70-01250

COLUMBIA UNIVERSITY

SYSTEMS RESEARCH GROUP

DEPARTMENT OF
ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING AND
APPLIED SCIENCE
NEW YORK, N.Y. 10027

N70-39720

COMPUTATIONAL METHODS FOR DIGITAL SIMULATION OF CONTINUOUS SYSTEMS

Richard A. DiPerna

Columbia University
New York, N. Y.

June 1970

Technical Report No. 123

COMPUTATIONAL METHODS FOR DIGITAL SIMULATION OF CONTINUOUS SYSTEMS

Richard A. DiPerna

June 1970

This work was supported in part by the National Aeronautics and Space Administration under Grant NGR-33-008-090 and by the National Science Foundation under Grant GK-2283. This research was done while the author was an Armstrong Fellow at Columbia University. Reproduction in whole or in part is permitted for any purpose of the United States Government.

ACKNOWLEDGMENT

Foremost, the author gratefully acknowledges Professor Thomas E. Stern for his encouragement and invaluable guidance and assistance throughout the period of research leading to this dissertation.

For their advice and helpful comments the author owes thanks to Professors Henry E. Meadows, Omar Wing and Ralph J. Schwarz. And to his colleagues, R. Jeffrey Bogumil and Satish K. Manocha, the author wishes to express his thanks for their careful reading of the typescript and for their suggestions. The author owes special appreciation to his wife Sharon for her help in editing all phases of the writing and for proofreading the final copy of this manuscript.

Finally, the author will always be deeply grateful to the Armstrong Memorial Research Foundation for its financial support during the author's years of study at Columbia University.

ABSTRACT

This research develops computational methods for the digital simulation of continuous systems. These methods are applicable to a wider class of systems and are more efficient than the methods used in currently available simulation languages. Two fundamentally different approaches to simulation are distinguished and described, and the characteristics of the simulation which result from using these methods are investigated.

Two serious limitations of available simulation languages are their inability to solve sets of implicit equations and to deal efficiently with stiff systems. Methods suitable for the efficient handling of stiff systems are developed within the framework of the two approaches to simulation. It is shown that these methods yield implicit equations for the overall simulation. In order to structure simulation languages to deal with implicit equations, sorting and iteration procedures for the efficient solution of these equations are presented.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd.)

CHAPTER I

INTRODUCTION

1.1 Motivation

As systems increase in size and complexity, simulation becomes an essential tool in their analysis and design. Simulation is used to corroborate theoretical derivations, to help in evaluation of alternate designs, to generate failure data without costly physical testing, to train human operators, to optimize design parameters, and for many other purposes.

Simulation may be applied to either continuous-time or discrete-time systems. The variables of continuous-time systems (hereafter referred to as continuous systems) are capable of changing at any instant of time, whereas the variables of discrete-time systems (discrete systems) change only at discrete instants, and their values at all other points are of no interest. It is the simulation of continuous systems on the digital computer that will be considered in this thesis. A point worth noting, however, is that the modeling of continuous systems for simulation on the digital computer necessarily involves the theory of discrete systems since the digital computer functions as a discrete-time machine.

Prior to 1960, the simulation of continuous systems was carried out almost exclusively on the analog computer. Over the past decade, however, interest in the use of the digital computer has grown, apparently beginning when the digital computer was first used to give check solutions for analog simulations. There are many reasons for the increased use of the digital computer for simulation, including greater accuracy, accessibility, no hardware setup, availability of mathematical functions and, most importantly, the existence of problem oriented simulation languages which allow the analyst to interface easily with the computer. Also, the growing availability of terminals and display devices is increasing the man-machine interaction, which was previously possible only with the analog computer.

Simulation languages for the digital computer have grown to a high level of sophistication, and can simulate a large class of systems. However, almost all languages are integrator oriented, that is, they separate integrators from the system and consider what remains as a functional block, which may be used as desired by a central integration routine. While for many systems this is effective, if any subsystem does not lend itself to evaluation as desired by the central integration routine, then difficulties may arise.

The goal of this research is to develop computational techniques both for the modeling of individual subsystems and for interconnection and coordination of the overall simulation. When incorporated into a general simulation language these techniques will extend the class of systems to which the language may be applied and increase its efficiency.

1.2 Historical Background

The first digital analog simulator was reported by Selfridge in 1955 [1.1]. This was a digital computer program developed at the U.S. Naval Ordinance Testing Station, Inyokern, Calif., to accept a larger problem and provide more accuracy than was possible on the available REAC (Reeves Electronic Analog Computer). Since that time, the field of simulation languages for digital computers has grown at a rapid rate. A history of simulation languages is given in several survey papers [1.2-1.4], thus only that history which is necessary to motivate this research will be presented here.

In their survey article, Clancy and Fineberg [1.3] state that the "essence" (in the metaphysical sense) of simulation languages may be parallelism, the apparent parallel operation of a serial digital computer. In many simulation languages, this apparent parallel operation is achieved by the analyst, who orders the input statements to the computer so that the proper computational sequence is followed at execution time. If one block follows another in the system under simulation, then the analyst must describe them to the computer in that order. Stein and Rose [1.5] in 1958 provided the theoretical and practical background needed to write a sorting routine, i.e., an algorithm to process the input statements in whatever order they appear and to deduce the proper computational sequence from them.

Although overlooked by many later authors, a sorting algorithm is one of the keys to a useful language. A sorting algorithm is contained in most of the successful simulation languages, including MIDAS (Modified Integration Digital Analog Simulator) [1.6], MIMIC [1.7], PACTOLUS [1.8], DSL/90 [1.9] and most recently 360/CSMP [1.10].

Each of these languages, however, treats integrator outputs as known entities, and thus the sorting algorithm always begins the computational sequence at the integrator outputs. If the system being simulated has memoryless loops, or if any subsystem has been modeled by an implicit difference equation, then these sorting algorithms will fail. The existence of memoryless loops will be discovered by 360/CSMP and a message to that effect printed for the user, and the simulation will not be continued. The user may then break each memoryless loop with a special coding block, and, provided that there are no multiply imbedded loops, the program will attempt to solve the loop by a very simple iteration procedure. It is one purpose of the present investigation to develop a sorting algorithm which will accept memoryless loops and set up the necessary computational procedure to solve them by iteration.

Even if the system being simulated contains no memoryless loops, it is possible that they will be introduced through the modeling of the dynamic subsystems. In the literature, it has been shown [1.11]that numerical stability and efficiency are related to the use of implicit numerical techniques. It will be seen that if subsystems are modeled using implicit techniques, memoryless loops result. Thus, a strong motivation is provided for the development of techniques to deal effectively with the equations of memoryless loops.

The numerical techniques referred to above lie in the domain of numerical analysis. Historically, the techniques of numerical analysis were the first to be applied to the discrete modeling of linear systems. After it became common to represent feedback control systems by transfer functions, the Tustin [1.12] and other substitutional techniques for simulation [1.13-1.14] were developed. Other methods followed [1.15-1.17]. Simultaneous with this work in simulation, many parallel concepts were developed in digital filtering [1.18]. Of particular interest in this respect is a paper by Steiglitz [1.19], in which an isomorphism between the discrete and continuous signal spaces leads to an identification of a discrete modeling method for continuous linear systems. In this thesis, the discrete modeling of systems will be approached from the point of view of numerical stability and efficiency, and the relationship of the techniques developed here to the work done in numerical analysis, digital filtering and simulation will be discussed.

1.3 Summary of the Thesis

In Chapter II, a general model of a system as an interconnection of individual subsystems is given as well as a more precise definition of what is meant by simulation. Two approaches to the simulation of continuous systems are presented, and the modeling of linear and non-linear subsystems is discussed. Some limitations of conventional numerical methods when applied to stiff systems are shown. Dahlquist's [1.11] definition of A-stability is given, and the implications of using A-stable methods are developed.

The concept of A-stability is extended in Chapter III to the discrete modeling of linear systems, and a new technique for applying the bilinear transformation to irrational transfer functions is developed.

A sorting algorithm for general simulation use is developed in Chapter IV, which sets up the computational sequence for solution of the problem by iteration. A presentation of iteration procedures for solution of nonlinear equations is also given.

REFERENCES

[1.1] R. C. Selfridge, "Coding a General Purpose Digital Computer to Operate as a Differential Analyzer," Proc. Western Joint Computer Conference (IRE) 1955, pp. 82-84.

[1.2] R. D. Brennan and R. N. Linebarger, "A Survey of Digital Simulation: Digital Analog Simulator Programs," Simulation 3 no. 6, pp. 22-36 (December 1964).

[1.3] J. J. Clancy and M. S. Fineberg, "Digital Simulation Languages: a Critique and a Guide," AFIPS Conference Proc. 27 Part I, 1965, Spartan Books, Washington, D. C., pp. 23-36.

[1.4] J. C. Strauss, "Digital Simulation of Continuous Dynamic Systems: an Overview," AFIPS Conference Proc. 33 FJCC 1968, Spartan Books, Washington, D. C., pp. 339-344.

[1.5] M. L. Stein and J. Rose, "Changing From Analog to Digital Programming by Digital Techniques," J. ACM 7 no. 1 (January 1960).

[1.6] R. T. Harnett and F. J. Sanson, "MIDAS Programming Guide," Rept. No. SEG-TDR-64-1, Wright-Patterson AFB, Ohio (January 1964).

[1.7] F. J. Sanson and H. E. Petersen, "MIMIC-Digital Simulator Program," SESCA Internal Memo 65-12, Wright-Patterson AFB, Ohio (May, 1965).

[1.8] R. D. Brennan and H. Sano, "PACTOLUS", AFIPS Conference Proc. FJCC 26, 1964, Spartan Books, Washington, D.C.

[1.9] W. M. Syn and R. L. Linebarger, "DSL/90-A Digital Simulation Program for Continuous System Modeling," AFIPS Conference Proc. SJCC 1966, Spartan Books, Washington, D.C.

[1.10] System/360 Continuous System Modeling Program-User's Manual (H20-0367) IBM Corporate Data Processing Division, 112 East Post Road, White Plains, N. Y.

[1.11] G. G. Dahlquist, "A Special Stability Problem for Linear Multistep Methods," BIT 3 no. 1, pp. 27-43 (1963).

[1.12] A. Tustin, "A Method of Analyzing the Behavior of Linear Systems in Terms of Time Series," J.IEE 94 pt. II-A, pp. 130-142, (May, 1947).

[1.13] A. Madwed, "Number Series Method for Solving Linear and Non-linear Differential Equations," Rept. No. 64445-T-26, Instrumentation Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., (April, 1950).

[1.14] R. Boxer and S. Thaler," A Simplified Method for Solving Linear and Nonlinear Systems," Proc. IRE 44, no. 1, pp. 89-101, (January, 1955).

[1.15] M. E. Fowler, "A New Numerical Method for Simulation," Simulation 4, no. 5, pp. 324-330 (May, 1965).

[1.16] J. S. Rosko, "Improved Techniques for Digital Modeling and Simulation of Nonlinear Systems," AFIPS Conf. Proc. SJCC 1968, Spartan Books, Washington, D. C., pp. 473-481.

[1.17] A. P. Sage and S. L. Smith, "Real-time Digital Simulation for Systems Control," Proc. IEEE 54, no. 12, pp. 1802-1812 (December, 1966).

[1.18] F. F. Kuo and J. F. Kaiser, eds., System Analysis by Digital Computer, John Wiley, New York, N. Y., (1966), Chapter 7.

[1.19] K. Steiglitz, "The Equivalence of Digital and Analog Signal Processing," Information and Control 8, 455-467, (1965).

CHAPTER II

INTERCONNECTED SYSTEMS

2.1 Introduction

In this Chapter, two basic methods for the simulation of continuous systems are given. In one approach, the overall system modeling (OSM) method, the continuous input-output-state relations of the individual subsystems are combined to obtain one relation for the overall system. This combined relation is then digitally simulated. The limitations of this approach, when applied to subsystems of many different mathematical types, is shown.

A second approach, the individual subsystem modeling (ISM) method, overcomes these limitations. The ISM method chooses a discrete model for each individual subsystem and interconnects these models to produce the overall simulation. Techniques for modeling individual subsystems are classified here. Most importantly, the stability and efficiency of the simulation for the different classes of modeling methods are discussed. Of particular interest is the use of A-stable techniques when dealing with stiff systems, i.e., systems with widely varied time constants. It should be noted that, for the OSM method, once the combined continuous system relation has been obtained, the completion of the method can be considered an application of the ISM method to a system containing only one subsystem. Hence, the discussion of stability and efficiency for the ISM method is relevant to the OSM method as well.

2.2 System Description

In order to define exactly what is meant by a digital simulation, a mathematical model of a system as an interconnection of subsystems must be given. A collection of N subsystems is considered, each defined by an input-output-state relation

$$y_i(t) = A_i[x_i(t_o),\ v_i, t]$$

where $y_i(t)$ is the vector output of the i^{th} subsystem, $x_i(t_o)$ is the initial state and v_i is the vector input. A_i is a function of the initial state $x_i(t_o)$ and a functional of v_i and t on the interval $[t_o,t]$. Only causal subsystems will be considered, i.e., for any two input vectors v_i^1 and v_i^2, if

$$v_i^1 = v_i^2 \quad \text{for } t_o \leqq t \leqq t_1$$

then causality implies

$$A_i[x_i(t_o),v_i^1,t] = A_i[x_i(t_o),v_i^2,t] \qquad t_o \leqq t \leqq t_1$$

for all t_o, t_1 and $x_i(t_o)$.

An interconnected system formed from these subsystems can be modeled as shown in Fig. 2.1. Letting

$$\underline{v}(t) = \mathrm{col}[v_1(t), v_2(t), \ldots, v_N(t)]$$

and defining $\underline{y}(t)$ and $\underline{x}(t)$ similarly as the column vectors of outputs and states, an overall relationship for the interconnected system may be written as

$$\underline{v}(t) = M\,\underline{y}(t) + P\,\underline{u}(t) \tag{2.1}$$

where $\underline{u}(t)$ is a vector of external inputs. For simplicity, it will be assumed, with no loss of generality, that each component of a subsystem's input vector is connected either to one component of another subsystem's output vector or to one of the external inputs. In equation (2.1), this means that each row of the combined matrix [M P] has one and only one non-zero entry. The input-output-state relationships for all of the subsystems can be written collectively as

$$\underline{y}(t) = A[\underline{x}(t_o), \underline{v}, t] \tag{2.2}$$

where

$$A[\underline{x}(t_o), \underline{v}, t] = \begin{bmatrix} A_1[x_1(t_o), v_1, t] \\ A_2[x_2(t_o), v_2, t \\ \vdots \\ A_N[x_N(t_o), v_N, t] \end{bmatrix}$$

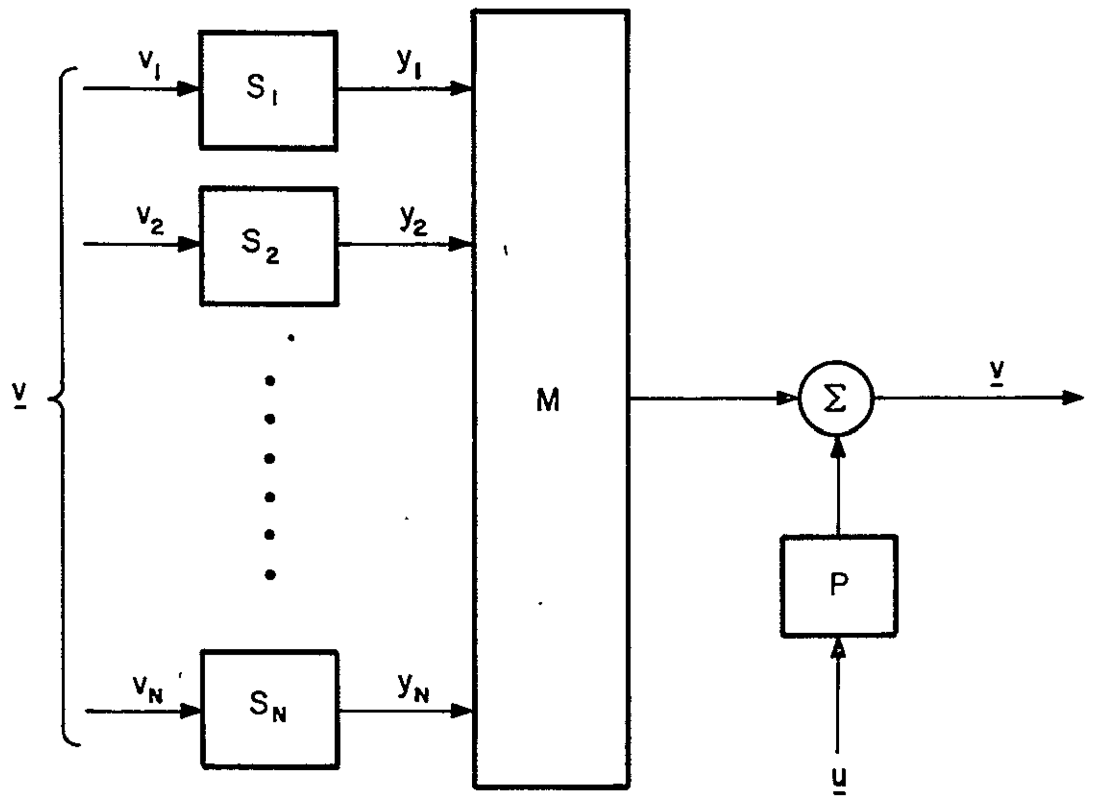


FIG. 2.1 INTERCONNECTION OF SUBSYSTEMS

If equation (2.2) is substituted into equation (2.1), we have

$$\underline{v}(t) = MA[\underline{x}(t_o),\underline{v},t] + Pu(t) \tag{2.3}$$

Since u(t) represents known external inputs, a solution of the system equations is the calculation of $\underline{v}$ in equation (2.3) and thus the determination of $\underline{y}(t)$ through equation (2.2).

A digital simulation of a system of the form shown in Fig. 2.1 is a numerical approximation to the solution of equation (2.3) at a sequence of discrete time points $t_n = t_o + \sum_{j=1}^{n} h_j$. The step size, h_j, may be fixed or variable depending on the type of numerical technique used. Approximations $\underline{y}_n$ and/or $\underline{v}_n$ are desired such that

$$\underline{y}_n \approx \underline{y}(t_n)$$

$$\underline{v}_n \approx \underline{v}(t_n)$$

2.3 The Overall System Modeling Method

In the OSM method of digital simulation, the individual subsystem input-output-state relations are combined to obtain a single relation for the overall system. A discrete model is then chosen for this combined relation to produce the simulation outputs. If each subsystem is described by a set of normal form differential equations or by an equivalent block diagram, then each input-output-state relation is of the form

$$\begin{aligned} \dot{x}_i(t) &= f_i[x_i(t),v_i(t),t] \\ y_i(t) &= g_i[x_i(t),v_i(t),t] \end{aligned} \qquad i = 1,2, \ldots N \tag{2.4}$$

If the output equation in (2.4) contains $v_i(t)$ as an argument, as shown, this represents an instantaneous feed-forward of information from input to output. If equations (2.4) are combined into a single vector equation, there results

$$\begin{aligned} \dot{\underline{x}}(t) &= F[\underline{x}(t),\underline{v}(t),t] \\ \underline{y}(t) &= G[\underline{x}(t),\underline{v}(t),t] \end{aligned} \tag{2.5}$$

where

$$F[\underline{x}(t),\underline{v}(t),t] = \begin{bmatrix} f_1[x_1(t),v_1(t),t] \\ f_2[x_2(t),v_2(t),t] \\ \vdots \\ f_N[x_N(t),v_N(t),t] \end{bmatrix} \text{ and } G[\underline{x}(t),\underline{v}(t),t] = \begin{bmatrix} g_1[x_1(t),v_1(t),t] \\ g_2[x_2(t),v_2(t),t] \\ \vdots \\ g_N[x_N(t),v_N(t),t] \end{bmatrix}$$

Substituting the combined output relationship of (2.5) into (2.1) yields

$$\underline{v}(t) = M\, G[\underline{x}(t),\underline{v}(t),t] + P\, \underline{u}(t) \tag{2.6}$$

Equation (2.6) will generally be implicit in $\underline{v}(t)$ because of the instantaneous connections from inputs to outputs in the individual subsystems. Equations (2.5) and (2.6) are a canonical form for the overall system. Equation (2.5) represents N coupled differential equations and equation (2.6) are N auxiliary constraint relations. The implications of this canonical form for computer-aided analysis of nonlinear networks is discussed by Stern [2.1].

