ARGESIM Report no. 5

Fuzzy Systems and Control

COMETT - Course
Seminar Modellbildung und Simulation

D. Murray-Smith, D.P.F. Möller, F. Breitenecker

ISBN 3-901608-05-2

ISBN 3-901608-05-2

ARGESIM Report No. 5

ARGE Simulation News (ARGESIM)
c/o Technical University of Vienna
Wiedner Hauptstr. 8-10
A-1040 Vienna, Austria
Tel: +43-1-58801 5386, 5374, 5484
Fax: +43-1-5874211
Email: argesim@simserv.tuwien.ac.at
WWW: <URL:http://eurosim.tuwien.ac.at/>

FOREWORD

Simulation may be defined as the process of developing a computer-based model of a system and of experimenting with this computer model. The purpose of simulation may be to gain a better understanding of the behavior of the system, or to locate specific problems, or to evaluate different strategies, or as an aid in designing a new process or system.. Automatic control systems provide one important area of application for simulation techniques.

This course is in two parts. The first deals with methods of process simulation for control systems applications, including the internal verification and external validation of plant models. The second part provides an introduction to the principles of fuzzy control and to fuzzy logic in system automation.

The introductory part of the course is general and addresses issues of plant modelling which are important in many different fields of application. Examples are presented, which illustrate the importance of accurate modelling and simulation in control system investigations for both conventional and fuzzy controller configurations. Two case studies are included, which provide further illustrations of the role of simulation techniques in control system applications, including issues of internal verification and external validation.

In the second half of the course the emphasis is on the application of fuzzy logic principles to automatic control problems. The first session provides an introduction to fuzzy logic concepts, while in the second the emphasis is on the application of fuzzy principles to control system design.

It is hoped that after the course a participant should be able to make decisions about the use of modelling and simulation methods for automatic control system applications, and to have an appreciation of the potential of fuzzy logic methods in control engineering and system automation. In particular, the course should serve to emphasize the importance of simulation techniques in the investigation of real control systems involving nonlinear plant characteristics or, as in the case of fuzzy systems, nonlinearities within the controller also. Analytical methods provide little real insight in these situations and simulation techniques are therefore of special importance in dealing with the complexities of practical control systems, whatever the specific area of application.

About ARGESIM

ARGE Simulation News (ARGESIM) is a non-profit working group providing the infra structure for the administration of **EUROSIM** activities and other activities in the area of modelling and simulation.

ARGESIM organizes and provides the infra structure for

- the production of the journal EUROSIM Simulation News Europe
- the comparison of simulation software (EUROSIM Comparisons)
- the organisation of seminars and courses on modelling and simulation
- COMETT Courses on Simulation
- "Seminare über Modellbildung und Simulation"
- development of simulation software, for instance: mosis - continuous parallel simulation, D_SIM - discrete simulation with Petri Nets, GOMA - optimization in ACSL
- running a WWW - server on EUROSIM activities and on activities of member societies of EUROSIM
- running a FTP-Server with software demos, for instance
  - * demos of continuous simulation software
  - * demos of discrete simulation software
  - * demos of engineering software tools
  - * full versions of tools developed within ARGESIM

At present ARGESIM consists mainly of staff members of the Dept. Simulation Technique and of the Computing Services of the Technical University Vienna.

In 1995 ARGESIM became also a publisher and started the series **ARGESIM Reports**. These reports will publish short monographs on new developments in modelling and simulation, course material for COMETT courses and other simulation courses, Proceedings for simulation conferences, summaries of the EUROSIM comparisons, etc.

