlc(wkid,lcid,stat,tranid,xpos,ypos)**
integer wkid, lcid, stat, tranid
double precision xpos, ypos

This function returns a position (*xpos,ypos*) in world coordinates, and the corresponding normalization transformation *tranId* is used to map back from normalized device coordinates to world coordinates. The parameter *stationId* specifies the workstation on which the specific **Locator** device (e.g. pen or mouse) is located and *locatorId* specifies which of the **Locator** devices it is. The third parameter, *status*, is a status indicator which returns information concerning how successful the logical device was in providing the requested values. *status* takes values *OK* or *NO*.

Initialize String - Initializes the specified string device.

C **InitializeString(station,strid,str,echo,xmin,xmax,ymin,ymax,len,data)**
*int *station, *strid, *echo, *len, data[ ];*
*double *xmin, *xmax, *ymin, *ymax;*
*char *str;*

FORTRAN **ginst(wkid,strid,str,echo,xmin,xmax,ymin,ymax,len,data)**
integer wkid, strid, echo, len, data(len)
double precision xmin, xmax, ymin, ymax
character(*) str*

This function initializes the specified string device. The parameter *station* specifies the workstation on which the specific **String** device (e.g. keyboard) is located and *strid* specifies which of the **String** devices it is. The third parameter, *str*, is an initial string of the string device. The parameter *echo* gives the prompt and echo types. If *echo* is equal to 1, then the screen will display the current string value in a rectangle echo area which is specified in device coordinates (*xmin, xmax, ymin, ymax*). Some string devices allow further control over the way they interact with the user. For example: initial cursor position, the size or shape of the cursor, the maximun number of input characters, etc. This information can be packed into a data record, specified by *data(len)*. In the current implementation, the parameters, *str, echo, len*, and *data* are dummy arguments. The initial cursor position is always at (*xmin, ymin*).

Request String - Requests a character string from the specified string device.

C **RequestString(stationId,strId,status,len,string)**
*int *stationId, *strId, *status, *len*
char string[ ];

FORTRAN **grqst(wkid,strid,stat,len,str)**
integer wkid, strid, stat, len
character(*) str*

This function returns a character string stored in the parameter *string* to the application program. The parameter *stationId* specifies the workstation on which the specific **String** device (e.g. keyboard) is located and *strId* specifies which of the **String** devices it is. The third parameter, *status*, is a status indicator which returns information concerning how successful the logical device was in providing the requested values. *status* takes values *OK* or *NO*. The parameter *len* is an input/output parameter. On the input, it passes the maximum length of the character array *string*[ ] to the routine **RequestString**. On the output, the parameter *len* returns the number of characters contained in the string array *string* which returns from the string input device.

B.2.5 Inquiry Functions

Inquire Operating States - Inquires GKS operating states.

C **InquireOperationStates(state)**
*int *state;*

FORTRAN **giqos(state)**
integer state

This function inquires the current GKS operating state. The parameter *state* will take one of the values: *GKSCLOSE, GKSOPEN, WSOPEN, WSACTIVE.*

Inquire List Workstation Type - Inquires a list of workstation names and correspondent type indices available in GKS package.

C **InquireListWorkstationType(numstation,typeid,stationame)**
*int *numstation, typeid*[ ];
*char *stationame*[ ];

FORTRAN **giqlws(nstat,typeid,NULL)**
integer nstat, typeid(*nstat*)

This function inquires a list of workstation names and correspondent type indices available in the GKS package. The function will return names and type indices in the arrays *typeid*[ ] and **stationame*[ ]. For input, *numstation* is the dimension of the arrays *typeid*[ ] and **stationame*[ ]. For output, *numstation* returns the maximum

number of available workstation types. If the maximum number of available workstation types is greater than the value of input parameter *numstation*, the return information in the arrays *typeid*[ ] and **stationame*[ ] will be truncated. If a *NULL* value is passed to the parameter *stationame*, the returned workstation name list will be disregarded. Note: the third parameter of the subroutine **giqlws** should be assigned to a *NULL* pointer in order to keep the argument compatible with C. For the FORTRAN user, the name of the workstation type can be obtained by the GKS routine **InquireWorkstationName**.