If equation (2.6) is implicit, then any of the large simulation programs will have difficulty. Implicit equations represent memoryless loops in the original system, and of the more advanced simulation languages, only a few will attempt to solve these equations. If the loops are first found by the user, and if there are no loops imbedded within one another (equations must be uncoupled), then some of the programs will attempt solution by simple iteration procedure which converges only under strict constraints on the form of the functions involved. The sorting and iteration procedures developed in Chapter IV provide the necessary methods for solving equation (2.6) for very general systems.

To continue with the OSM method, if the simulation is limited to subsystems with no direct connections from input to output, then equation (2.6) specializes to

$$\underline{v}(t) = M\ G[\underline{x}(t),t] + P\ \underline{u}(t)$$

and this may be substituted into equations (2.5) to give

$$\begin{aligned} \dot{\underline{x}}(t) &= F[\underline{x}(t), M\, G[\underline{x}(t),t] + P\, \underline{u}(t),t] \\ \underline{y}(t) &= G[\underline{x}(t), M\, G[\underline{x}(t),t] + P\, \underline{u}(t),t] \quad . \end{aligned} \qquad (2.7)$$

This is the normal form representation for the overall system. Any standard integration technique, e.g., Runge-Kutta, may now be applied to this combined set of equations. The available simulation languages [see Refs. [1.2-1.10] of Chapter I] use essentially this approach, although equation (2.7) is never explicitly generated by the program. Instead, the program determines the proper computational sequence (sorting) so that evaluation of the subsystems in the prescribed order yields the necessary derivative values.

While <u>any</u> integration technique can be applied to equation (2.7), the available simulation languages are restricted to the use of explicit methods, i.e., methods which use only past values of input, output, and state to calculate the present output. The reason for this is that the sorting algorithms assume that integrator outputs are known quantities and thus are available for evaluation of derivatives. For implicit integration techniques, this is not true, because implicit equations must be solved at each time point to obtain the output or state. The limitations of explicit integration techniques, and hence of presently available simulation languages will be shown in the next section.

If all the individual subsystems are described by normal form differential equations, then the OSM method can be used. If implicit equations of the form (2.6) are present, or if it is desired to use an implicit integration scheme, then techniques must be developed to solve the resultant implicit equations. The sorting and iteration procedures presented in Chapter IV are designed for this purpose.

If, however, any subsystem is not in normal form, for example, if only its impulse response is given, then it may not be possible to use the OSM method, because a combined relation for the overall system may be difficult or impossible to obtain. In this case, another approach must be taken. When the subsystems are of different mathematical types, it is preferable to develop a discrete model for each of the individual subsystems and then to combine these models to obtain the overall simulation. This method, called the individual subsystem modeling (ISM) method, is described in the next section.

2.4 The Individual Subsystem Modeling Method

The ISM method for digital simulation of continuous systems, as indicated in the previous section, is first to approximate each subsystem by a discrete model, and then to interconnect these individual models in order to obtain a simulation of the overall system. In this section, techniques applicable to the modeling of general linear and nonlinear systems will be discussed. A detailed development of discrete models for linear systems will be left for Chapter III.

The subsystem easiest to model is a memoryless subsystem described by an input-output relation of the form

$$y(t) = g(v(t),t)$$

This equation is carried over into a discrete model of the form

$$y_n = g(v_n,t_n) \tag{2.8}$$

If all subsystems are memoryless, e.g., resistive electrical networks, then when the individual models are interconnected, the discrete version of equation (2.3) for the overall system becomes

$$\underline{v}_n = M\, G(\underline{v}_n,t_n) + P\, \underline{u}_n \tag{2.9}$$

where

$$G(\underline{v}_n,t_n) = \begin{bmatrix} g_1(v_{1,n};t_n) \\ g_2(v_{2,n};t_n) \\ \vdots \\ g_N(v_{N,n};t_n) \end{bmatrix}$$

A simulation requires the solution of the N coupled implicit equations in (2.9) at each time point, t_n. This set of equations is usually sparse: therefore, in order to calculate the solution efficiently, advantage should be taken of the structure of the system [2.2]. The sorting and iteration procedures of Chapter IV are designed for this purpose. The modeling of memory subsystems, including all dynamic systems, is an area of wide interest and varied approach. It is possible, however, to form some classifications of the available techniques which help in determining the nature of the interconnection process once the individual models have been chosen.

The general form for a discrete model of the input-output-state relation of a subsystem can be written either as

$$y_n = p(y_o,\ldots,y_n;v_o,\ldots,v_n;t_o,\ldots,t_n) \tag{2.10}$$

or as

$$x_n = q(x_o,\ldots,x_n;v_o,\ldots,v_n;t_o,\ldots,t_n) \tag{2.11a}$$

$$y_n = r(x_o,\ldots,x_n;v_o,\ldots,v_n;t_o,\ldots,t_n) \tag{2.11b}$$

where, as in Section 2.2, y_k is the output vector, x_k the state vector and v_k the input vector at time t_k.

Normally only a few of the arguments shown in the equations are actually present. The classification of the discrete models is determined by the presence or absence of certain arguments as set forth in the following definitions.

<u>Definition 2.1</u>: A discrete model of the form (2.10) which contains y_n as an argument, or a discrete model of the form (2.11a) which contains x_n as an argument, is called <u>internally implicit</u>.

The presence of y_n as an argument in (2.10) or of x_n in (2.11a) indicates that the right hand side cannot be explicitly evaluated at time t_n. The word "internally" is used in the definition to distinguish an equation of the form (2.10) or (2.11a), which must be solved in order to update an individual subsystem, from other implicit relations which may arise when all the individual models are interconnected. Subsystems of the following type generate these equations for the overall system.

<u>Definition 2.2:</u> A discrete model is called externally memoryless if it is of the form (2.10) and contains v_n as an argument or if it is of the form (2.11) and either:

i) contains v_n as an argument in (2.11b) or

ii) contains v_n and x_n as arguments in (2.11a) and (2.11b) respectively.

This terminology results from the following point of view: If the subsystem is considered to store all necessary past values of input, output, or state, then the appearance of v_n in equation (2.10) or (2.11) indicates that the output at time t_n is directly dependent on the input at time t_n, which is similar to the discrete model of a memoryless subsystem, equation (2.8).

To illustrate these definitions, consider the following example:

Example 2.1: A set of normal form differential equations of the form

$$\dot{x} = f(x,v,t)$$
$$y = g(x,t)$$

can be approximated by many techniques. Euler integration,

$$x_n = x_{n-1} + hf(x_{n-1},v_{n-1},t_{n-1})$$
$$y_n = g(x_n,t_n)$$

is neither internally implicit nor externally memoryless. Backward Euler integration,

$$x_n = x_{n-1} + hf(x_n,v_n,t_n)$$
$$y_n = g(x_n,t_n)$$

however is both: internally implicit because x_n appears on both sides of the first equation, and externally memoryless by Definition 2.2ii. The second order Runge-Kutta formula

$$x_n^* = x_{n-1} + hf(x_{n-1}, v_{n-1}, t_{n-1})$$

$$x_n = x_{n-1} + \frac{h}{2}[f(x_{n-1}, v_{n-1}, t_{n-1}) + f(x_n^*, v_n, t_n)]$$

$$y_n = g(x_n, t_n)$$

is also externally memoryless, but is not internally implicit. An example of the fourth possibility, internally implicit but not externally memoryless, is not easily given for this example, since for most standard integration methods, the appearance of x_n is usually linked with the appearance of v_n. For general subsystems, however, the two definitions are independent.

At the beginning of this section, it was stated that each subsystem would be approximated by a discrete model, and the individual models then interconnected to produce the overall simulation. In the original physical system, the connected variables are equal for all time. A discrete model of a continuous subsystem, as we have defined it in this section, produces output values, y_n, only at the discrete time points of the simulation, t_n, and values at any other time points are not available. Accordingly, when an output of one subsystem is connected to the input of another, equality is defined only at the points t_n.

To see how this discretization may affect the choice of numerical techniques used, consider a 4th order Runge-Kutta integration method for a set of normal form equations of the type considered in Example 2.1. This method requires evaluation of the quantity

$$hf\left(x_{n-1} + \frac{K_1}{2} , v_{n-1/2} , t_n - \frac{h}{2}\right)$$

where

$$v_{n-1/2} = v\left(t_n - \frac{h}{2}\right)$$

and K_1 has been previously evaluated. If the subsystem were to be used alone, with $v(t)$ an external continuous input, then $v(t_n - h/2)$ would be available exactly. If, however, the input to this subsystem is the output of another subsystem, then only values v_{n-1} and v_n are available, while the intermediate value $v_{n-1/2}$ is not. Of course interpolation may be used to approximate the desired value, but in general this causes a loss in accuracy which negates the purpose of using a high-order integration technique. This discussion leads to the following definition.

Definition 2.3: A discrete model is admissible for the ISM method if it is of the form (2.10) or (2.11), i.e., it only requires inputs at the discrete time points of the simulation.

In all that follows, only admissible models for the ISM method will be considered.

In Example 2.1, it was seen that the choice of the numerical technique to be used determines whether or not a subsystem is modeled as externally memoryless. The effect upon the overall simulation of modeling all memory subsystems as externally memoryless will now be determined.

The notation of equation (2.10) or (2.11) for an externally memoryless subsystem can be simplified if all arguments for past time points are suppressed. It will also be assumed for the moment that the subsystem is not internally implicit. With these conventions understood, the input-output-state equation of an externally memoryless subsystem can be written exactly as that of an ordinary memoryless subsystem

$$y_n = r(v_n, t_n) \ .$$

Equation (2.1) for the overall simulation then becomes

$$\underline{v}_n = M\, R\, (\underline{v}_n, t_n) + P\, \underline{u}_n \tag{2.12}$$

where all variables are defined as in (2.7). Again, as in the case of a system composed only of memoryless elements, a sparse system of possibly nonlinear implicit equations must be solved. Had the dynamic subsystems not been modeled as externally memoryless, then this would have been the case only if there were loops of ordinary memoryless elements.

ıııe special case where one or more subsystems are internally implicit will now be considered. An internally implicit subsystem has, by definition, an implicit equation associated with it. In order to update the subsystem, this equation must be solved. Two very different computational approaches to the solution of this equation are possible. The first approach assumes that this equation is solved internal to the subsystem, by its own solution technique, e.g., iteration . From an input output point of view, the implicit equation does not exist, so that the previous discussion of externally memoryless subsystems still applies, and equations of the form (2.12) result for the overall simulation. This approach ignores the interconnection of the subsystem with other subsystems as far as its internal implicit equation is concerned.

A more efficient approach, however, is to consider all the implicit equations of the entire simulation simultaneously. These include both the equations of internally implicit subsystems and the equations which result when externally memoryless subsystems are interconnected. A detailed description of exactly how this is accomplished is given in Chapter IV.

The remainder of this section will show why it may be desirable to model dynamic subsystems as externally memoryless even though the added complexity of implicit equations results. The discussion first will be limited to subsystems described in normal form. It will be shown that for these subsystems the motivation for externally memoryless modeling stems from a consideration of the stability and efficiency of numerical integration methods. For simplicity of notation, the system equations will be assumed to be in the form

$$\dot{x} = f(x,t), \qquad x(0) = x_o. \tag{2.13}$$

The input to the subsystem, v, has not been shown, since, when modeling an individual subsystem, the input is considered a known function of time. The output equation is also not shown, since it does not enter into the choice or use of an integration method for the state equation. The eigenvalues of the Jacobian, $J = \frac{\partial f}{\partial x}$, are denoted $\lambda_1, \lambda_2, \ldots, \lambda_N$ (arranged in order of increasing magnitude) and the linearized system is assumed to be asymptotically stable, i.e., $\mathrm{Re}(\lambda_i) < 0$, $i = 1,\ldots,N$.

In many applications it often happens that equation (2.13) is stiff, i.e., the stiffness ratio, $\rho \triangleq |\lambda_N|/|\lambda_1| \gg 1$. For these systems, the solution is smooth after a transient period during which there is a rapid variation. Examples abound in circuit theory, chemical kinetics, nuclear reactor calculation, and in many other fields. The majority of classical integration methods are unsatisfactory for stiff systems because the fast component of the solution continues to affect the numerical solution long after its effect in the physical system has died out.

For most single-step methods, e.g., Runge-Kutta, the step size is limited by the largest eigenvalue of the Jacobian, J, i.e.,

$$\max_i |h \lambda_i| < d \tag{2.14}$$

where d is a constant that is characteristic for the method. If this condition is not satisfied, then the solution will in general be numerically unstable, and hence useless.

It is reasonable to assume that the solution interval, τ, is proportional (by some factor c such as 3 or 10) to the inverse of the magnitude of the smallest eigenvalue,

$$\tau = c\left[\min_i |\lambda_i|\right]^{-1} . \tag{2.15}$$

A lower bound on the number of steps necessary to compute the solution over the interval τ is τ/h where h satisfies (2.14), and thus from (2.15), $\tau/h \sim \rho$. If the spread of eigenvalues is large, then the number of steps required will be excessive. For example, this bound can easily become arbitrarily large if some parameter, say a capacitor value in a linear R-C electrical network, becomes arbitrarily small. Methods having this limitation are called step length limited.

Explicit multi-step integration methods have much the same problem when used to integrate stiff systems of differential equations.

The following example shows these difficulties for some particular integration methods.

Example 2.2*: Consider the linear system of differential equations,

$$\dot{x} = Ax, \qquad x(0) = x_o$$

with distinct real eigenvalues, $0 > \lambda_1 > \lambda_2 > \dots > \lambda_N$.

Since the analytic solution is

$$x(nh) = \left(e^{hA}\right)^n x_o = e^{hA}\, x((n-1)h)$$

and all eigenvalues are negative, the least that should be required is that

$$\lim_{n \to \infty} x_n = 0 \ .$$

For the explicit Euler, Heun or usual 4th order Runge-Kutta schemes, the approximate solution is

$$x_n = [M(hA)]^n\, x_o \tag{2.16}$$

where

$$M(q) = \begin{array}{ll} 1+q & \text{Euler} \\ 1+q+q^2/2! & \text{Heun} \\ 1+q+q^2/2!+q^3/3!+q^4/4! & \text{Runge-Kutta} \end{array}$$

*This example is taken from a short paper, "Stiff Systems of Differential Equations," by J. S. Rosenbaum, prepared for a course in nonlinear systems at Columbia University, (Fall, 1968)

Since the eigenvalues of A are distinct, there exists a matrix P such that

$$P^{-1} AP = \text{diag}[\lambda_1, \ldots, \lambda_n] = \Lambda .$$

Therefore, the difference equation (2.16) becomes

$$x_n = P^{-1}[M(h\Lambda)]^n Px_o$$

which implies that

$$\lim_{n \to \infty} x_n = o \qquad \text{iff} \quad |M(h\lambda_i)| < 1 \quad i = 1, \ldots, N .$$

In particular, for Euler's method, there results

$$x_n = x_{n-1} + h\, f(x_{n-1},t)$$

or

$$x_n = P^{-1}(1 + h\,\Lambda)\, P\, x_{n-1}$$

therefore, stability requires that

$$|1 + h\,\lambda_i| < 1 \quad i = 1, \ldots, N$$

or

$$|h\,\lambda_i| < 2$$

For Heun's method the stability condition is that $|h\,\lambda_i| < 2$ and for the Runge-Kutta method the stability condition is $|h\,\lambda_i| < 2.78$.

To demonstrate the difficulties that these conditions imply for stiff systems, consider the following second order example,

$$\dot{v} = Dv + C \;, \qquad v(o) = v_o$$

where

$$D = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

For this equation, $v_1(t) = v_2(t) = 2(1-e^{-t})$ and the stability conditions for Euler $(h < 2)$, Heun $(h < 2)$ and Runge-Kutta $(h < 2.78)$ place reasonable limits on h. However, consider the system

$$\dot{w} = A\,w + C \qquad w(o) = \begin{bmatrix} -.1 \\ 0 \end{bmatrix}$$

where

$$A = \begin{bmatrix} -500.5 & 499.5 \\ 499.5 & -500.5 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

for which the solution is

$$w_1 = 2(1 - e^{-t}) - .1\, e^{-1000t}$$
$$w_2 = 2(1 - e^{-t}) - .1\, e^{-1000t}$$

For $t > .02$, $.1\ e^{-1000t} < 2.5 \times 10^{-10}$, or $v \approx w$. However, the eigen-values of A are -1 and -1000, which requires that h be 1000 times smaller for the w equation than for the v equation.

In the previous discussion, it has been seen that the efficiency of numerical integration methods when applied to stiff systems is closely related to the stability of the numerical solution. In many applications, it is stability rather than accuracy considerations which necessitate the use of a very small step size. Dahlquist [2.3] has given a definition of stability for numerical methods which leads to the identification of integration methods suitable for stiff systems.

Definition 2.4: A numerical integration method is called A-stable if all solutions tend to zero as $n \to \infty$ when the method is applied with fixed positive h to any differential equation of the form

$$\dot{x} = qx$$

where q is a complex constant with negative real part.

This definition states that for A-stable methods, the transient solution eventually goes to zero, and also that the transient solution does not affect the step size.

It can easily be shown that the usual explicit 4th order Runge-Kutta method is not A-stable, for the Runge Kutta method yields a truncated Taylor series when applied to equation (2.15), i.e., it gives the sequence

$$[1 + qh + (qh)^2/2! + (qh)^3/3! + (qh)^4/4!]^n \qquad n = 0,1,2,\ldots$$

and this does not tend to zero everywhere in the left half plane, $Re(q) < 0$. Runge-Kutta methods are not admissible for the ISM method, but they can be used in the OSM method. If the OSM method is to be used efficiently for stiff systems, then, since explicit Runge-Kutta methods are not A-stable, implicit Runge-Kutta methods must be used [2.4]. Implicit Runge-Kutta techniques give externally memoryless discrete models, because they always use the input at the present time. This, as shown previously, results in implicit equations of the form (2.12), which must be solved at each time point.

A linear multi-step integration method has the form

$$a_k x_n + a_{k-1} x_{n-1} + \ldots + a_o x_{n-k} = h(b_k f_n + \ldots + b_o f_{n-k}) \qquad (2.17)$$

where a_i, b_i are real constants, $i = 0,1,2,\ldots k$, and $f_m = f(x_m,t_m)$. If the vectors $x_0,x_1,\ldots,x_{k-1}$ and $v_0,v_1,\ldots, v_{k-1}$ are given, then $x_k,x_{k+1},\ldots$ are computed recursively through (2.17). If the method is implicit, i.e., $b_k \neq 0$, then equation (2.13) must be solved for x_n at each time point. In the notation of this section, $b_k \neq 0$ also indicates that the technique is externally memoryless. Since only values at the discrete points t_n are required, linear multi-step methods are admissible for the ISM method, and hence the discussion to follow is relevant to both the ISM and OSM methods.