Up to now the following reports have been published:

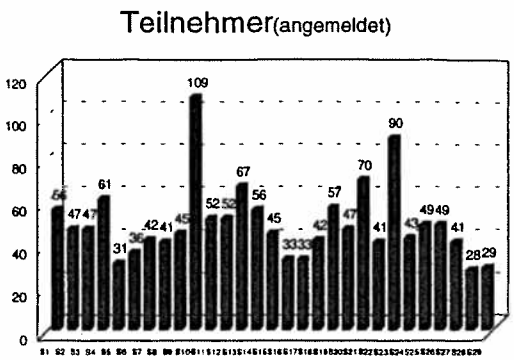
No.	Title	Authors / Editors	ISBN
#1	Congress EUROSIM'95 - Late Paper Volume	F. Breitenecker, I. Husinsky	3-901608-01-X
#2	Congress EUROSIM'95 - Session Software Products and Tools	F. Breitenecker, I. Husinsky	3-901608-02-8
#3	EUROSIM'95 - Poster Book	F. Breitenecker, I. Husinsky	3-901608-03-6
#4	Seminar Modellbildung und Simulation - Simulation in der Didaktik	F. Breitenecker, I. Husinsky, M. Salzmann	3-901608-04-4
#5	Seminar Modellbildung und Simulation - COMETT - Course "Fuzzy Systems and Control"	D. Murray-Smith, D.P.F. Möller, F. Breitenecker	3-901608-05-2
#6	Seminar Modellbildung und Simulation -COMETT - Course "Object-Oriented Discrete Simulation"	N. Kraus, F. Breitenecker	3-901608-06-0
#7	EUROSIM Comparison 1 - Solutions and Results	F. Breitenecker, I. Husinsky	3-901608-07-9
#8	EUROSIM Comparison 2 - Solutions and Results	F. Breitenecker, I. Husinsky	3-901608-08-7

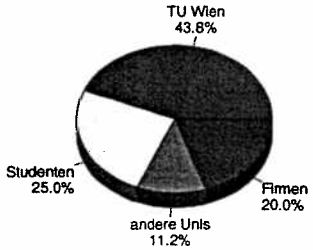
For information contact: ARGESIM, c/o Dept. Simulation Techniques,
attn. F. Breitenecker, Technical University Vienna
Wiedner Hauptstraße 8-10, A - 1040 Vienna
Tel. +43-1-58801-5374, -5386, -5484, Fax: +43-1-5874211
Email: argesim@simserv.tuwien.ac.at

Seminare über Modellbildung und Simulation

Seit dem Frühjahr 1991 veranstaltet das EDV-Zentrum gemeinsam mit der Abteilung Regelungsmathematik und Simulationstechnik des Instituts für Technische Mathematik und der ARGE Simulation News (ARGESIM) Vortragsveranstaltungen zum Thema Modellbildung und Simulation (Simulationsseminare). Organisatoren sind I. Husinsky und F. Breitenecker. Das Ziel ist, verschiedene Simulationswerkzeuge vorzustellen, über ihre Einsatzmöglichkeiten zu informieren und Erfahrungen auszutauschen. Ferner werden bekannte Simulationsfachleute eingeladen, Grundsatzvorträge zum Thema Simulation zu halten. Im allgemeinen werden die Seminare teilweise von Firmen gesponsert oder über Simulationsprojekte mitfinanziert. Sie dauern einen halben oder einen Tag, es gibt schriftliche Unterlagen zu den Vorträgen und Softwareprodukten. Ein Buffet fördert die Kommunikation zwischen den Seminarteilnehmern in den Pausen.
Bis jetzt haben folgende Seminare stattgefunden:

S1	23. 4. 1991	ACSL
S2	4. 6. 1991	CTRL_C, XANALOG
S3	22. 10. 1991	SIMUL_R
S4	5. 5. 1992	ACSL
S5	6. 5. 1992	MicroSaint
S6	17. 6. 1992	Objektorientierte Modellbeschreibung und qualitative Simulation (F. Cellier, University of Arizona)
S7	1. 7. 1992	Diskrete Simulation und Analyse (D. Kelton, University of Minnesota)
S8	23. 10. 1992	GPSS/H (T. Schriber, University of Michigan)
S9	10. 12. 1992	SIMPLE
S10	2. 2. 1993	MATLAB und SIMULINK
S11	25. 3. 1993	Modellbildung mit Bondgraphen (D. Karnopp, University of California)
S12	24. 5. 1993	MicroSaint
S13	22. 6. 1993	ACSL
S14	21.10.1993	XANALOG, SIMNON
S15	22.10.1993	GPSS/H (T. Schriber, University of Michigan)
S16	11.11.1993	IDAS
S17	7.12.1993	SIMPLE++
S18	14.12.1993	Petrinetze, D_SIM (R. Hohmann, Magdeburg)
S19	4.2.1994	Modellbildung und Simulation in der Lehre
S20	14.3.1994	GPSS/H und Proof (T. Schriber, University of Michigan)
S21	13.4.1994	ACSL
S22	10.5.1994	SIMUL_R, Partielle Differentialgleichungen
S23	22. 11. 1994	MATLAB/SIMULINK
S24	14.12.1994	SIMPLE++
S25	31.1.1995	Parallele Simulation, mosis
S26	28.3.1995	ACSL
S27	29.3.1995	MicroSaint
S28	13.6.1995	COMETT II, Part one, Discrete Simulation
S29	28.6.1995	COMETT II, Part two, Simulation and Automatisation





Die Teilnehmer, etwa 30 bis 110 je Seminar, kommen zum Großteil von der TU, aber auch von anderen Universitäten und aus der Industrie. Bei den bisherigen Seminaren waren etwa 20% der Teilnehmer aus der Industrie.

Das Programm eines Seminars setzt sich im allgemeinen aus einem oder zwei Grundlagenvorträgen, mehreren Anwendervorträgen, Produktpräsentationen, Vorführungen am Rechner und Diskussionen zusammen.

Die Teilnehmer werden um eine Anmeldung gebeten, daher können die Unterlagen (Seminarberichte), die zu Beginn des Seminars verteilt werden, schon eine Teilnehmerliste enthalten. Ab Herbst 1995 erscheinen die Unterlagen als ARGESIM Report. Alle, die bereits an einem Seminar teilgenommen haben, werden automatisch zu den weiteren Seminaren eingeladen.

Information:
I. Husinsky, EDV-Zentrum, Technische Universität Wien, Wiedner Hauptstr. 8-10, A-1040 Wien,
Tel: (0222) 58801 5484, Fax: (0222) 587 42 11,
E-Mail: husinsky@edvz.tuwien.ac.at

Prof.Dr. F. Breitenecker, Abt. Regelungsmathematik u. Simulationstechnik, Inst. 114, Technische Universität Wien, Wiedner Hauptstr. 8-10, A-1040 Wien,
Tel: (0222) 58801 5374, Fax: (0222) 587 42 11,
E-Mail: fbreiten@email.tuwien.ac.at

TABLE OF CONTENTS

Foreword	iii
About ARGESIM	iv
Seminare "Modellbildung und Simulation"	v
Problem Organisation for Control System Simulation	1
Simple Examples of Control System Simulation Using Common Simulation Tools	21
Internal Verification and External Validation of Simulation Models for Control Systems Analysis and Design	49
SLIM - A Simple Continuous System Simulation Language	83
Was ist Fuzzy Logic	111
Fuzzy Logic in USA, Japan, Deutschland	119
Strukturelle Gliedrung der Fuzzy Applikationen	133
Fuzzy Control	155

Fuzzy Control for Automatisation

Freitag, *8. September 1995,*

Technische Universität Wien
Seminarraum des EDV-Zentrums

Programm

9^{00} **Begrüßung, Vorstellung des COMETT-Kurses**
F. Breitenecker

9^{15} **Problem Organisation For Control System Simulation**
Simple Examples of Control System Simulation Using Common Simulation Tools
Internal Verification and External Validation of Simulation Models for Control Systems Analysis and Design
Prof. Dr. D. Murray-Smith, University of Glasgow