Inquire Workstation Name - Inquires a workstation name with the specified type index.

C **InquireWorkstationName(typeid,stationame)**
*int *typeid;*
*char *stationame;*

FORTRAN **giqwsn(typeid,name)**
integer typeid
character(*) name*

Inquires a workstation type name *stationame* (e.g. "tek4115") which corresponds to a given type index *typeId*.

Inquire Maximum Displays - Inquires the maximum display sizes of the specified workstation.

C **InquireMaximumDisplays(stationId,xmax,ymax)**
*int *stationId, *xmax, *ymax;*

FORTRAN **giqmdy(wkid,xmax,ymax)**
integer wkid, xmax, ymax

Inquires the maximum display size (*xmax,ymax*) in raster units for the workstation *stationId*.

Inquire Color Facilities - Inquires the color facilities of the specified workstation.

C **InquireColorFacilities(stationId,iscolor,maxcolor)**
*int *stationId, *iscolor, *maxcolor;*

FORTRAN **giqcf(wkid,iscol,maxcol)**
integer wkid, iscol, maxcol

This function inquires the color facilities of the workstation *stationId*. *iscolor* returns *YES* if the workstation has a Video Lookup Table (VLT), otherwise returns *NO*. *maxcolor* returns the maximum number of available colors for the workstation *stationId*.

B.2.6 Error Handling

GKS has a well defined set of error messages which will be reported back to the application program. The philosophy adopted in GKS is to report all errors by putting details of the error in the error file which was specified when GKS was opened. A typical error will record the following information:

(1) Error Number: a number indicating which error has occurred.

(2) GKS Function: the name of the GKS function that was being called when the error was detected.

What actions will be taken when an error has been recognized depends on the application program. GKS provides an entry for user-supplied error handling procedure. If no user routine has been specified, GKS calls the installation-supplied error handling procedure **ErrorHandling**.

Error Logging - A procedure called by the standard GKS error handling procedure.

C **ErrorLogging(errorId,funcId,errorfp)**
*int *errorId, *funcId;*
*FILE *errorfp;*

FORTRAN **gerlog(errid,fnid,NULL)**
integer errid, fnid

Writes the error message associated with error number *errorId* and the name of the calling GKS function associated with function index *funcId* onto the file (FILE *)*errorfp*. The parameter *errorfp* may be *stdout*, *stderr*, or NULL. If *errorfp* is NULL, the error file which specified by the GKS routine **OpenGKS** will be used to record the error message. This function is usually called by the GKS routine **ErrorHandling**.

Error Handling - A procedure called by GKS when an error is detected. It may be user-supplied.

C **ErrorHandling(errorId,funcId,errorfp)**
*int *errorId, *funcId;*
*FILE *errorfp;*

FORTRAN **gerhnd(errid,fnid,NULL)**
integer errid, fnid

The parameters are the same as for the **ErrorLogging** function. The default version of this routine simply calls **ErrorLogging**. If the user wishes to handle errors, he may write his own error-handling routine **ErrorHandling** with the same parameters. The user-supplied error handling routine should still call **ErrorLogging** before returning. The only GKS functions which can be invoked in the error handling procedure are: **ErrorLogging**, **InquireXXX** and **EmergencyCloseGKS**.

Emergency Close GKS - Close GKS in case of an error.

C **EmergencyCloseGKS()**

FORTRAN **geclks**

This function tries to close GKS and ensure that any output information buffered within GKS is transmitted to the workstation. GKS itself will invoke this function in response to some classes of unrecovered error. This function may be invoked by a user-supplied error handling procedure.

B.2.7 Utility Functions

GKS Flush - Flush buffered output.

C **GKSFlush()**

FORTRAN **gflush**

This function flushes the internal output buffer to the currently defined output device and will ignore any writing error that may occur.

Window Erase - Erases a rectangular area (window).

C **WindowErase(x1, y1, x2, y2)**
double x1, y1, x2, y2;

This subroutine erases a rectangular area which is described by the world coordinates (x1,y1) and (x2,y2) of the current normalization transformation with x1 $\leq$ x2 and y1 $\leq$ y2.