Dahlquist [2.3] proves the following theorem, presented here using the terminology of this Chapter.

<u>Theorem 2.1:</u> An A-stable linear multi-step method must be internally implicit and externally memoryless, i.e., $b_k \neq 0$.

Thus, in order to achieve A-stability of the simulation using linear multi-step methods, externally memoryless models must be used. It will be seen in Chapter III that this same motivation for externally memoryless modeling exists in the simulation of linear rational and irrational transfer functions.

To carry the discussion of linear multi-step methods further, it is necessary to define the order, p, of an approximate method. Assume that the values $x_0, x_1, \ldots, x_n$ are exact, i.e., $x_i = x(t_i)$, and that the value x_{n+1} is calculated using the method in question. Expanding both x_{n+1} and $x(t_{n+1})$ in a Taylor series about the point $t = t_n$, there results

$$x(t_{n+1}) = x(t_n) + h/1!\ x'(t_n) + h^2/2!\ x''(t_n) + \ldots$$

and

$$x_{n+1} = x(t_n) + h/1!\ c_1 + h^2/2!\, c_2 + \ldots$$

The highest exponent of h for which corresponding terms in the two series are equal is called the order, p, of the method, i.e.,

$$c_1 = x'(t_n), \ldots, c_p = x^{(p)}(t_n), \quad \text{but } c_{p+1} \neq x^{(p+1)}(t_n)$$

Dahlquist proves the following theorem.

Theorem 2.2: The order, p, of an A-stable linear multi-step method cannot exceed two. The smallest truncation error is obtained with the trapezoidal rule,

$$x_n = x_{n-1} + h/2(f_{n-1} + f_n)$$

A-stable linear multi-step methods for the solution of stiff systems of differential equations have been discussed in the literature [2.5-2.7] and Liniger and Willoughby [2.5] give several examples of the dramatic increase in efficiency that is possible using these methods.

The previous discussion has shown that externally memoryless modeling is useful in order to achieve an efficient simulation of stiff systems. Another motivation for the use of externally memoryless models exists when the waveforms of the system being simulated have discontinuities or discontinuous derivatives. In this case, the use of the present value of the input will tend to force the solution to react more quickly to the effect of the discontinuity. For example, if the input to a system in normal form (2.13) is a ramp, beginning at $t_n = T > t_o$, then, if an explicit method is used, the next value x_{n+1} is calculated using only past values of input, which are identically zero. An externally memoryless technique, however, uses the value of the input at time t_{n+1}(nonzero), and hence calculates x_{n+1} using the fact that the input has changed.

In the next section, an example is presented which shows the benefits of externally memoryless modeling for systems with discontinuous waveforms. This example consists of a transmission line with nonlinear loads and thus represents two coupled subsystems of different mathematical types. For this reason, it is necessary to use the ISM method.

2.5 Transmission Line Example

In this section, an example is presented which illustrates the use of externally memoryless modeling to achieve accurate simulation for an interconnection of two systems of different mathematical type. The problem considered is shown in Fig. 2.2, and consists of a step current source shunted by a parallel R-C network feeding a lossless transmission line. The transmission line is terminated in a nonlinear load. As suggested by Wing [2.8], we may model this network as two interconnected subsystems, one representing the source and load, and one representing the transmission line. Fig. 2.3 is a block diagram of this representation.

If the transmission line is regarded as a two-port, the terminal relation may be written as

$$i(t) = h(t)*v(t)$$

where $i(t) = \text{col}[i_1(t), i_2(t)]$ and $v(t) = \text{col}[v_1(t), v_2(t)]$, $h(t)$ is the impulse response of the network and * denotes convolution. Thus, for this system, $v(t)$ is regarded as the input and $i(t)$ as the output.

For the lossless transmission line, $h(t)$ is found to be

$$h_{11} = h_{22} = \delta(t) + 2 \sum_{k=1}^{\infty} \delta(t-2k)$$

$$h_{12} = h_{21} = - \sum_{k=1}^{\infty} \delta(t-2k+1)$$

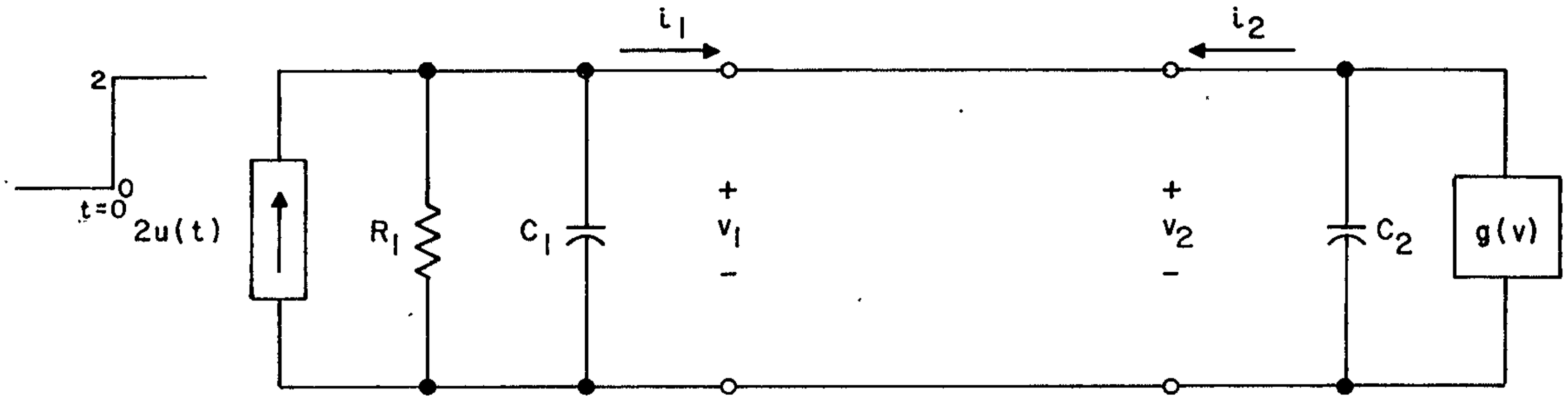


FIG. 2.2 TRANSMISSION LINE EXAMPLE

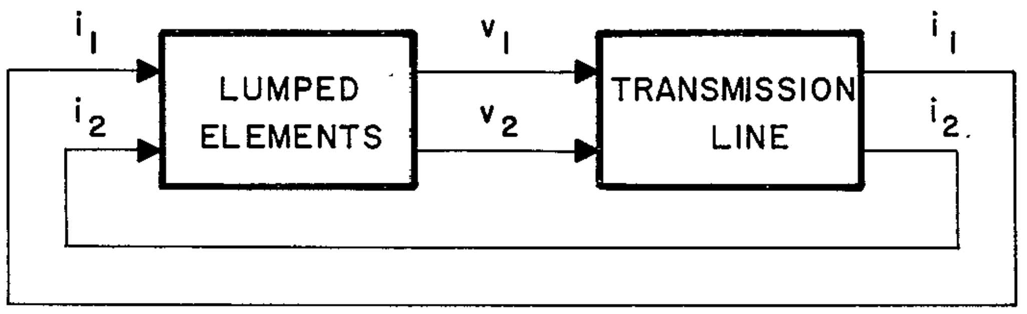


FIG. 2.3 BLOCK DIAGRAM OF TRANSMISSION LINE PROBLEM

Since the impulse response contains only impulses, a numerical approximation to the convolution of h(t) with v(t) becomes a simple weighted sampling of present and past values of v.‡ The impulse at the origin in h_{11} and h_{22} samples the present value, v_n, and hence this model is externally memoryless. If the transmission line were not lossless, a more complex h(t) would result, with a corresponding increase in the complexity of the discrete model. In general, however, the model would still be externally memoryless.

The discrete model of the lossless line can be reduced to

$$i_n = v_n + I_{n-1}$$

where I_{n-1} depends only upon the past values of the input, v. The sampling involved in calculating I_{n-1} at each step occurs because it takes two time units for a signal to travel down the line, be reflected, and return.

The lumped elements will be described by a set of normal form differential equations

$$\dot{v} = f(v,i) \tag{2.18}$$

which for the network given is

$$\begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \end{bmatrix} \quad \begin{bmatrix} (2 - v_1/R_1 - i_1)/C_1 \\ (- g(v_1) - i_2)/C_2 \end{bmatrix}$$

‡ In Chapter III it will be shown that this is not necessarily the best way to approximate h(t), but it will be used here for simplicity.

Here, i is now considered the input, and v the output. The trapezoidal rule will be used as a discrete model for the lumped elements, but modified, as suggested by Liniger and Willoughby [2.5], to include one Newton-Raphson iteration.

The trapezoidal rule applied to equation (2.18) gives

$$v_n = v_{n-1} + \frac{h}{2}[f_{n-1} + f_n]$$

where $f_k = f(v_k, i_k)$.

Letting $z = v_n$, this can be written

$$Q(z) = z - \frac{h}{2}[f_{n-1} + f(z, i_n)] - v_{n-1} = 0$$

and taking one derivative with respect to z,

$$Q'(z) = I - \frac{h}{2} f_z(z, i_n)$$

where $f_z = \dfrac{\partial f(z, i_n)}{\partial z}$.

Taking one Newton-Raphson iteration using z_o as the initial point gives

$$z^1 = z^0 - [Q'(z)]^{-1}\Big|_{z^0} \cdot Q(z^0) .$$

Taking only this first iteration, letting $z_o = v_{n-1}$, there results the explicit integration formula

$$v_n = v_{n-1} + [I - \frac{h}{2} f_v(v_{n-1},i_n)]^{-1} \cdot \frac{h}{2}[f_{n-1} + f(v_{n-1},i_n)] \,. \tag{2.19}$$

This formula is the discrete model for the lumped elements. It should be noted that the model is externally memoryless due to the presence of the input value i_n, but it is no longer internally implicit because the Newton-Raphson iteration has removed v_n as an argument on the right hand side. To simplify notation, equation (2.19) is written symbolically as

$$v_n = F(i_n,i_{n-1},v_{n-1}) \,. \tag{2.20}$$

Since both subsystems are modeled by externally memoryless techniques, it is expected that implicit relationships will exist when the individual models are interconnected. The interconnection has in essence been accomplished already, through the use of identical variable names, i and v, when modeling each subsystem. Therefore, the general equation for interconnected models, equation (2.1), becomes for this example

$$\begin{bmatrix} i_n \\ \\ v_n \end{bmatrix} = \begin{bmatrix} v_n + I_{n-1} \\ \\ F(i_n,i_{n-1},v_{n-1}) \end{bmatrix} \tag{2.21}$$

Solution techniques for the general case of equation (2.1) will be left for Chapter IV, but so as to be able to continue with the example, a simple iteration procedure will be used here. If the second equation in (2.21) is substituted into the first, there results

$$i_n = F(i_n, i_{n-1}, v_{n-1}) + I_{n-1} \ . \tag{2.22}$$

Since i_{n-1}, v_{n-1}, and I_{n-1} are known quantities at time t_n, this equation is of the form

$$x = \varphi(x) \ .$$

An obvious iterative solution technique is

$$x^{k+1} = \varphi(x^k)$$

which will be shown in Chapter IV to converge if

$$\left\| \frac{\partial \varphi}{\partial x} \right\| \leqq c < 1$$

where $\|\cdot\|$ denotes the matrix norm $\|A\| \triangleq \max_i \sum_j |a_{ij}|$.

This method is used to solve equation (2.22), resulting in

$$i_n^{k+1} = F(i_n^k \ , \ i_{n-1} \ , \ v_{n-1}) + I_{n-1} \cdot$$

At each time point t_n, an initial estimate i_n^o must be chosen. The use of the previous value, $i_n^o = i_{n-1}$ was first tried, but it was found that if a linear prediction was made, the total number of iterations necessary for convergence was reduced. Hence,

$$i_n^o = 2i_{n-1} - i_{n-2}$$

Each time a reflected wave reaches one end of the line, discontinuities in the slopes of the variables occur. The use of externally memoryless modeling for both the transmission line and the lumped elements helps in handing these waveforms, as explained in the previous section. During calculation, it is seen that the iteration count in solving equation (2.22) is higher for the time points where the discontinuities occur than for time points where the waveforms are well behaved.

The transmission line example discussed above was run on a time-shared SDS/940 computer for both linear and nonlinear loads. For the first simulation, $R_1 = 1K$, $C_1 = 1pf$, $R_2 = 2K$, and $C_2 = 1pf$. Fig. 2.4 is a plot of v_1, v_2, i_1 and i_2 for a step size of 1/32 ns. The iteration at each step was stopped if the difference between the predicted and calculated values was less than an iteration tolerance, ϵ.

Since with linear loads the entire problem is linear, it is possible to compute the exact values using Laplace transform techniques and working with the incident and reflected waves. These values agree very closely with the computed results, the error being too small to show in Fig. 2.4.

To determine the effect upon solution accuracy of choosing different values for the iteration tolerance, ϵ, and the step-size, h, several runs for different values were made. These results are summarized in Table 2.1. It is seen that reducing ϵ beyond a certain value simply increases the iteration count with no increase in accuracy. As would be expected, this value of ϵ is approximately the same as the truncation errors introduced in the numerical approximation techniques.

The linear resistor R_2 is now replaced by a diode modeled by the diode equation

$$i = I_o(e^{v/v_T} - 1)$$

with I_o = .005 ma and v_T = .026 v. A plot of the results for h = 1/64 ns appears in Fig. 2.5 and Fig. 2.6. It is not possible to obtain analytic results, but the computed results approach the correct steady-state values, and several runs with smaller step-sizes indicate that these results are approximately of the same accuracy as achieved for the linear example.

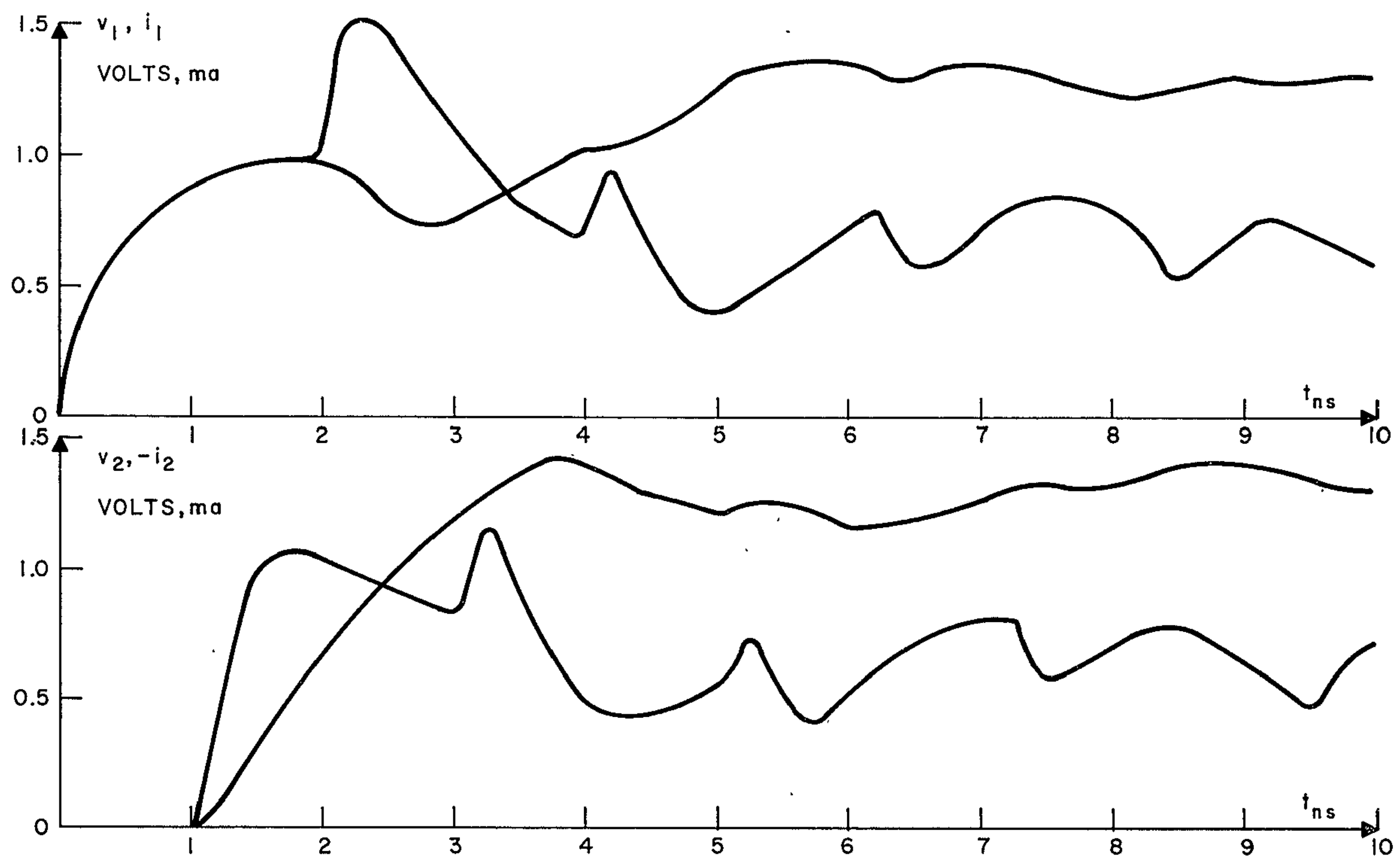


FIG. 2.4 RESULTS FOR TRANSMISSION LINE PROBLEM WITH LINEAR LOADS

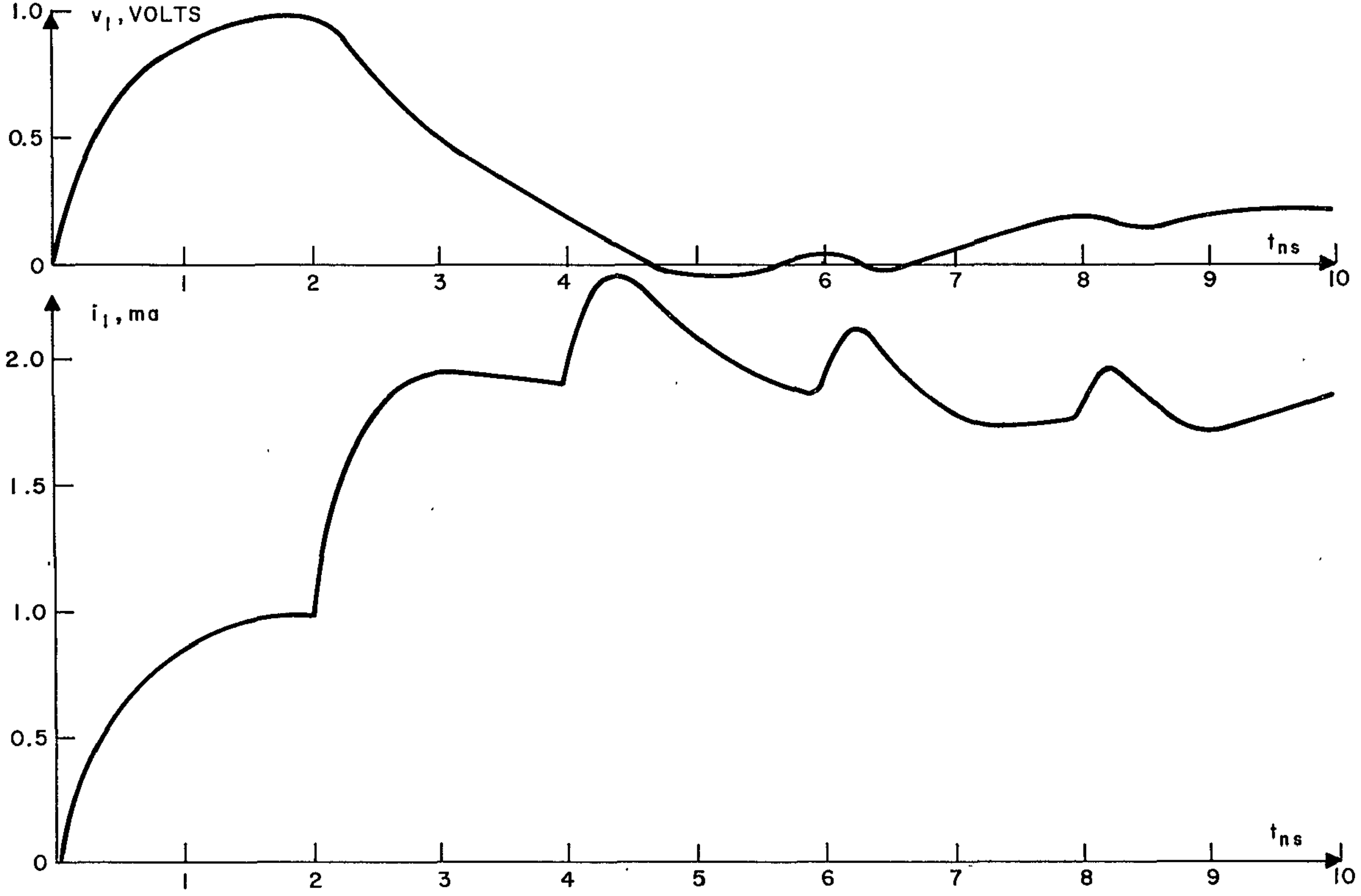


FIG. 2.5 RESULTS FOR TRANSMISSION LINE PROBLEM WITH DIODE LOAD (v_1, i_1)