10^{45} **Kaffeepause**

11^{00} **Case Study I - Plant Modelling for a Two-Tank Liquid Level Control System**
Case Study II - An Aircraft Automatic Landing System
Prof. Dr. D. Murray-Smith, University of Glasgow

12^{30} **Mittagspause**

13^{30} **Fuzzy Systems**
Prof. Dr. D.P.F. Möller, TU Clausthal-Zellerfeld

15^{00} **Kaffeepause**

15^{30} **Fuzzy Control**
Prof. Dr. D.P.F. Möller, TU Clausthal-Zellerfeld

17^{00} **Abschließende Diskussion**

Seminar über Modellbildung und Simulation, TU Wien, 8. 9. 1995, Teilnehmerliste

Dipl.Ing.	Astner	Klaus			Jochen-Rindt-Str. 22/4/2	A	- 1230 Wien	astner@iiasa.ac.at
Prof. Dr.	Breitenecker	Felix	Technische Universität Wien	Abt. Simulationstechnik	Wiedner Hauptstr. 8-10	A	- 1040 Wien	fbreiten@email.tuwien.ac.at
Dipl.Ing.	Cizl	Claudia	(TU Wien, Abt. Simulationstechnik)		Sibeliusstr. 9/3/4	A	- 1100 Wien	
	Forsthuber	Edwin	(TU Wien, Abt. Simulationstechnik)		Ortsstr. 9	A	- 2362 Biedrmannsdorf	e9026231@fbma.tuwien.ac.at
Dipl.-Ing.	Gannadi-Khosh	Habib	TU Wien	Inst. f. Tragwerkslehre	Karlsplatz 13/254	A	- 1040 Wien	
Dipl.Ing.	Geißler	Robert	TU Wien	Inst.f. Computertechnik	Gußhausstr. 25-29	A	- 1040 Wien	robert@ict.tuwien.ac.at
Dr.	Hick	.	Österr. Forschungszentrum Seibersd			A	- 2444 Seibersdorf	
	Klug	Markus	(TU Wien, Abt. Simulationstechnik)		Frauengasse 19/3	A	- 1170 Wien	mklug@ws1.atv.tuwien.ac.at
Dipl.Ing.	Koroknai	Stefan	(TU Wien, Abt. Simulationstechnik)		Obermüllnerstr. 17/31	A	- 1020 Wien	koroknai@ws1.atv.tuwien.ac.at
Dr.	Krösl	Peter	Ludwig Boltzmann Institut für	Experimentelle Traumatologie	Donaueschingenstr. 13	A	- 1200 Wien	
	Lämmerhofer	Michael	(TU Wien, Abt. Simulationstechnik)		Mariahilferstr. 143/6	A	- 1150 Wien	
	Mittermayr	Christian	TU Wien	Inst. f. Analytische Chemie	Getriedemarkt 9	A	- 1060 Wien	
	Popp	Peter Hanns	(TU Wien, Abt. Simulationstechnik)		Weilburgstr. 10/5/1	A	- 2500 Baden	
	Redlein	Alexander	TU Wien	Inst. f. Automation	Treitlstr. 3	A	- 1040 Wien	
Dr.	Salzmann	Manfred	(TU Wien, Abt. Simulationstechnik)		Lorenz Müllerg. 1/138	A	- 1200 Wien	msalz@ws1.atv.tuwien.ac.at
	Schäfer	Erich	Universität für Bodenkultur	IWGF, Abt. Siedlungswasserbau	Nußdorfer Lände 11	A	- 1190 Wien	schaefer@iwgf-sig.boku.ac.at
Prof.Dr.	Troch	Inge	TU Wien	Abt. Simulationstechnik	Wiedner Hauptstr. 8-10	A	- 1040 Wien	
	Valentin	Christoph	(TU Wien, Abt. Simulationstechnik)		Weinbergg. 93/18/1	A	- 1190 Wien	
Dipl.Ing.	Weinmeier	Peter	TU Wien	Inst. f. El. Maschinen	Gußhausstr. 27-29	A	- 1040 Wien	
Dr.	Weisz	Willy	TU Wien	EDV-Zentrum	Wiedner Hauptstr. 8-10	A	- 1040 Wien	weisz@edvz.tuwien.ac.at
	Widerin	Ute	(TU Wien, Abt. Simulationstechnik)		Hofmühlg. 7A	A	- 1060 Wien	