GKS Draw Box - Draws a box.

C **GKSDrawBox(x1, y1, x2, y2)**
double x1, y1, x2, y2;

GKS Fill Box - Generates a fill box.

C **GKSFillBox(x1, y1, x2, y2)**
double x1, y1, x2, y2;

GKS Draw Circle - Draws a circle.

C **GKSDrawCircle(x1, y1, x2, y2, nsides)**
double x1, y1, x2, y2;
int nsides;

GKS Fill Circle - Generates a fill circle.

C **GKSFillCircle(x1, y1, x2, y2, nsides)**
double x1, y1, x2, y2;
int nsides;

APPENDIX C

MIMO HELP FACILITY

C.1. Introduction

This Appendix describes a sophisticated, easy-to-use help facility available in the MIMO system. This help facility provides quick on-line assistance for whatever type of information has been set up in files by MIMO system personnel. The format of these *help* files is similar to the file format of the DELIGHT *helper* facility [Nye.2]. As a simple example, the MIMO help facility can process a file containing the following lines:

```
%N  shell - Shell escaped.
%U  shell [shell command line]
%D  "shell"  allows you to execute any  UNIX  shell command inside  of
%+  MIMO.  For instance, to see all users who are currently  working on
%+  the machine, you may type "shell who".  If "shell" is typed without
%+  arguments,  MIMO  creates  a  new "csh" under the  UNIX  system.
%+  Return to MIMO system is by exiting the shell.
%E  shell
%+  shell uptime ; ps ux
%+  shell ps au | grep MIMO
%+  shell date ; who am i
%SA  !
%A  Stephen Wuu
```

Then, users can obtain the following output by typing *help shell* while running the MIMO system:

```
NAME
        shell - Shell escaped.
USAGE
        shell [shell command line]
```

DESCRIPTION

"shell" allows you to execute any UNIX shell command inside of MIMO. For instance, to see all users who are currently working on the machine, you may type "shell who". If "shell" is typed without arguments, MIMO creates a new "csh" under the UNIX system. Return to MIMO is by exiting the shell.

EXAMPLE

shell
shell uptime ; ps ux
shell ps au | grep MIMO
shell date ; who am i

SEE ALSO

!

AUTHOR

Stephen Wuu

This Appendix is organized as follows. Section 2 gives the usage, description, and examples of the basic interactive *help* command by showing actual output from the help facility when *help help* is typed. Section 3 shows how MIMO system personnel (or users) create a *help* entry so that it can be used by others. Finally, Section 4 describes how users can make a hard copy of on-line documentation.

C.2. Interactive Help Facility : Usage and Description

The MIMO help facility is designed for running in an interactive environment. The *help* command can be issued interactively whenever the system is waiting for a command from the user. The usage and description of the *help* command are shown below, which is actual output from the help facility:

NAME

help - Provide on-line help for MIMO commands and features.

USAGE

help [[-options] COMMAND_NAME]

help -k KEYWORD

help -s [MATCH_PATTERN]

help -f FILE_NAME

DESCRIPTION

"help" makes available quick on-line assistance for commands, features, or any other topic for which information has been set up in help file format. To see a list of all available help entries, type "help -s *".

OPTIONS

-s - list available subjects whose leading letters match the given pattern, e.g., l* or subsys*.

-k - list available subjects whose %N (Name) lines contain the given KEYWORD.

-u - print the usage of COMMAND_NAME.

-d - print the description of COMMAND_NAME.

-o - print the options of COMMAND_NAME.

-e - print the examples of COMMAND_NAME.

-f - includes helper files.
Type "help help_file" to see the file format.

EXAMPLE

help load

help -s

help -s c*

help -k MIMO

help -f HelpMIMO

help -u load

help -d help

help -o help

help -e load

help

SEE ALSO

? , help_file, echo_to, echo_onto and echo_end.

C.3. Creating Simple Help Entries in Files

To create a help entry for the MIMO help facility, both system personnel and users just create a file with the following format (again, this is the output from the help facility when *help help_file* is typed):

NAME

help_file - File format description for the "help" command.