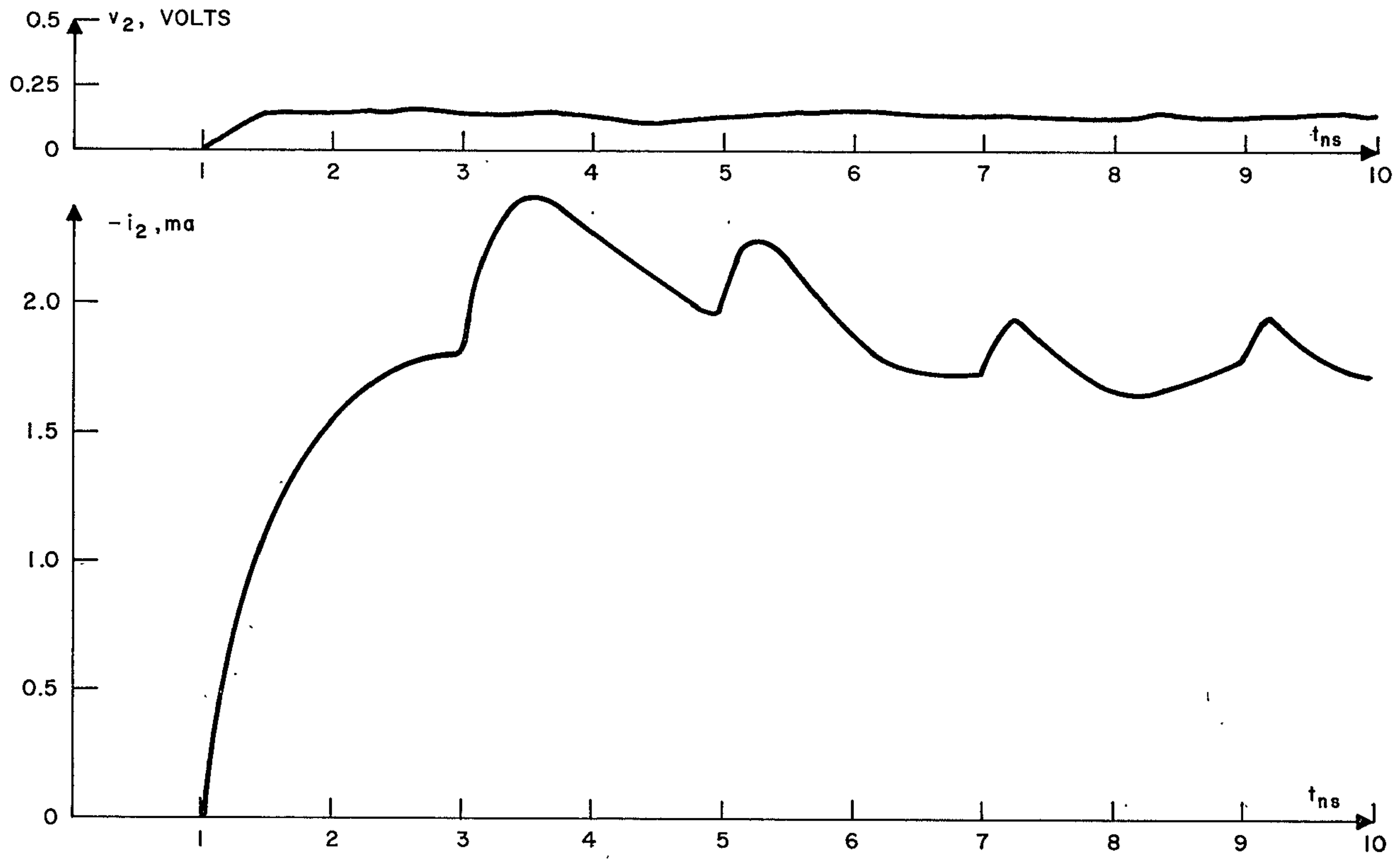


FIG. 2.6 RESULTS FOR TRANSMISSION LINE PROBLEM WITH DIODE LOAD (v_2, i_2)

TABLE 2.1

Statistics For Transmission Line Example With Linear Loads.

Step Size (ns)	Iteration Tolerance	Maximum Error ($\times 10^{-3}$)	Average Error ($\times 10^{-3}$)	Iterations/ Step
1/32	.01	4.7	1.51	1.26
1/32	.005	2.1	.88	1.45
1/32	.001	1.7	.32	1.89
1/32	.0005	1.6	.34	2.01
1/32	.0001	1.4	.32	2.34
1/32	K set = 2	1.5	.33	2.00
1/64	.005	2.1	.599	1.10
1/64	.001	.47	.172	1.51
1/64	.0005	.42	.091	1.72
1/64	.0001	.38	.081	2.00
1/64	5×10^{-5}	.37	.081	2.07
1/64	1×10^{-5}	.35	.076	2.33
1/64	K set = 2	.37	.079	2.00
1/128	.0005	.260	.077	1.24
1/128	.001	.099	.021	1.77
1/128	5×10^{-5}	.094	.018	1.89
1/128	1×10^{-5}	.092	.019	2.04
1/128	5×10^{-6}	.090	.020	2.10
1/128	K set = 2	.090	.019	2.00

REFERENCES

[2.1] T. E. Stern, "Computer-Aided Analysis of Nonlinear Networks," Columbia University Systems Research Group Tech. Rpt. 107, Department of Electrical Engineering, Columbia University, New York, N. Y. (July, 1968).

[2.2] D. V. Steward, "Partitioning and Tearing Systems of Equations," SIAM J. Numer. Anal. Series B 2 no. 2, pp. 345-365 (1965).

[2.3] G. G. Dahlquist, "A Special Stability Problem for Linear Multistep Methods," BIT 3 no. 1, pp. 27-43 (1963).

[2.4] B. Ehle, "The Application of Implicit Runge-Kutta Processes to Stiff Systems," Dept. of Computer Science, University of Waterloo, Ontario, Canada.

[2.5] W. Liniger and R. A. Willoughby, "Efficient Numerical Integration of Stiff Systems of Ordinary Differential Equations," Res. Rpt. RC-1970, IBM Watson Research Center, Yorktown Heights, New York (Dec. 20, 1967).

[2.6] I. W. Sandburg and H. Shichman, "Numerical Integration of Systems of Stiff Nonlinear Differential Equations," Bell System Tech J. 47 no. 4, pp. 511-527 (April, 1968).

[2.7] M. R. Osborne, "On the Integration of Stiff Systems of Ordinary Differential Equations," Computer Center, Australian National Univ. Canberra, A.C.T. Tech Rpt. No. 24 (Sept., 1968)

[2.8] O. Wing, "Computer Analysis of Distributed Parameter Systems With Nonlinear Boundary Conditions," Proc. of the Hawii International Conf. on System Sciences, Jan. 29-31, 1968, pp. 424-427.

CHAPTER III

MODELING OF LINEAR SYSTEMS

3.1 Introduction

In this Chapter, general methods of A-stable modeling of linear fixed systems are presented. It is shown that A-stable techniques result in externally memoryless models, and hence the question of solution of implicit equations for the overall simulation again arises.

A-stable techniques are applied to the irrational transfer functions that are typical of distributed systems. An example is given in order to illustrate the difficulties which may be encountered when the more conventional approaches to modeling are used.

3.2 A-stable Modeling of Fixed Linear Systems

In Chapter II, a general formulation of a system as an interconnection of subsystems was given, and some general approaches to modeling of the subsystems were discussed. If each subsystem is now considered to be linear and fixed (but not necessarily lumped), then the following problem can be considered. Let the original interconnected system be stable. It is possible that the original system may contain some subsystems which are unstable, even though the overall system is stable. The problem is to determine a class of discrete modeling methods, which, if used to model each individual subsystem, gives A-stability for the overall simulation when the individual models are interconnected. Application of A-stable modeling to the unstable subsystems would generally produce unstable models. What is required here, however, is that the interconnection of all the models, both stable and unstable, produce an A-stable overall model when the original overall system is stable. It will also be required that the model be consistent, i.e., a reasonable approximation of the original system. This problem of A-stable modeling will now be more precisely defined using some basic definitions and theorems (Theorems 3.1 - 3.4) from the theory of linear systems (see e.g., [3.1-2].

<u>Definition 3.1</u>: A linear fixed causal continuous system, S, with input vector u and output vector y is stable if its zero-state response remains bounded for any bounded input.

If F(s) represents the transfer function matrix of S, with input and output defined as above, and if f(t) is the impulse response of S, i.e., f(t) is the inverse Laplace transform of F(s), then the following well known theorem applies.

Theorem 3.1: S is stable if and only if each component of the impulse response, $f_{ij}(t)$ satisfies the condition

$$\int_0^\infty |f_{ij}(t)|dt < \infty \ .$$

It will prove useful later to have a necessary condition for stability in the frequency domain.

Theorem 3.2: A necessary condition for the stability of S is that each component of the transfer function matrix, $F_{ij}(s)$, be finite for $Re(s) \geq 0$.

Proof:

$$F_{ij}(s) = \int_0^\infty f_{ij}(t)e^{-st}\, dt$$

From this, there follows

$$|F_{ij}(s)| \leqq \int_0^\infty |f_{ij}(t)|e^{-\sigma t}\, dt \qquad R_e(s) \geq 0 \, .$$

where $\sigma = \text{Re}(s)$. If $\text{Re}(s) \geq 0$, then $e^{-\sigma t} \leqq 1$, and therefore

$$|F_{ij}(s)| \leqq \int_0^\infty |f_{ij}(t)|\,dt \qquad \text{Re}(s) \geq 0 \,.$$

The right hand side is finite by Theorem 3.1, which completes the proof. Note that this theorem applies to irrational as well as rational transfer functions.

Since an identification must be made between stability of a continuous system and stability of its discrete model, the following definition and theorems for stability of discrete systems are given.

Definition 3.2: A linear fixed causal discrete system, D, with input vector u and output vector y, is stable if its zero-state response remains bounded for any bounded input sequence.

If G(z) represents the transfer function matrix of D, and if g(n) is the response to the input sequence 1,0, ..., i.e., g(n) is the inverse z-transform of G(z), then the following theorem applies.

Theorem 3.3: D is stable if and only if each component of the zero-state response, $g_{ij}(n)$, satisfies the condition

$$\sum_{n=0}^{\infty} |g_{ij}(n)| < \infty \,.$$

It is interesting to note that Theorem 3.1 does not require $|f_{ij}(t)| \to 0$ as $t \to \infty$ but Theorem 3.3 does require $|g_{ij}(n)| \to 0$ as $n \to \infty$.

2

A necessary condition for stability in the frequency domain will be useful later.

Theorem 3.4: A necessary condition for the stability of D is that each component of the transfer function matrix, $G_{ij}(z)$, be finite for $|z| \geq 1$.

The proof follows that of Theorem 3.2 and makes use of Theorem 3.3, and so will not be given here.

A class of discrete models for continuous systems will be defined by a mapping from the s-plane to the z-plane. For each subsystem transfer function $F_i(s)$ let

$$F_i^*(z,h) = F_i(g(z,h))$$

where the mapping $s = g(z,h)$ is chosen so that the resulting discrete model $F_i^*(z,h)$ has the following properties:

(1) Consistency

Let

$$f_i(n,h) = \frac{1}{2\pi j} \oint_C F_i^*(z,h) z^{n-1} \, dz$$

where C is a circle enclosing all the singularities of $F_i^*(z,h)z^{n-1}$, i.e., $f_i(n,h)$ is the inverse z-transform of $F_i^*(z,h)$. The discrete model is then said to be consistent if

$$\lim_{\substack{n\to\infty\\ h\to o\\ nh=t}} \left| \frac{f_i(n,h)}{h} \right| = f_i(t)$$

except possibly at discontinuities.

Rewriting, it is required that

$$\lim_{n\to\infty} \left| \frac{n}{t} f_i\left(n, \frac{t}{n}\right) \right| = f_i(t) .$$

(2) <u>A-Stability</u>

If $F_i(s)$ is a stable transfer function, then $F_i^*(z,h)$ must be stable for all $h \geq 0$, i.e., the discrete model is stable for all positive values of the step-size.

The second property implies the following theorem.

<u>Theorem 3.5</u>: If $s = g(z,h)$ is an A-stable mapping, i.e., it satisfies condition (2) above, then

$$\text{Re}(s) < 0 \quad \text{implies } |z| < 1 .$$

<u>Proof</u>: $F_i(s)$ stable implies $\text{Re}(s) < 0$ if $F_i(s)$ is not finite, from Theorem 3.2. $F_i^*(z,h)$ stable implies $|z| < 1$ if $F_i^*(z,h)$ is not finite from Theorem 3.4. Hence if $s = g(z,h)$ satisfies condition (2), A-stability, then

$$\text{Re}(s) < 0 \text{ implies } |z| < 1 .$$

Thus it is seen that the left half s-plane must map into the interior of the unit circle in the z-plane.

It is obvious that if the same consistent A-stable transformation is made for each subsystem, then the overall model, formed from the interconnection of the individual models, will also be consistent and A-stable. It will be shown later however, that applying different transformations to subsystems, each of which is consistent and A-stable, will not lead in general to a consistent A-stable overall model.

It now remains to determine functions g(z,h) which satisfy both conditions (1) and (2) above, that is, they are both consistent and A-stable. If the search is limited to rational functions of z, then it is possible to derive certain characteristics of the allowable transformations. Consider a k-th order transformation of the form

$$g(z,h) = \frac{1}{h} \cdot \frac{a(z)}{b(z)} = \frac{1}{h} \cdot \frac{a_k z^k + \ldots + a_o}{b_k z^k + \ldots + b_o} \qquad (3.1)$$

where a(z) and b(z) have no common zeros. If the system being modeled is of the form 1/s, then a transformation (3.1) is equivalent to using a linear multi-step integration formula. Dahlquist [3.3] has worked with A-stable integration formulas of this type, and it is possible to follow a similar development here. The following theorem establishes necessary and sufficient conditions for a transformation of the form (3.1) to be A-stable.

Theorem 3.6: A k-th order transformation $s = g(z,h)$ of the form (3.1) is A-stable if and only if $g(z) = a(z)/hb(z)$ is analytic for $|z| \geq 1$.

Proof: If $s = g(z,h)$ is A-stable, from Theorem 3.5,

$$\text{Re}(s) < 0 \quad \text{implies} \quad |z| < 1$$

or

$$|z| \geq 1 \quad \text{implies} \quad \text{Re}(s) \geq 0 \ .$$

For a transformation of the form (3.1),

$$s = \frac{a(z)}{h\ b(z)} \quad \text{if } b(z) \neq 0 \ .$$

If z_1 is a zero of $b(z)$, then $a(z_1) \neq 0$ since $a(z)$ and $b(z)$ have no common factors. In the neighborhood of z_1,

$$\frac{a(z)}{h\ b(z)} \approx - c(z-z_1)^{-m} \qquad c \neq o$$

for some positive integer m. But $\text{Re}[a(z)/hb(z)] \geq 0$ in a whole circle around z_1 if $|z_1| \geq 1$ for stability. Hence $b(z) \neq 0$ if $|z| \geq 1$, i.e., $a(z)/hb(z)$ is analytic for $|z| \geq 1$.

Theorem 3.7: A k-th order transformation of the form (3.1) is not A-stable if $b_k = 0$.

Proof: For some integer m, $m \geq 1$,

$$b(z) \approx c\, z^{k-m}\,, \quad \text{when} \quad z \to \infty\,, \quad c \neq o\,.$$

However, $a(z) \approx a_k z^k$, $a_k \neq 0$. Hence

$$\frac{a(z)}{b(z)} \approx d\, z^m\,, \quad \text{where} \quad d \neq 0\,,\ m \geq 1\,.$$

This however, contradicts the condition $\mathrm{Re}(s) \geq 0$ for $|z| \geq 1$

Theorem 3.8 has important implications for the modeling of linear systems. For A-stable transformations, Theorem 3.8 implies that the resultant model is externally memoryless, as shown below. Consider the initial value theorem for the z-transform

$$f_i(o,h) = \lim_{z\to\infty} F_i[g(z,h)]$$

For g(z,h) of the form (3.1), with $b_k \neq 0$

$$\lim_{z\to\infty} g(z,h) = \frac{1}{h}\cdot\frac{a_k}{b_k}$$

hence

$$f_i(o,h) = F_i\left[\frac{1}{h}\cdot\frac{a_k}{b_k}\right]$$

and therefore f(0,h) will in general have a nonzero value. The use of the discrete model can be represented by the convolution

$$y_n = \sum_{j=o}^{n} f_i(j,h) v_{n-j}.$$

where y is the subsystem output and v is the subsystem input. The leading term in the convolution is $f_i(0,h)\ v_n$, and hence the present value of the input appears on the right hand side of the discrete input-output relation. By Definition 2.2, the model is therefore externally memoryless, and implicit equations are expected for the overall interconnected model. It is important to note that these implicit equations result specifically from the A-stable modeling.

As an example of a transformation of the Form (3.1), consider the well known bilinear transformation,

$$S = \frac{2}{h} \cdot \frac{z-1}{z+1} \tag{3.2}$$

This is both A-stable and consistent as is shown below:

(1) Consistency

For the bilinear transformation applied to a transfer function F(s), consistency requires

$$\lim_{n\to\infty} \frac{n}{t} f_i\left(n, \frac{t}{n}\right) = f(t)$$

but

$$\lim_{n\to\infty} \frac{n}{t} f_i(n,h) = \lim_{n\to\infty} \frac{1}{2\pi j} \oint_C F_i\left[\frac{2}{n} \cdot \frac{z-1}{z+1}\right] \quad z^{n-1}\ dz$$

where C is a circle centered at z=0, with radius $r > 1$ since the integrand has all singularities inside the unit circle. The integral along C in the z-plane can be transformed into an integral in the s-plane.