Problem Organisation For Control System Simulation

1. Introduction

Many methods exist for describing a mathematical model. Sets of ordinary differential equations and algebraic equations are clearly of central importance in lumped parameter dynamic models. In the case of linear models used for control system design there are other forms of description which may be more natural, including transfer functions, block diagrams, signal flow graphs and bond graphs. All of these can be useed to show, in diagrammatic form, the mathematical operations to be performed and the order in which information must be processed.

2. Descriptions for continuous-variable models: reduced and state variable forms

Two of the most widely used forms of mathematical description for dynamic systems are the so-called **reduced form** and the **state variable form**. An example of a simple model expressed in reduced form is the equation

$$M\ddot{y}+R\dot{y}+Ky = f(t) \qquad (1)$$

which is the differential equation commonly used to represent the mechanical system of Figure 1 involving the displacement, y(t), of a mass M which is subjected to an external force f(t) while suspended by means of a linear spring of stiffness K and damping element having viscous resistance R.

The corresponding description in state variable form involves two first order equations in place of the single second order equation. It takes the form

$$\dot{x}_1 = x_2 \qquad (2)$$

$$\dot{x}_2 = -\frac{K}{M}x_1 - \frac{R}{M}x_2 + \frac{f(t)}{M} \qquad (3)$$

where the new variable x_1 is the displacement, y(t), and the new variable x_2 is the velocity, dy/dt. These are known as the **state variables** of the system. An nth order description in reduced form is equivalent to a set of n first order ordinary differential equations in state variable form. There is no unique set of state variables for a given model in reduced form.

Physical reasoning is often used in selecting quantities to be defined as state variables. In the system of Figure 1 it is clear that displacement and velocity have advantages over other possible sets of state variables. Displacement is a variable of particular interest in the model and is also likely to be a measured quantity in the real system. Velocity is also likely to be a measurable quantity.

The two-dimensional vector having components x_1 and x_2 is known as the state vector. The importance of the state vector is that if the initial state is defined and the system inputs

are known all future states are defined. The two equations defining the state-space model above can be rewritten as a single vector matrix equation

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0 & 1\\ -\frac{K}{M} & -\frac{R}{M}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ \frac{1}{M}\end{bmatrix} f(t) \qquad (4)$$

or, more concisely, as

$$\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{b}u \qquad (5)$$

where **x** is the state vector, A is a 2×2 square matrix and **b** is a two-element column vector. This equation relates the rate of change of the state to the present state and the input. It is a form which is particularly convenient for simulation purposes since the numerical solution can then be obtained for each equation within the state-space model simply by a process of integration. A second order system, with two state variables, thus requires two integration operations in the corresponding simulation program; a sixth order model, with six state variables, would require six integrations.

In the case of a nonlinear model the state space model would have the form

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}(t), u(t)) \qquad (6)$$

where **f** now denotes the vector of derivative functions and u(t) is the input.

3 Conversion from reduced form to state-variable form

If the model is given in the form

$$\frac{d^n y}{dt^n} = f\left[t, y, \frac{dy}{dt}, \ldots, \frac{d^{n-1}y}{dt^{n-1}}, u(t)\right] \qquad (7)$$

where u(t) represents an input forcing function and f is a given linear or nonlinear function, it is always possible to convert the model to state space form by selecting the following as the state variables

$$x_1 = y$$

$$x_2 = \frac{dy}{dt}$$

.