DESCRIPTION

To create files needed for quick table lookup by "help", one first creates a file using the following format for each item for which help may be requested:

%N Name - Brief description
%U Usage
%D Description
%O Options
%E Examples
%SA See Also
%B Bugs
%SF Source Files
%A Author
%CL Command list for level prompt
%+ Continuation of any of above

EXAMPLES

%N echo_to - Make all succeeding alphanumeric output echo to a file.
%U echo_to FILE_NAME
%D "echo_to" provides user hard copy of all alphanumeric output sent
%+ to the terminal screen. It creates file "FILE_NAME" to store, e.g.,
%+ output data, MIMO help manuals or error messages for debugging.
%E echo_to man.load
%+ help load
%+ echo_end
%+ echo_to Output.1
%SA echo_onto, echo_end.

SEE ALSO

help, echo_to, echo_onto and echo_end.

An example of a file called, say, *testhelp* that contains two help entries follows:

```
%N  bell - turn on or off the error message bell on/off or show its state.
%U  bell [ on/off ]
%+  bell [ y/n ]
%D  "bell on" or "bell y" turn on the error message bell which causes the
%+  bell to ring whenever an error occur. "bell off" or "bell n" turn off
the
%+  bell. Typing "bell" alone shows whether it is on or off.
%E  bell
%+  bell on
%+  bell off
%N  terminal - Set terminal type or show terminal choices or type.
%U  terminal [TERM_NAME]  - Set/Show the present terminal type.
%+  terminal choices             - Show terminal choices.
%E  terminal
%+  terminal choices
%+  terminal tek4115pen
```

After creating this file, one could go into MIMO (by simply typing MIMO) and type the command *help -f testhelp*. This would cause MIMO to read file *testhelp* and set up help entries for these two commands. One could then immediately test the entries by typing *help bell* or *help terminal*. This sequence is summarized in the following terminal dialogue and what is actually typed is shown in **boldface** print:

```
% MIMO
>> help -f testhelp
including file <testhelp>
>> help bell
NAME
    bell - turn on or off the error message bell or show its state.
```

USAGE

bell [on/off]

bell [y/n]

DESCRIPTION

"bell on" or "bell y" turn on the error message bell which causes the bell to ring whenever an error occur. "bell off" or "bell n" turn off the bell. Typing "bell" alone shows whether it is on or off.

EXAMPLE

bell

bell on

bell off

≫ quit

C.4. Generating Hard Copy of MIMO documentations

In many instances, users need hard copy of the help entries that can be viewed online. In the MIMO system, hard copy of these help entries can be easily created using the commands *echo_to*, *echo_onto* and *echo_end*. The usage and description of these commands are shown as follows:

NAME

echo_to - Make all succeeding alphanumeric output echo to a file.

USAGE

echo_to FILE_NAME

DESCRIPTION

"echo_to" provides the user hard copy of all alphanumeric output sent to the terminal screen. It creates file "FILE_NAME" to store, e.g., output data, MIMO help manuals or error messages for debugging.

EXAMPLE

echo_to man.load

help load

echo_end

echo_to Output.1

SEE ALSO

echo_onto, echo_end.

NAME

echo_onto - Appends all succeeding alphanumeric output to the end of a file.

USAGE

echo_onto FILE_NAME

DESCRIPTION

"echo_onto" provides user hard copy of all alphanumeric output sent to the terminal screen. It appends, e.g., output data, MIMO help entries or error messages to the end of the file "FILE_NAME".

EXAMPLE

```
echo_onto man.load
help load
echo_end
echo_onto Output.1
```

SEE ALSO

echo_to, echo_end.

NAME

echo_end - Turn off echo_to or echo_onto.

USAGE

echo_end

DESCRIPTION

"echo_end" turns off the echoing of output started by "echo_to" or "echo_onto".

SEE ALSO

echo_to, echo_onto.

Now, we can generate hard copy of help entries using the following commands:

```
% MIMO
>> echo_to man.load
```

```
Created file "man.load"
>> help load
NAME
      load - Load subsystems, input signals or files.
      .....
>> echo_end
Echo output is in file "man.load"
>> shell lpr man.load
```