If the inverse transformation

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

$$dz = \frac{\left(1 - \frac{sh}{2}\right)(h/2) - \left(1 + \frac{sh}{2}\right)(-h/2)}{(1 - sh/2)^2}\, ds = \frac{h\, ds}{\left(1 - \frac{sh}{2}\right)^2}.$$

is made, there results

$$\lim_{n\to\infty} \frac{1}{2\pi j} \oint_{C'} F(s) \frac{\left(1 + \frac{st}{2n}\right)^{n-1}}{\left(1 - \frac{st}{2n}\right)^{n-1}} \frac{ds}{\left(1 - \frac{st}{2n}\right)^2}$$

where C' is a line parallel to the imaginary axis and an infinite semicircle. This can be reduced to

$$\lim_{n\to\infty} \frac{1}{2\pi j} \oint_{C'} F(s) \left[\frac{1 + \frac{st}{2n}}{1 - \frac{st}{2n}}\right]^n \frac{ds}{\left(1 - \frac{st}{2n}\right)\left(1 + \frac{st}{2n}\right)}$$

and as $n\to\infty$, $\left[\frac{1 + \frac{st}{2n}}{1 - \frac{st}{2n}}\right]^n \to e^{+st}$. Thus, there results in the limit

$$\frac{1}{2\pi j} \int_{C'} F(s)\, e^{st} ds = f(t)$$

and consistency has been proved.

(2) A-Stability

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

where $s = a + jb$. Clearly, if $a < 0$, $|z| < 1$ for all $h > 0$.

A system described by a rational transfer function can always be put into normal form, hence for all rational transfer functions, the following theorem of Dahlquist [3.1] applies.

Theorem 3.8: The order of an A-stable technique cannot exceed 2. The smallest truncation error is achieved with the bilinear transformation (trapezoidal rule).

The bilinear transformation also has the desirable characteristic that rational transfer functions of order p transform into discrete models of the same order.

The bilinear transformation has been used in many different areas and approached in many different ways. In numerical analysis, it represents the generating function of the trapezoidal rule, which was one of the first techniques to be used for the integration of differential equations. In the field of digital filtering, it is used to obtain a discrete filter from a given continuous filter, with a compensation usually made because of a nonlinear warping of the frequency scale [3.4]. This nonlinear warping occurs because if

$$s = \frac{2}{h}\left(\frac{z-1}{z+1}\right)$$

and if $z = e^{s_1 t}$, then

$$s = \frac{2}{h}\tanh\left(\frac{s_1 h}{2}\right).$$

Working in digital signal processing, Steiglitz [3.5], has defined an isomorphism between the space $L_2(-\infty,\infty)$ of continuous signals and the space l_2 of discrete signals. He then shows that this isomorphism can be used to associate a discrete filter with every continuous filter through the bilinear transformation.

The bilinear transformation is not new to simulation, where it is called the Tustin [3.6] method. In the application of this method (to rational transfer functions), however, an arbitrary unit delay is inserted into every feedback loop to prevent the occurrence of implicit equations. The approach presented in this chapter does not insert any delays, for this in general destroys the A-stability of the model. The bilinear transformation will be considered further in the next section in order to explore the implications of A-stable modeling, especially when applied to irrational transfer functions.

3.3 The Bilinear Transformation

The bilinear transformation is equivalent to the trapezoidal rule, which was discussed in Chapter II for use with general nonlinear subsystems. In the simulation of a complex nonlinear system, if the trapezoidal rule is used for the nonlinear subsystems and the bilinear transformation for the linear subsystems, then the same A-stable discrete modeling method will have been used for the entire simulation. For this reason, the application of the bilinear transformation to both rational and irrational transfer functions will be considered further in this section.

If the subsystem transfer function is rational, $F_i(s) = \frac{p(s)}{q(s)}$, direct substitution of the bilinear transformation $s = g(z,h)$ gives a new rational function $F_i^*(z,h)$ of the same order as $F(s)$. This can be used to define a difference equation to be used as the discrete model of the subsystem. It has been observed [3.4] that for high order systems, the method is less sensitive to numerical round off error if $F_i(s)$ is first expanded into partial fractions and then the substitution $s = g(z,h)$ made. A simple example will serve to illustrate the method.

Example 3.1: Let $F(s)$ be the second order damped system

$$F(s) = \frac{K}{s^2 + 2\delta s+1}$$

Substituting $s = \frac{2}{h} \cdot \frac{z-1}{z+1}$, there results

$$F^*(z,h) = \frac{K}{\left[\frac{2}{h} \cdot \frac{z-1}{z+1}\right]^2 + 2\delta\left[\frac{2}{h} \cdot \frac{z-1}{z+1}\right] + 1}$$

and expanding

$$F^*(z,h) = \frac{K(h/2)^2\ (z^2 + 2z + 1)}{[1+\delta h + (h/2)^2]\ z^2 + [-1 + (h/2)^2]\ 2z + [1-\delta h + (h/2)^2]}$$

If the input to this system is denoted v_n and the output y_n, then this equation may be interpreted as a difference equation, giving the discrete model

$$y_n = \left[\frac{1}{1 + \delta h + (h/2)^2}\right] \Big\{K(h/2)^2[v_n + 2v_{n-1} + v_{n-2}] - [-1 + (h/2)^2]2y_{n-1} - [1-\delta h + (h/2)^2]y_{n-2}\Big\} \tag{3.3}$$

Note the appearance of v_n on the right hand side of equation (3.3), which means that the model is externally memoryless, as expected. The computed response of this model to a step input is shown in Fig. 3.1 for different values of the step size, h. For these computations, K was taken as unity, and a damping ratio of $\delta = .15$ was used. The error between the exact solution and the computed response for $h = .1$ is smaller than the resolution of the graph, hence the curve for $h = .1$ can also be considered as the exact response.

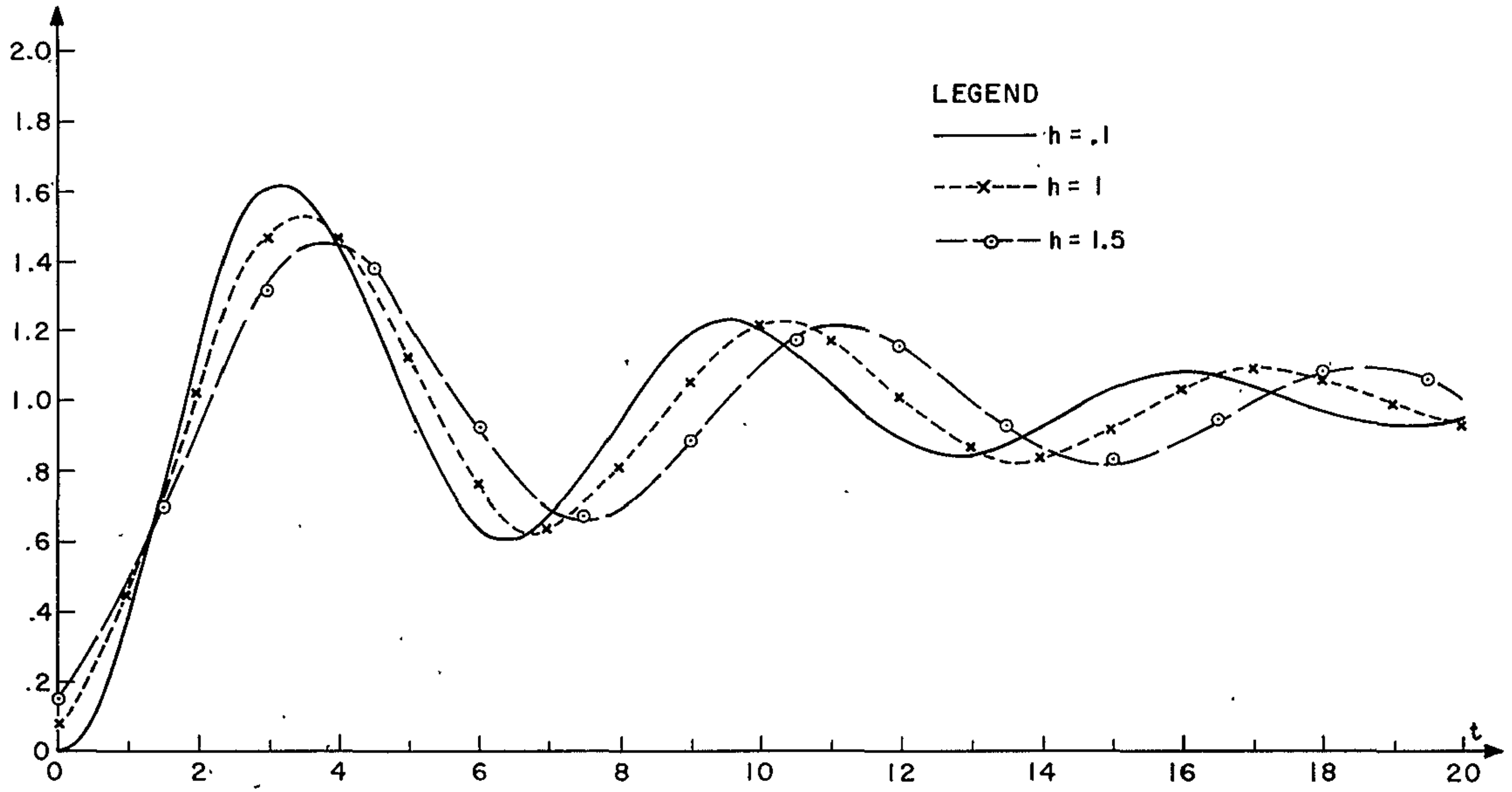


FIG. 3.1 STEP RESPONSE OF BILINEAR DISCRETE MODEL OF $\dfrac{1}{s^2 + .30s + 1}$

If F(s) is not rational, a finite difference scheme cannot be used, hence the calculation is done in the time domain, determining f(n,h) (the unit response) and using the convolution summation

$$y(n) = \sum_{k=o}^{n} f(n-k,h)v(k) .$$

The f(n,h) are identified as coefficients in the expansion

$$F^*(z,h) = \sum_{n=o}^{\infty} f(n,h)z^{-n} .$$

Depending on the form of F(s), various techniques can be used to determine the desired coefficients. One method is a direct expansion in powers of z^{-1} as shown in the following example.

Example 3.2: Let

$$f(t) = \frac{1}{\sqrt{\pi t}} \quad , \quad F(s) = \frac{1}{\sqrt{s}}$$

Then applying the bilinear transformation,

$$F^*(z,h) = \sqrt{\frac{1+z^{-1}}{1-z^{-1}}} \sqrt{\frac{h}{2}} = \frac{1}{1-z^{-1}} \sqrt{\frac{h}{2}} \sqrt{1-z^{-2}}$$

Expanding $\sqrt{1-z^{-2}}$ using the binomial theorem,

$$\sqrt{1-z^{-2}} = 1 - 1/2\, z^{-2} + \frac{1/2(-1/2)}{2!} z^{-4} - \frac{1/2(-1/2)(-3/2)z^{-6}}{3!} + \ldots$$

gives

$$F^*(z,h) = \frac{h}{\sqrt{2h}} [1 + z^{-1} + 1/2\, z^{-2} + 1/2\, z^{-3} + 3/8\, z^{-4} + 3/8\, z^{-5} + \ldots]$$

Thus

$$\{f(n,h)\} = \frac{h}{\sqrt{2h}} \{1, 1, 1/2, 1/2, 3/8, 3/8, \ldots\}$$

Another method to evaluate f(n,h) involves the use of Laguerre functions. For a general subsystem,

$$F^*(z,h) = F^* [g^{-1}(s,h),h] = F(s) \qquad (3.4)$$

hence

$$F(s) = \sum_{n=o}^{\infty} f(n,h) \frac{\left(1 - \frac{sh}{2}\right)^n}{(1 + sh/2)^n}$$

To carry the development further, a discussion of the properties of the Laguerre functions (see e.g., [3.7], p. 93f) must be given. The Laguerre polynomials are defined as

$$\ell_n(q) \triangleq e^q \frac{d^n}{dq^n} (q^n e^{-q})$$

$$= (-1)^n \left(q^n - \frac{n^2}{1!} q^{n-1} + \frac{n^2(n-1)^2}{2!} q^{n-2} + \ldots + (-1)^n n!\right)$$

thus

$$\ell_0(q) = 1$$

$$\ell_1(q) = 1-q$$

$$\ell_2(q) = q^2 - 4q + 2$$

The Laguerre functions are defined as

$$\varphi_n(q) \triangleq \frac{e^{-q/2}\ell_n(q)}{n!}$$

These functions form an orthonormal set,

$$\int_0^\infty \varphi_m(q)\ \varphi_n(q)\ dq = \delta_{mn}$$

and have the generating function

$$\varphi(q,x) = \frac{1}{1-x}\, e^{-\frac{1}{2}\left(\frac{1+x}{1-x}\right)q} = \sum_{n=o}^{\infty} x^n\ \varphi_n(q) \tag{3.5}$$

A useful recurrence relation for the Laguerre functions is

$$\varphi_{n+1}(q) = \frac{(2n+1-q)\ \varphi_n - n\varphi_{n-1}}{n+1} \tag{3.6}$$

Their Laplace transform pair is

$$\Phi_n(s) = \mathcal{L}[\varphi_n(t)] = \frac{(s - 1/2)^n}{(s + 1/2)^{n+1}}$$

Returning now to the problem of finding f(n,h), equation (3.4) is expressed in the form

$$F(s) = \left(1 + \frac{sh}{2}\right) \sum_{n=o}^{\infty} f(n,h) \frac{\left(1 - \frac{sh}{2}\right)^n}{\left(1 + \frac{sh}{2}\right)^{n+1}}$$

or, letting

$$\overline{F}(s) = \frac{F(s)}{1 + \frac{sh}{2}} \text{ , and } \overline{f}(t) = \mathcal{L}^{-1} [\overline{F}(s)]$$

as

$$\overline{f}(t) = \sum_{n=o}^{\infty} f(n,h)(-1)^n \varphi_n\left(\frac{2t}{h}\right).$$

Thus f(n,h) equals the product of $(-1)^n$ and nth coefficient in the expansion of f(t) in Laguerre functions $\varphi_n\left(\frac{2t}{h}\right)$. Note, because of orthogonality,

$$f(n,h) = (-1)^n \int_o^{\infty} \overline{f}(t) \varphi_n\left(\frac{2t}{h}\right) dt.$$

Other equivalent relations can be obtained.

This second method will be demonstrated with an example.

Example 3.3: Consider an ideal delay,

$$f(t) = u_o(t-T)$$

$$F(s) = e^{-sT}$$

then

$$F^*(z,h) = e^{-\frac{2T}{h}\left[\frac{1-z^{-1}}{1+z^{-1}}\right]} \tag{3.7}$$

In this example, effort can be saved if the similarity of (3.7) to (3.5) is noted.

Let

$$x = -z^{-1}, \quad q = \frac{4T}{h}$$

Let

$$\hat{F}(q,x) = \frac{e^{-q/2\left(\frac{1+x}{1-x}\right)}}{1-x} = \frac{1}{1-x} F^*\left(-x^{-1}, \frac{4T}{q}\right)$$

Thus, from (3.5)

$$\frac{1}{1-x} F^*\left(-x^{-1}, \frac{4T}{q}\right) = \sum_{n=o}^{\infty} x^n \varphi_n(q)$$

or

$$F^*\left(z, \frac{4T}{q}\right) = (1+z^{-1}) \sum_{n=o}^{\infty} (-1)^n z^{-n} \varphi_n(q)$$

$$= \sum_{n=o}^{\infty} (-1)^n Z^{-n} \varphi_n(q) + \sum_{n=o}^{\infty} (-1)^n Z^{-n-1} \varphi_n(q) \tag{3.8}$$

From (3.8) there results

$$f(0,h) = \varphi_o(q) = e^{-q/2}$$

$$f(1,h) = \varphi_1(q) - \varphi_0(q) = qe^{-q/2}$$

$$\vdots$$

$$f(n,h) = (-1)^n [\varphi_n(q) \quad -\varphi_{n-1}(q)] \qquad n \geq 1 \qquad (3.9)$$

If the recursion relation (3.6) is used to obtain $\varphi_n(q)$, then $f(n,h)$ is easily obtained from the previously computed values of the Laguerre functions. It should be noted that the step-size, h, is not limited to an integral submultiple of the delay time, T. The model obtained is valid for any positive h.

Plots of f(n,h) for a time delay of three units, T=3, and for various values of h are given in Fig. 3.2. These results are difficult to interpret, since a discrete model of a unit impulse at T=3 is being generated. The plots of Fig. 3.3, however, in which the step response of the above model is plotted, demonstrate that the model produces a reassonable approximation to the true step response, a step at time T=3.

The bilinear transform applied to e^{-st} results in a more complex model than would the usual z-transform identification $z = e^{sh}$. In general, consider a transfer function of the form

$$F(s) = \hat{F}(s,e^{-sT})$$

where for simplicity, T = kh, k an integer.

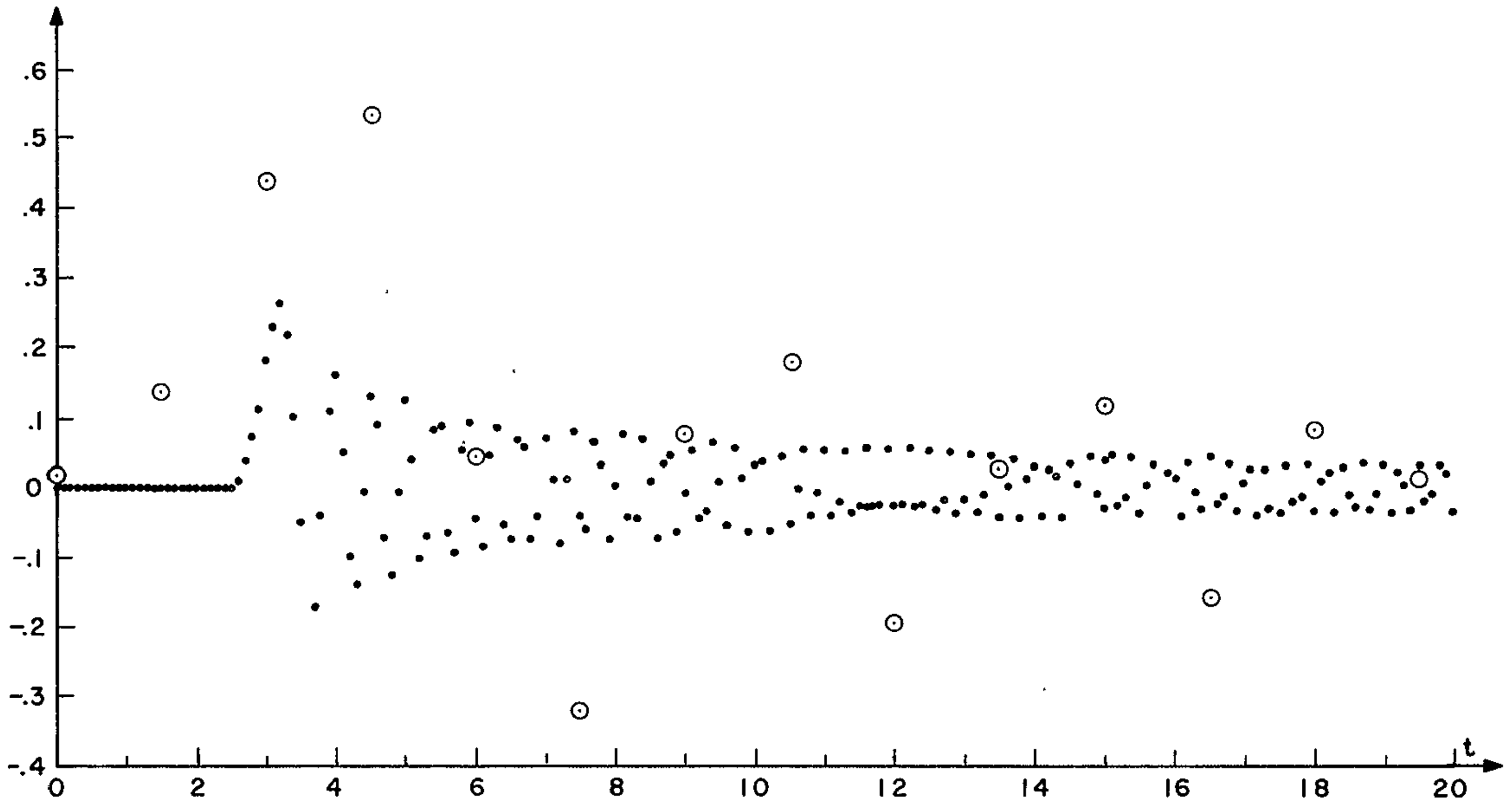


FIG. 3.2 BILINEAR APPROXIMATION OF $\bar{e}^{ST}$, FOR T=3 AND h=0.1 AND h=1.5

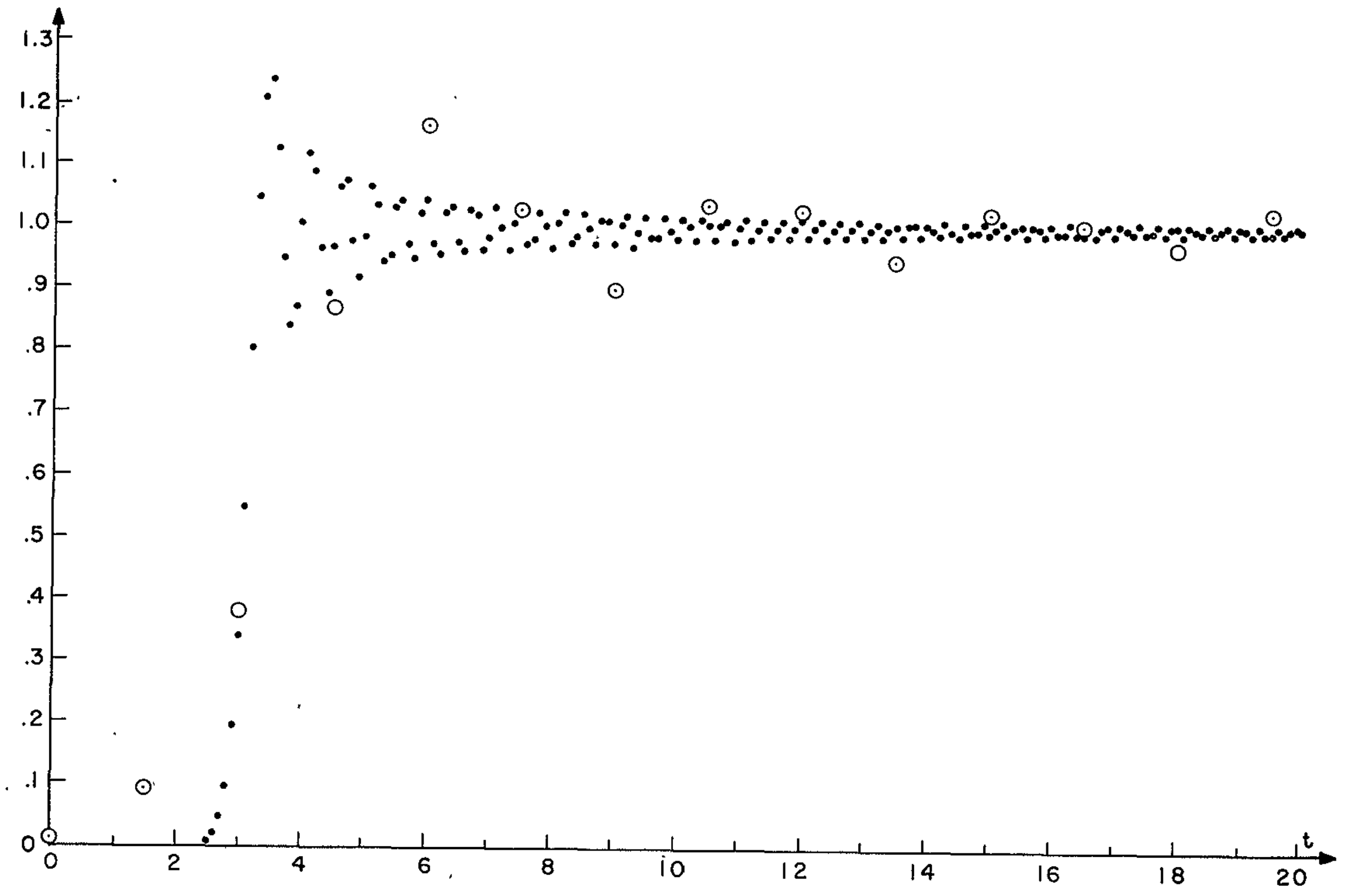


FIG. 3.3 STEP RESPONSE OF BILINEAR APPROXIMATION OF e^{-sT}, FOR T=3 AND h=0.1 AND h=1.5

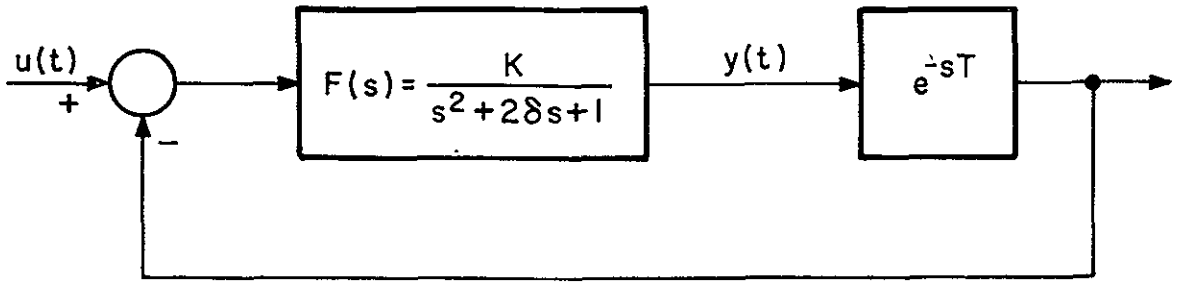
One might suppose that the transformation

$$F^*(z,h) = \hat{F}(g(z,h),\ z^{-k})$$

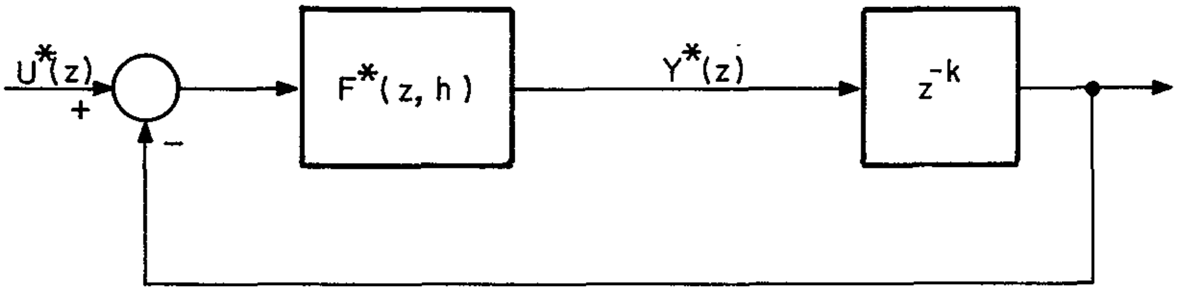
would have the same stability properties as $g(z,h)$ alone, since $z = e^{sh}$ also maps the left hand s-plane into the interior of the unit circle in the z-plane. This corresponds to using one A-stable transformation on one part of the problem, and another on a second part. The second transformation, $e^{-st} = z^{-k}$, replaces a delay in the continuous time domain by a delay in the discrete-time domain, and thus is very easy to implement. The fact is however, that A-stability is <u>not</u> generally preserved when different transformations are used, as is shown by the following example.

<u>Example 3.4</u>: Consider the system shown in Fig. 3.4a, which consists of a damped second order system followed by a time delay in a simple feedback loop. The second order system has been considered in Example 3.2, and the model derived there using the bilinear transform, equation (3.3) will be used here.

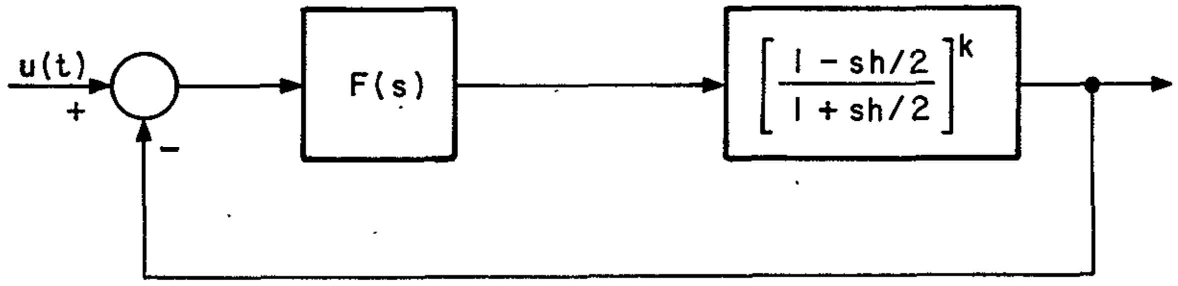
If a discrete model for the overall system is formed, combining the above model for the second order system with the simple model $e^{-st} = z^{-k}$ for the ideal delay, where $k = T/h$, an integer, the resultant model will not be A-stable, as can be seen from the following argument.



(a) CONTINUOUS SYSTEM WITH UNIT DELAY



(b) MODEL OF CONTINUOUS SYSTEM, USING TWO DIFFERENT TRANSFORMATIONS



(c) CONTINUOUS SYSTEM FOR DETERMINING STABILITY OF THE MODEL (b) (ABOVE)

FIG. 3.4 SYSTEMS FOR EXAMPLE 3.4

The discrete transfer function z^{-k} is obtained by usual z-transformation of e^{-sT}. However, it is possible to consider z^{-k} to be the result of applying the bilinear transformation to the transfer function

$$G(s) = \left[\frac{1 - sh/2}{1 + sh/2} \right]$$

$$G(g(z,h)) = z^{-k}$$

Hence, if the original continuous system were to have G(s) in place of e^{-sT} as shown in Fig. 3.4c, A-stable modeling (bilinear everywhere) would result in the model of Fig. 3.4b. If the system of Fig. 3.4c is stable, then the model of Fig. 3.4b will be A-stable. However, it is easy to choose values for the parameters for which the original system is stable, but for which the systems of Fig. 3.4c and therefore the discrete model of Fig. 3.4b are not stable.

Nyquist plots for both the original system, Fig. 3.4a, and the system of Fig. 3.4c are given in Fig. 3.5, with T=3, K = .39, and $\delta = .15$, k=2. The first plot does not enclose the -1 point, but the second does. Thus the original system is stable, yet the model of Fig. 3.4b, derived using two different modeling techniques for the separate parts of the problem, is not stable.

To verify these conclusions, a computer simulation to obtain the step response was performed and the results are given in Fig. 3.6. The calculated results for both models using a very small step size, h = .01, coincide. As expected from the Nyquist diagram, the solution is a damped sinusoid, settling to some positive value. The result for the model of Fig. 3.4b, which mixes two modeling techniques, is shown to be unstable. The remaining curve is the response of the discrete model formed by using the bilinear transformation for both subsystems. The model for e^{-sT} was derived earlier, equation (3.9).

It was found that the model of Fig. 3.4b was marginally stable at h = .75, and since the natural frequency of the second order system is equal to unity, step-sizes larger than .75 are well within the limits of the sampling theorem. The model formed using the bilinear transformation for both subsystems is, as expected, A-stable, i.e., stable for any value of h.

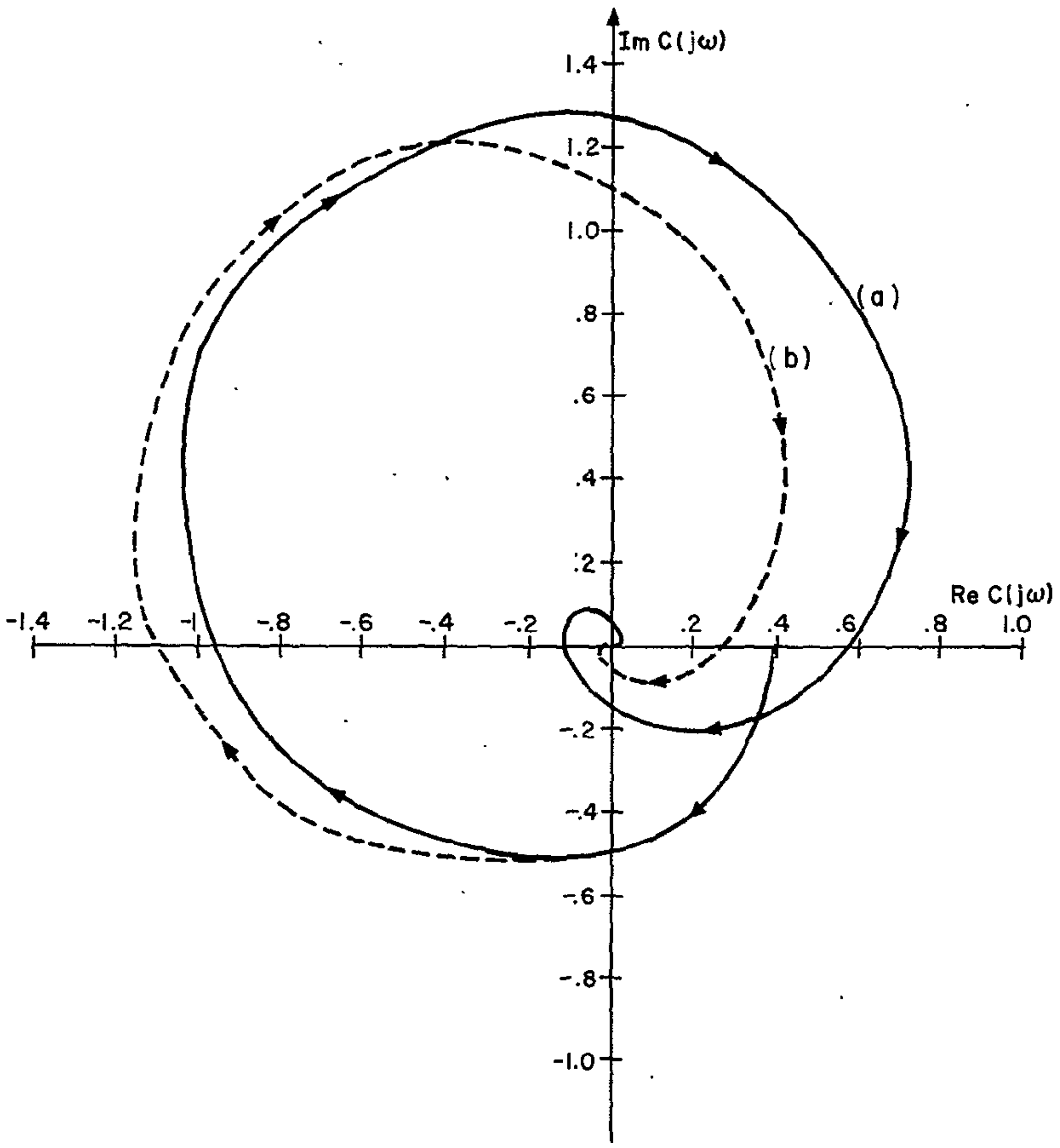


FIG. 3.5 NYQUIST PLOTS OF: (a) $F(j\omega)\cdot e^{j\omega T}$; (b) $F(j\omega)\cdot\left[\frac{1-j\omega/2}{1+j\omega/2}\right]^2$

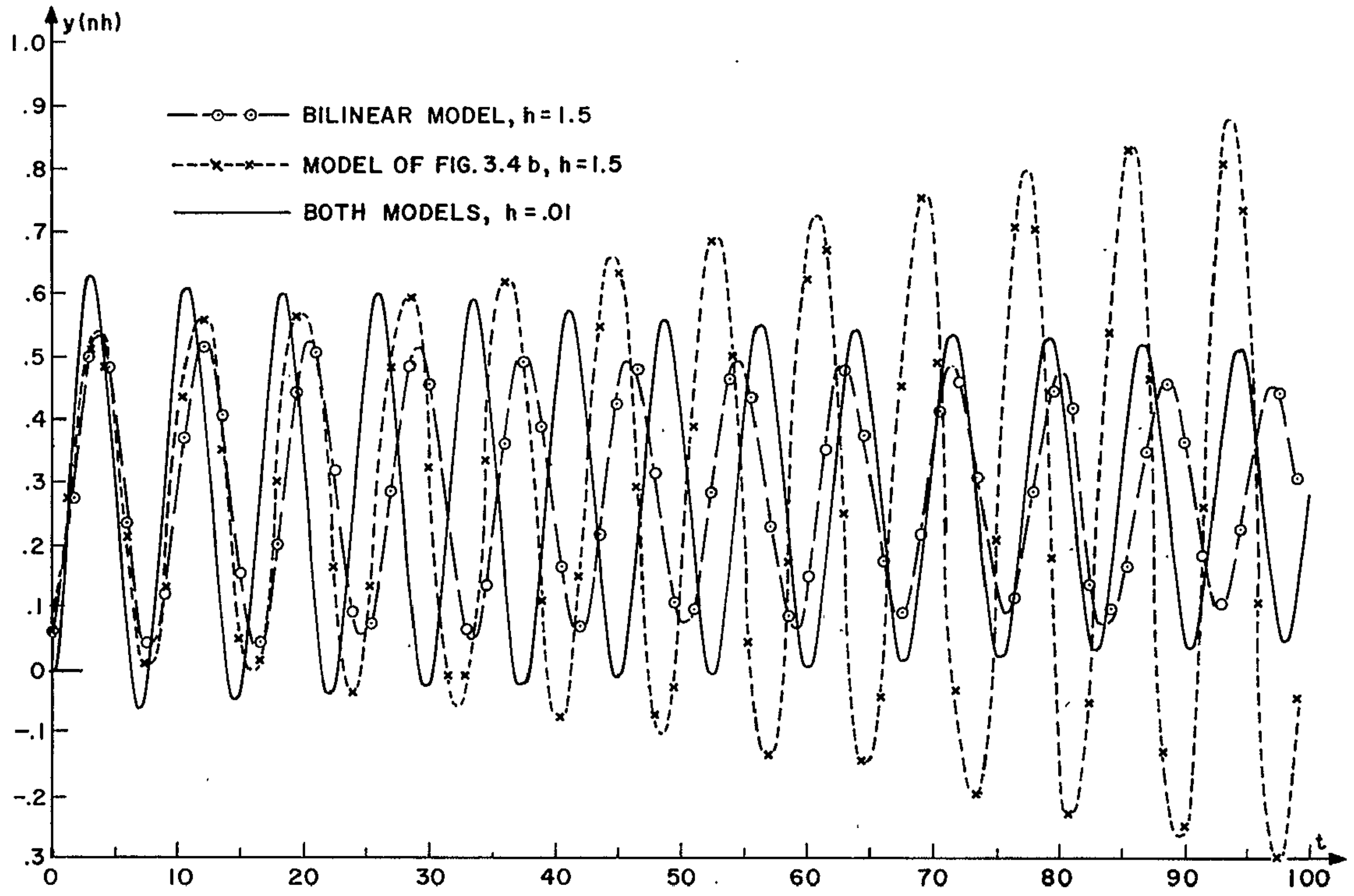


FIG. 3.6 COMPUTED RESPONSES FOR EXAMPLE 3.4

REFERENCES

[3.1] R. J. Schwarz and B. Friedland, Linear Systems, McGraw-Hill, New York, New York (1965).

[3.2] L. A. Zadeh and C. A. Desoer, Linear System Theory, McGraw-Hill, New York, N. Y. (1963).

[3.3] G. G. Dahlquist, "A Special Stability Problem for Linear Multi-Step Methods," BIT 3, no. 1, pp. 27-43 (1963).

[3.4] F. F. Kuo and J. F. Kaiser, eds,, System Analysis by Digital Computers, Chapter 7, John Wiley, New York, N. Y. (1966)

[3.5] K. Steiglitz, "The Equivalence of Digital and Analog Signal Processing," Information and Control, 8, pp. 455-467 (1965).

[3.6] A. Tustin, "A Method of Analyzing the Behavior of Linear Systems in Terms of Time Series," J.I.E.E. 94 pt. II-A, pp. 130-142 (May 1947).

[3.7] R. Courant and D. Hilbert, Methods of Mathematical Physics, Vol. I, Interscience, New York, N. Y. (1965).

CHAPTER IV

SORTING AND ITERATION

4.1 Introduction

It was seen in the previous chapters that for both the OSM and the ISM methods, a set of equations, usually implicit, must be solved at each time point to provide the simulation outputs. Computational procedures for the solution of these equations are presented in this Chapter.

4.2 System Equations and Directed Graphs

The OSM method for simulation, which was presented in Chapter II, resulted in a constraint equation (2.6) to be solved at each time point t_n. This equation, in discrete notation, is

$$\underline{v}_n = M\ G(\underline{x}_n,\underline{v}_n,t_n) + P\underline{u}_n\ . \qquad (4.1)$$

As in equation (2.6), G represents the combined vector of output equations for each system described in normal form. The appearance of $\underline{v}_n$ on the right hand side of (4.1) is due solely to direct physical connections from input to output in the original continuous system.

The ISM method for simulation produced a similar equation to be solved, equation (2.12), repeated here for convenience,

$$\underline{v}_n = M\ R\ (\underline{v}_n,t_n) + P\underline{u}_n \qquad (4.2)$$

where R represents the combined vector of discrete models for the input-output-state relations of the subsystems, assuming none are internally implicit. The appearance of $\underline{v}_n$ on the right hand side of (4.2) is due either to direct physical connections from input to output in the original physical system (which was the only reason that $\underline{v}_n$ appears in (4.1)) or to the modeling of dynamic subsystems as externally memoryless. The case of internally implicit subsystems is left to the end of this section.

In equation (4.1), the state of each subsystem is assumed to be computed before it is needed in the evaluation of each component of G, thus $\underline{x}_n$ can be regarded as a known entity. For solution purposes, equations (4.1) and (4.2) are thus in the same form, and in what follows only equation (4.2) need be considered.

It will be recalled that the combined matrix [M⋮P] has only one non-zero element, a(+1), in each row, so that equation (4.2) represents a sparse system of equations. To attempt solution without taking advantage of the structure of the equations would be computationally inefficient. For example, consider a simple cascade of N single-input single-output subsystems, with an external input, u_1, feeding the first system. Equation (4.2) for this system is

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} r_1(v_1,t_n) \\ r_2(v_2,t_n) \\ r_3(v_3,t_n) \\ r_4(v_4,t_n) \\ \vdots \\ r_n(v_n,t_n) \end{bmatrix} + \begin{bmatrix} u_1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where the time subscript has been deleted on all variables except t_n in order to simplify notation. As one possible approach to solution, consider a simple iteration procedure of the type used in the transmission line example of Section 2.4, i.e.,

$$\underline{v}_n^{k+1} = M\,R\,(\underline{v}_n^k, t_n) + P\,\underline{u}_n$$

which begins with some initial estimate v_n^o. For each iteration, all N subsystems are evaluated, and it can be shown that convergence is achieved in exactly N iterations. Hence a total of N^2 subsystem evaluations must be made. But this would be an extremely inefficient method of solution, for the equations can be solved by simply evaluating them in the proper cascaded order,

$$\begin{aligned} v_1 &= u_1 \\ v_2 &= r_1(v_1, t_n) \\ &\vdots \\ v_N &= r_{N-1}(v_{N-1}, t_n) \end{aligned}$$

In general, a sequential evaluation of the above type is possible only if equation (4.2) is not implicit, i.e., the system is free of directed loops. For example, if the input to the first system of the cascade connection considered above is connected to the final output, the result is a simple feedback loop. If each equation is substituted into the succeeding equation, there results,

$$v_N = r_N(r_{N-1}(r_{N-2}(\dots\ r_1(v_N)\ \dots)))$$

Thus there exists only one unknown, v_N to be determined. For this simple loop, a substitution technique is one obvious method that reduces the work to be done in solving the equations. In general, however, the structure is more complex and more sophisticated techniques will be necessary.

In order to take advantage of the structure of equation (4.2) for general systems, a directed graph representation will be used. Each vertex of the graph will represent one subsystem, and the arcs will represent components of $\underline{v}_n$. The external inputs, $\underline{u}_n$, will be considered source nodes. An example is given in Fig. 4.1, in which the arc $v_{i,j}$ represents the j^{th} input to subsystem i. This example can be used to suggest a general solution technique based on the theory of directed graphs.

Looking at the graph, it can be seen that there are no directed arcs from the set of nodes [3,4,5,6] to the set [1,2], or equivalently, there is only feed-forward of information from subsystems [1,2] to subsystems [3,4,5,6]. Hence, with $v_{1,1} = u_1$ (known), one method of solution is to solve the equations represented by the directed loop between systems [1] and [2] and thus to produce $v_{3,1}$. With $v_{3,1}$ determined, the relationships among the subsystems [3,4,5,6] can now be solved.

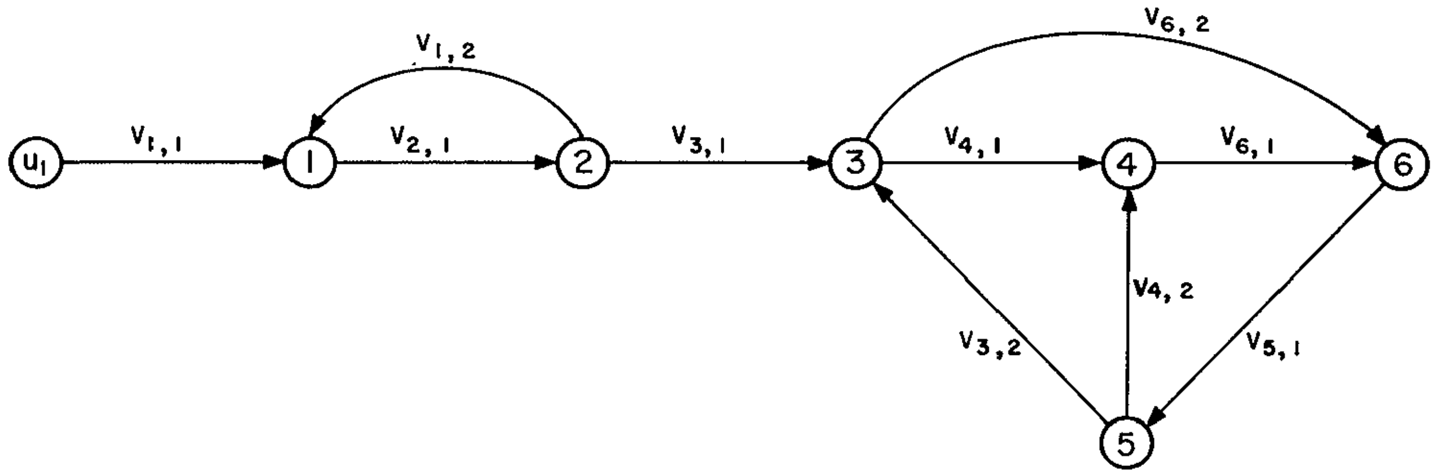


FIG. 4.1 SYSTEM REPRESENTATION BY A DIRECTED GRAPH

Generalizing, it is desired to partition the subsystems (nodes) into sequential groups, with each group containing a minimum number of subsystems, so that only a feed-forward of information exists from each group to the groups that follow. With this partitioning, the equations of each group can be solved simultaneously, and the results used in solving the equations of the groups which follow. A minimum number of systems is desired in each group because this minimizes the number of equations which must be solved simultaneously. For each group, the subgraph formed by deleting all arcs from the system graph except those interconnecting the nodes of the group will be called the group-graph. For complex systems, an algorithm is necessary to accomplish the partitioning and one developed by Steward [4.1], based on previous work of Sargent and Westerberg [4.2] is given in Section 4.3.

The partitioning process is only the first step toward a solution, for it now remains to solve the coupled equations of each partitioned group. A subset of the variables of the group-graph, called iteration variables, are chosen such that it would be possible to evaluate all the subsystems of the group in some order if these variables had known values. If we denote this subset as w_i, for the ith group, then, for this group, equation (4.2) can be simplified to

$$w_i = Q_i(w_i) \tag{4.3}$$

where the function Q_i is simply a sequence of subsystem evaluations.

If all the subsystems of the group are linear, then equation (4.3) is linear, i.e.,

$$w_i = Dw_i + d \, .$$

In this case, the equations of the group can be solved explicitly to give,

$$w_i = [I-D]^{-1} \, d \, .$$

By definition, the determination of w_i allows the explicit evaluation of all the subsystems of the group, and hence the updating process for the group is complete. If, however, the group contains any non-linear subsystem, then an explicit solution is not generally possible, and iteration procedures must be used.

An iteration procedure is a technique of the form

$$w_i^{k+1} = E[Q_i(w_i^k] \qquad (4.4)$$

where w_i^k denotes the kth iterate and the sequence begins with some initial estimate w_i^o . E is some estimation operator which chooses a new and hopefully better value for w_i from the previous value. In order to reduce the computational effort, the smallest possible set of variables w_i should be chosen for each partitioned group.

The choice of a minimum number of iteration variables for each group can be shown equivalent to a standard problem in the theory of directed graphs, the determination of a minimum feedback arc set for each group-graph. The following definitions, taken from Younger [4.3], will help to make this identification.

Definition 4.1: For a directed graph, a feedback arc set is a set of arcs which, if removed, leaves the resultant graph circuit free.

Definition 4.2: A feedback arc set is minimum if no other feedback arc set for that graph consists of a smaller number of arcs.

Definition 4.3: A sequential ordering of a graph of N nodes is a 1-1 function from the nodes of the graph to the integers 1,...,N.

Definition 4.4: For a directed graph of N nodes with sequential order R, the connection matrix

$$C = [C_{R(i),R(j)}]_{N,N}$$

has one row and one column for each node of the graph, and the entry at row R(i), column R(j), denoted $C_{R(i),R(j)}$ = number of arcs from node i to node j.

Younger shows that for any sequential ordering R of the nodes of the graph, the removal of those arcs (i,j) for which $R(i) \geq R(j)$ must eliminate all directed loops. If all arcs (i,j) for which $R(i) \geq R(j)$

are selected as iteration variables, then, since no directed loops remain, the subsystems can be evaluated in the order determined by R. Hence, a sequential ordering determines a choice of iteration variables. To find a minimum number of such variables, it is therefore desired to find a sequential ordering R which minimizes the number of arcs (i,j) for which $R(i) \geq R(j)$. This is exactly the problem of determining a minimum feedback arc set for the group-graph. This problem is considered in the next section, but first a discussion of systems containing internally implicit subsystems will be given.

It was seen in Chapter II that there are two approaches to the solution of the equations of internally implicit subsystems. In the first approach the equations were to be solved by a process internal to the subsystem. Hence, from an input-output viewpoint, the equations of the subsystems need never be considered and the development of the proceeding section applies exactly.

It is possible, however, to treat the equations of an internally implicit subsystem simultaneously with the implicit equations of the overall simulation. This approach will in general be more efficient since the structure of the overall simulation is taken into account. A method of accomplishing this will now be given.

The equations of an internally implicit subsystem can have two forms, equations (2.10) and (2.11). Equation (2.11) will be considered first, since the solution technique is more easily illustrated for this form. Equation (2.11) is,

$$x_n = q(x_o,\ldots,x_n;\ v_o,\ldots,v_n;t_o,\ldots,t_n)$$
$$y_n = r(x_o,\ldots,x_n;\ v_o,\ldots,\ v_n;t_o,\ldots,t_n).$$

Since, for this model, the equation is implicit in the state, x_n, an obvious solution technique is to consider the state as an auxiliary iteration variable for this subsystem. This in no way affects the choice of iteration variables for each group-graph, since only inputs and outputs are shown in these graphs.

If the system equation is of the form (2.10), repeated here,

$$y_n = p(y_o,\ldots,y_n;\ v_o,\ldots,v_n;t_o,\ldots,t_n)$$

then the implicit equation for this subsystem involves the output, y_n. This would represent a self-loop in the group-graph, were it to be added there. In order to break all loops, this self loop must be broken. Thus y_n can be immediately labeled as an iteration variable, and removed from the group-graph.

To remove the arcs corresponding to y_n for a specific subsystem, it is necessary only to modify the matrix M as follows. The (i,j)th entry of M indicates that the j^{th} output is connected to the i^{th} input. Thus all entries in the column corresponding to the output in question need simply be removed. In the following section, it will be assumed that this modification has been made, if necessary.

4.3 Sorting Procedure

As seen in the previous section, the sorting procedure should act in two steps. The first step is to partition the nodes of the system graph into sequential groups such that there exists only feed-forward of information from any group to those groups which follow it in the ordering. This defines which subsystem equations must be solved simultaneously and in what order. The second step is to find a minimum feedback arc set for each group-graph, which determines a set of iteration variables for each group of equations.

A preliminary simplification of the system graph is possible which reduces the complexity of the partitioning process. The source nodes (external inputs) and the arcs which leave them, can never be iteration variables, for they represent known quantities. In essence, the source nodes can be considered as a first group of the partition, for their "equations" are always solved by definition. Hence in what follows, the matrix of external input connections P, need not be considered.

In order to apply the partitioning algorithm, the connection matrix, C, must be determined. All the information necessary to determine this matrix is contained in the matrix M of equation (4.2) from which the system graph was derived. Adding together all the rows of M corresponding to subsystem i, i=1,...,N and adding together all the columns corresponding to subsystem j, j=1,...,N, there results a new matrix, S, whose $(i,j)^{th}$ element is the number of arcs from subsystem j to subsystem i. Hence, the connection matrix is the transpose of S, or $C = S^t$.

Once the connection matrix is determined, the algorithm presented by Steward [4.1] can be used to partition the subsystems into the desired groups. This algorithm proceeds as follows:

(1) A column is sought having no predecessors, i.e., no entry in the column, and that column and the row corresponding to it eliminated. This process is repeated until there are no further columns without predecessors.

(2) Any path traced following predecessors must lead into a loop. Since the graph is finite and by Step (1) there is no column without a predecessor, such a path must eventually repeat a column. Retracing the path from this column gives a loop.

The union of two columns is defined to be the sum of the columns, mod 2.

(3) When a loop is found, the set of columns in the loop is replaced by one column which is the union of the columns replaced. This is called collapsing the columns of the loop. Similarly, the rows corresponding to these columns are collapsed. Any entries which result on the diagonal are set equal to zero, since they simply represent connections within the collapsed loop.

When a column without predecessors is eliminated, that column and the columns which collapsed to form it represent the subsystems in a group. The order in which columns without predecessors are eliminated gives an order in which these groups may be solved.

Example 4.1: The connection matrix for the graph of Fig. 4.1, eliminating the known input $v_{1,1}$ and source u_1, is

$$C = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 1 & 0 & 1 \\ 4 & 0 & 0 & 0 & 0 & 0 & 1 \\ 5 & 0 & 0 & 1 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}$$

There are no columns without predecessors. Beginning in column (1), the loop 1-2 is found. Collapsing the loop gives

$$\begin{array}{c|ccccc} & 1\text{-}2 & 3 & 4 & 5 & 6 \\ \hline 1\text{-}2 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1 & 0 & 1 \\ 4 & 0 & 0 & 0 & 0 & 1 \\ 5 & 0 & 1 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Column (1) now has no predecessors, and can be removed. Thus, [1,2] is the first group. Removing this column and row gives

$$\begin{array}{c|cccc} & 3 & 4 & 5 & 6 \\ \hline 3 & 0 & 1 & 0 & 1 \\ 4 & 0 & 0 & 0 & 1 \\ 5 & 1 & 1 & 0 & 0 \\ 6 & 0 & 0 & 1 & 0 \end{array}$$

The loop 3-5-6 is now found, and collapsing these rows and columns gives,

$$\begin{array}{c|cc} & 3\text{-}5\text{-}6 & 4 \\ \hline 3\text{-}5\text{-}6 & 0 & 1 \\ 4 & 1 & 0 \end{array}$$

These two columns form a loop, hence the second group is [3,4,5,6]. This grouping was previously obtained intuitively.

With the partitioning accomplished, a minimum set of iteration variables must be found for each group. In the previous section, it was stated that the problem of determining a minimum set of iteration variables corresponds to the problem of determining a minimum feedback arc set for the group-graph. In order for this correspondence to hold for all systems, a simple modification must be made in the group-graph, under certain conditions, to adjust for the following difficulty. In the group-graph, a single output of one subsystem going to many inputs represents only one variable, but many arcs. However, if the subsystem has separate outputs going to each of the other inputs, then this represents many variables and as many arcs. The two graphs look identical except for labeling of the arcs, but are very different as far as choice of iteration variables is concerned. If the single variable of the first situation is chosen for iteration, then this removes all the arcs which it labels. To break all the arcs in the second case, all the variables must be chosen for iteration. To remedy this difficulty, a single output going to a multiplicity of inputs will first be removed from the node through a single arc to a new inserted node which will then branch to the necessary input locations. The cutting of the single arc connecting the original node and the inserted node will always be equivalent to the cutting of all the branching arcs.

Once the graph has been modified as above, a minimum feedback arc set for the group-graph is a minimum set of iteration variables. It was seen in the previous section that an ordering of the nodes determines a feedback arc set. Hence, such an ordering will be determined according to the following definition [4.3].

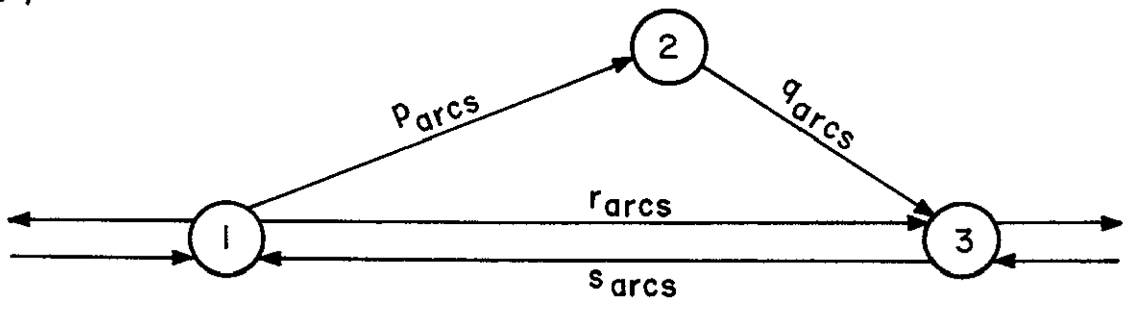
Definition 4.5: An optimum ordering R is a sequential ordering of the nodes of a directed graph for which $\{(i,j),\ R(i) \geq R(j)\}$ is a minimum feedback arc set.

Before an optimum ordering is sought, recognition of certain characteristics of the graph may simplify the problem. The first concerns loops of two arcs. A theorem of Younger's gives this simplification.

Theorem 4.5: The set of optimum orderings for a given graph is invariant under the removal of loops involving two arcs. In the case of multiple arcs between nodes, e.g., p arcs from node i to node j and q arcs from node j to i, with $p > q$, then up to q arcs may be removed from i to j and the same number removed from j to i.

Hence, when searching for an optimum sequential ordering, all loops involving only two nodes can be removed from the graph. As an example, the first group of the graph in Fig. [4.1], nodes [1,2], contains a loop which can be removed. In this case, both possible orderings are optimal, for the ordering [1,2] gives $v_{1,2}$ as the only iteration variable, and the ordering [2,1] gives $v_{2,1}$ as the only iteration variable.

A second simplification, which is an extension of a simplification first given by Sargent and Westerberg [4.2], can be accomplished if the form shown in Fig. 4.2a appears in the graph. The arcs to the left and right of nodes 1 and 3 respectively denote connections to the remaining nodes of the graph. For this configuration, to cut all paths 1-2-3, only the min[p,q] arcs need be cut. Node 2 therefore can be temporarily removed, and the arcs from nodes 1-3 replaced, a)



b)

POSSIBLE ORDERING	NUMBER OF FEEDBACK ARCS	
1 2 3	s	NODE 1 PRECEDES NODE 3
2 1 3	$p + s$	
1 3 2	$q + s$	
3 2 1	$p + q + r$	NODE 3 PRECEDES NODE 1
2 3 1	$p + r$	
3 1 2	$q + r$	

FIG. 4.2 SPECIAL CONFIGURATION WHICH LEADS TO SIMPLIFICATION OF THE SYSTEM GRAPH

using Theorem 4.5, by a number of arcs r-s+ min[p,q], where a positive result is interpreted as arcs from 1-3 and a negative result as arcs from 3-1. If the chart in Fig. 4.2b is examined, it is seen that the final position of node 2 is determined by the relative position of nodes 1 and 3. If 1 precedes 3, then the final ordering should be 1-2-3 and if 3 precedes 1, then the final ordering should be 2-3-1 if $p < q$ or 3-1-2 if $q < p$.

The simplification techniques presented above greatly reduce the effort involved in finding an optimum ordering; however, the algorithm developed by Younger will work whether or not they have been applied. Younger's algorithm is given in complete detail in [4.3], but the salient characteristics are presented here.

Definition 4.6: For a sequential ordering R of a directed graph, a consecutive subgraph is any nonempty subgraph composed of nodes consecutively ordered by R and the arcs connecting them in the overall graph.

For subgraphs G_1 and G_2 of a directed graph G which contain no common node, let $C_{G_1G_2}$ denote the number of arcs each of which leaves a node in G_1 and enters a node in G_2.

Definition 4.7: An ordering R for a directed graph is admissible if:

(a) the graph ordered by R satisfies $C_{G_1G_2} \geq C_{G_2G_1}$ for all appropriate consecutive subgraphs G_1,G_2

(b) the feedback arc set determined by R contains no proper subset that is also a feedback arc set.

An admissible ordering for a group-graph is intuitively a good ordering. A first admissible ordering is easy to find and in fact, the search process could be stopped here if it were decided that a small, but not necessarily minimum, number of iteration variables would be acceptable. Younger's algorithm begins by finding an admissible reference ordering. A new ordering R', which has a smaller number of feedback arcs, is sought, through a perturbation process. This new ordering R' becomes the new reference ordering, and this process repeated until a reference ordering is obtained which cannot be improved. This is an optimum ordering, and determines a minimum number of iteration variables.

Example 4.2: A brief sketch of the application of this algorithm to the graph of Fig. 4.1 will now be given. For the second partitioned group, nodes [3,4,5,6], the first step is to choose an initial ordering and check to see if it is admissible. If it is not, the testing process yields the information needed to change to an ordering which is admissible. For the example, the given numbering of the nodes, [3,4,5,6] serves as a first try. This is shown to be not admissible, because the subgroup [3,4] has no arcs going to the group [5], while [5] has two returning to [3,4]. The algorithm permutes the nodes to obtain the admissible reference ordering [3,6,5,4]. This ordering gives the two iteration variables $v_{6,1}$ and $v_{3,2}$. It is seen in the graph that if these two arcs are removed, no loops remain. However, this ordering is not optimum, for the algorithm next produces the ordering [5,3,4,6] with only one feedback arc, $v_{5,1}$. It is seen in the graph that removing only $v_{5,1}$ does break all loops. Fig. 4.3 shows how the overall simulation would proceed, once the sorting information is available for this example.

Younger's algorithm has one characteristic, already alluded to, that makes it more suitable to computer sorting of the equations than other techniques. This algorithm can be stopped at any point, and the results used meaningfully, because whatever admissible reference ordering has been found at that point serves to determine a good (but not optimum) set of iteration variables. The other applicable techniques, in particular dynamic programming, must work completely to a solution before any choice of iteration variables is possible. For a large problem, it may be preferable to preset an amount of computation time that will be allowed for this stage of the simulation, and stop the search if this time is reached. With Younger's algorithm, the simulation proper could be begun at this point, whereas with dynamic programming the search has simply failed.

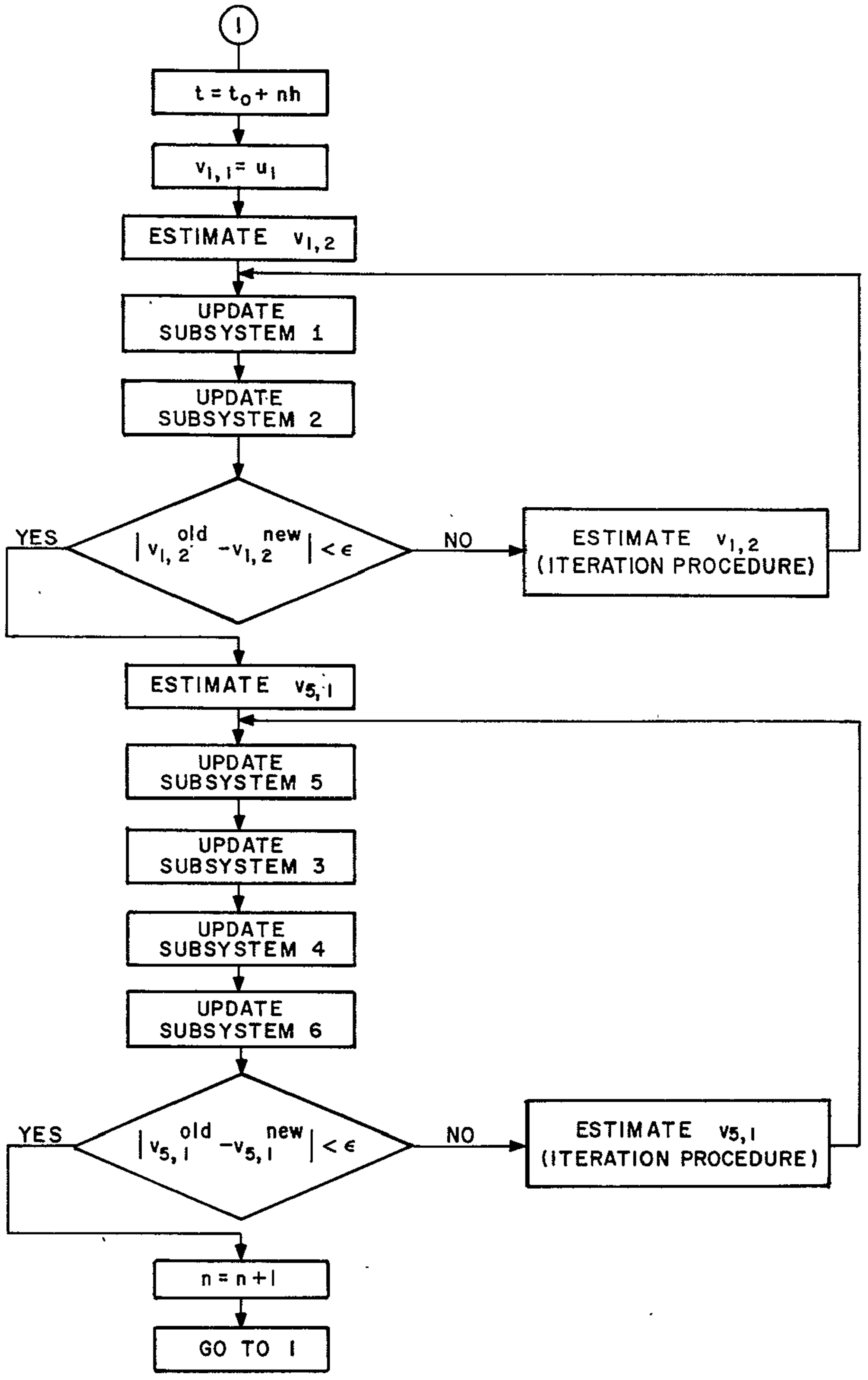


FIG. 4.3 FLOW DIAGRAM OF SIMULATION FOR EXAMPLE 4.2

A summary of the entire sorting procedure is given below:

(1) Partition the nodes with the partitioning algorithm.

(2) Modify any outputs which connect to more than one input to include an extra node.

(3) Apply the two simplification methods to each group-graph (optional)

(4) Find a sequential ordering for each group which determines a minimum set of iteration variables (optionally, a suboptimal set).

The final result is the sequential ordering of the groups, with each group properly ordered (and thus determining the iteration variables).

4.4 Iteration Procedures

In the preceding section, the equations of each partitioned group were shown to have the form of equation (4.3), repeated here

$$w_i = Q_i(w_i) \tag{4.5}$$

where Q_i is a sequence of subsystem evaluations determined by the sequential ordering, R_i for the ith group-graph. The ordering R_i was found which minimized the number of variables, w_i. In this section, techniques for solving (4.5) for each group-graph will be discussed, where it is assumed that the group contains at least one nonlinear subsystem. The purpose of this discussion is to show how the structure of the system model influences the choice of an iteration procedure.

In order to simplify notation, the subscript i will be dropped in what follows, since only the equations of one group will be considered at a time. It will be recalled that equation (4.5) must be solved at each time point, and that the time subscript, n, was previously removed. Thus in what follows, the solution of equations of the form

$$w = Q(w)$$

are considered.

The first method which suggests itself is <u>direct iteration</u> of the form[‡‡]

$$w^{k+1} = Q(w^k) \ , \ w^o \tag{4.6}$$

[‡‡] Throughout this section the superscript notation $(\ )^k$ indicates "k-th iterate".

This method was successfully used in the transmission line example of Section 2.4. The properties of this iteration procedure follow easily from some definitions and theorems of analysis (see, e.g., [4.4], p. 627ff).

Definition 4.8: A mapping $Q(w)$ of the w-space into itself is said to be a contraction if there exists a $c < 1$ such that

$$\|Q(w) - Q(w')\| \leqq c\|w-w'\| \qquad \text{for all } w,w'$$

Here $\|\cdot\|$ denotes a vector norm, e.g. $\|x\| \triangleq \max_i |x_i|$

Let $Q_w = \frac{\partial Q}{\partial w}$

Theorem 4.6: Given a continuously differentiable function $Q(w)$, if

$$\|Q_w\| \leqq c < 1 \qquad \text{for all } w,$$

then $Q(w)$ is a contraction. Here $\|\cdot\|$ denotes the matrix norm

$$\|Q\| \triangleq \max_i \sum_j |q_{ij}| .$$

<u>Theorem 4.7:</u> If Q is a contraction mapping, a unique solution w^* of equation (4.6) exists. w^* can be obtained as the limit of a sequence $\{w^k\}$, where

$$w^{k+1} = Q(w^k) \qquad k = 0,1,2... \tag{4.7}$$

and w^o is any given initial value.

The rate of convergence of w^k to the solution is given by

$$\|w^k - w^*\| \leqq \frac{c^k}{1-c} \|w^1 - w^0\| .$$

The direct iteration procedure thus converges only if the norm of the Jacobian of Q satisfies a Lipschitz condition. The Lipschitz constant, c, determines the rate of convergence of the sequence and if $c > 1$, the sequence may diverge. For most systems, the function Q is not a contraction and more sophisticated iteration procedures must be used.

At this point it is worth noting the effect of integration step-size on the convergence of the iteration procedure. The implicit equations will often result from the use of A-stable modeling of subsystems, in which case the step-size h will appear as a parameter in the function Q. In this case, Q will often be a contraction only for values of the step-size less than some value, h_c. If direct iteration is to be used to solve the equations, then to achieve convergence, it must be required that h be less than h_c. But this reintroduces a limitation on h;

which negates the purpose of A-stable modeling. Thus direct iteration is not generally useful when A-stable modeling has been used.

Another useful technique is Newton-Raphson, which for equation (4.6) is

$$w^{k+1} = w^k - \left[I - Q_w^k\right]^{-1} [w^k - Q(w^k)] \qquad k = 0,1,2,\ldots$$

where
$$Q_w^k \triangleq \left.\frac{\partial Q}{\partial w}\right|_{w = w^k} .$$

The difficulty in applying Newton-Raphson to the solution of equation (4.6) for general systems results from the necessity of evaluating the Jacobian Q_w^k at each iteration. In general, Q_w^k may be very difficult to compute, or may simply not be available. If the Jacobians of each individual subsystem are included in the mathematical model of that subsystem, then the sorting procedure of the previous section can be used to select the proper derivatives as the loops are traversed. It would be desirable however, to avoid as much of the aforementioned computation as possible. A method due to Broyden [4.5] has this property. The method of Broyden chooses a new estimated value for $\left[I - Q_w^k\right]^{-1}$ from the values of the functions at each iteration. The method uses this estimated value in place of the true value in a Newton iteration, and hence belongs to a class of methods called quasi-Newton methods.

This method is outlined below, with slight modifications due to the special nature of the simulation equations being solved.

(1) Obtain an initial estimate w^o of the solution. The final value for w at the previous time point will usually be a good choice.

(2) Obtain an initial estimate of the iteration matrix $\left[I-Q_w^o\right]^{-1} = H^o$.

Again, the final value for H from the previous time point is a very good choice.

(3) Compute $f^k = w^k - Q(w^k)$

(4) Compute $p^k = -H^k f^k$

(5) Choose t^k such that if

$$w^{k+1} = w^k + p^k t^k$$

then

$$\|f^{k+1}\| < \|f^k\| .$$

Broyden shows [4.6], that if the method is close to a solution of (4.5), then the Newtonian value of unity for t^k can be chosen and the norm minimization avoided. Since, in the problem considered here, good initial estimates are available at each time step, t^k will be set to unity.

(6) Test f^{k+1} for convergence. If convergence has not been reached,

(7) Compute $y^k = f^{k+1} - f^k$

(8) Compute $H^{k+1} = H^k - (H^k y^k - p^k t^k)(p^k)^t H^k/(p^k)^t H^k Y^k$

(9) Repeat from step (4).

Broyden's method avoids the computation of the Jacobian, but the cost may be an increased number of iterations at each step. However, since a good initial estimate of the solution and iteration matrix are available at each step, in general the number of iterations will not be much higher than that achieved using true Newton-Raphson.

The following example illustrates the necessity of using the more complex iteration procedures.

Example 4.2: Consider the transmission line example of Section 2.4, with the capacitor C_2 removed from the nonlinear termination (see Fig. 2.2). With the capacitor removed, there results a term in the Jacobian Q_w corresponding to the incremental resistance of the right hand termination, i.e.,

$$\frac{\partial v_2}{\partial i_2} \ .$$

In the linear case, this term equals R_2, and with a value of 2K, the norm of Q_w exceeds unity in magnitude and it is expected that direct iteration would diverge. Similarly, for the diode load,

$$\frac{\partial v_2}{\partial i_2} = \frac{v_T}{i_2 - I_0}$$

which for small values of i_2 also exceeds unity in magnitude. It was confirmed experimentally that direct iteration diverged for either the linear or diode load.

Both Newton-Raphson and Broyden's method were now applied to the problem with good results. For the linear load, with h = 1/32 and ϵ = .001, both methods took almost exactly the same average number of iterations per step, 1.78. This is not surprising, for once Broyden's method has solved the equations at the first time point, it has converged to the exact value of the iteration matrix, which is constant in time for the linear loads. Hence, after the first time point, there is essentially no difference between Broyden's method and Newton-Raphson. In the nonlinear case, both methods converged, with Broyden's method taking a slightly higher average number of iterations per step, 1.92 for Broyden's versus 1.83 for Newton-Raphson. Computationally however, this means that Broyden's method was more efficient, since two extra function evaluations are needed to compute the necessary derivatives for Newton-Raphson.

The above analysis and computational results suggest that it is useful to have available a good hierarchy of iteration schemes, using direct iteration if possible, followed by Broyden's method if the former diverged. Ideally, however, a general purpose simulation language would have a large repertoire of iteration routines, similar to the presently available repertoires of integration methods, with the choice left to the user.

REFERENCES

[4.1] D. V. Steward, "Partitioning and Tearing Systems of Equations," J. SIAM Numer. Anal. Ser. B 2, no. 2, pp. 345-365 (1965).

[4.2] R. W. H. Sargent and A. W. Westerberg, "Speed-up in Chemical Engineering Design," Tran. Instn. Chem. Engrs. 42, pp. T190-T197, (1964).

[4.3] D. H. Younger, "Minimum Feedback Arc Sets for a Directed Graph," IEEE Trans. Circuit Theory CT-10, no. 2, pp. 229-245, (June, 1963).

[4.4] L. V. Kantorovich and G. P. Akilov, "Functional Analysis in Normed Spaces, Pergamon Press, London, England (1964).

[4.5] C. G. Broyden, "A Class of Methods for Solving Nonlinear Simultaneous Equations," Math. Comp. 19, pp. 577-593, (October, 1965).

[4.6] C. G. Broyden, "A New Method of Solving Nonlinear Simultaneous Equations," Computer Journal 12, no. 1, pp. 94-99, (February, 1969).

CHAPTER V

CONCLUSIONS

The primary goal of this research has been to develop computational methods for the digital simulation of continuous systems. It was desired to develop methods which would be applicable to a wider class of systems and would be more efficient than currently available simulation languages.

By analyzing current simulation methods, it was possible to distinguish and describe two fundamentally distinct approaches to simulation, herein named the OSM and the ISM methods. The OSM method, in general use in available simulation languages, models the overall system, its component parts or subsystems having already been incorporated. The ISM method models the subsystems and then interconnects these models. It was shown that the ISM method is applicable to a wider class of systems than the OSM method. Thus, to achieve greater generality for a simulation languages, the ISM method should be used. Within this context the types of numerical modeling techniques admissible for the ISM method were determined, and the characteristics of the simulation which result from using these methods were investigated.

One problem which limits the generality of available simulation languages is their inability to solve sets of implicit equations. Previously, this problem was not considered a limitation serious enough to justify structuring a simulation language around the solution of implicit equations. As was shown in this research, however, the solution of implicit equations is related to another very serious limitation of current simulation languages: their inability to simulate stiff systems efficiently.

A-stable modeling is discussed as a satisfactory technique for the efficient simulation of stiff systems, and a new development for the application of A-stable methods to modeling of linear systems is given. It was shown that A-stable modeling results in implicit equations for the overall simulation. Thus, since the practical, efficient handling of stiff systems involves implicit equations, solving these equations becomes a necessity and justifies the structuring of simulation languages around this problem.

In order to structure simulation languages to deal with implicit equations, sorting and iteration procedures for the efficient solution of these equations are presented. Subsystems are sorted to obtain a good computational sequence. Since nonlinear systems are generally present, iteration procedures to solve the system equations are presented.