.

$$x_n = \frac{d^{n-1}y}{dt^{n-1}}$$

$$x_n = \frac{d^{n-1}y}{dt^{n-1}} \qquad (8)$$

In the case of a mechanical system, this could involve selecting position, velocity, acceleration etc. as the set of state variables. The resulting state space description has the form

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\vdots$$

$$\dot{x}_{n-1} = x_n$$

$$\dot{x}_n = f[t, x_1, x_2, \ldots, x_n, u(t)] \qquad (9)$$

Since the choice of state variables is never unique the state variables may be numbered in any order.

4 Transfer function descriptions

Linear lumped parameter models can also be written very conveniently as transfer functions, defined as the ratio of the Laplace transform of the model output variable to the Laplace transform of the input when all initial conditions are zero. Most introductory texts dealing with control systems engineering (e.g. [1], [2]) provide a detailed account of transfer function models and their derivation.

The assumption that all initial conditions are zero when using a transfer function type of model allows a given differential equation to be transformed into the Laplace domain simply by replacing d/dt by s, d^2/dt^2 by s^2 and so on for higher derivatives. Subsequent analysis is all carried out in terms of algebraic relationships which both simplifies the mathematical operations which have to be performed and can also provide additional physical insight. For example, the mass-spring-damper system defined in Equation (1) transforms to

$$Ms^2Y(s) + RsY(s) + KY(s) = F(s) \qquad (10)$$

where Y(s) is the Laplace transform of y(t), F(s) is the Laplace transform of f(t) and all

initial conditions on y(t) and its derivatives are zero. The quantity Y(s) can then be manipulated algebraically to give

$$(Ms^2+Rs+K)\,Y(s) = F(s) \tag{11}$$

and thus

$$\frac{Y(s)}{F(s)} = \frac{1}{Ms^2+Rs+K} \tag{12}$$

In general a transfer function G(s) may be written in the form

$$\frac{Y(s)}{U(s)} = G(s) = \frac{A(s)}{B(s)} \tag{13}$$

where Y(s) is the Laplace transform of the output variable, U(s) is the Laplace transform of the input and where A(s) and B(s) are polynomials in s.

Many properties of a transfer function depend upon the denominator B(s). The roots of the **characteristic equation** B(s)=0 largely determine the form of the output when the transfer function is subjected to a given input. These roots are the **poles** of the transfer function. The number of poles is, of course, equal to the order of the system as discussed above in the context of state variable descriptions.

5. Block diagram and signal flow graph representations

Linear systems can be described using block diagrams or signal flow graphs. Block diagrams and signal flow graphs are important for simulation since these both provide a simple means of expressing the structure of a complex model. Many simulation packages are block diagram orientated and a thorough understanding of these diagrams is essential.

In the block diagram approach to the description of a system each element is described by a single block. The arrow entering the block represents the input variable and the arrow leaving the block represents the output variable. The block contains an expression, usually in transfer function form, which relates the output to the input. Signal flow diagrams incorporate exactly the same information as the block diagram. In this case input and output variables are represented by nodes and the line connecting two nodes is the equivalent of the block in the block diagram approach. A given system can be represented at many different levels using a block diagram or signal flow graph. For example Figure 2 shows valid block diagram and signal flow graph representations for the mass-spring-damper system of Equation (1). In these diagrams the only variables which appear are the input variable F(s) and the output Y(s). The diagrams describe only the relationship between the chosen input variable and the chosen output variable. Figure 3 shows another level of representation. Here the diagrams involve a number of blocks or signal flow graph elements in combination and the structure provides additional information about the system. It should, of course, be noted that the diagrams of Figures 2 and 3 are exactly equivalent and Figure 2 may be derived from Figure 3 using standard rules for block diagram or signal flow graph manipulation and reduction.

6. Block diagram and signal flow graph methods for transfer function simulation

Several methods exist for deriving a set of equations in state variable form from a given transfer function. The methods given here are based on block-diagrams or signal-flow graphs. more than one approach is presented because some methods, which can be entirely satisfactory with low order models, present problems of numerical robustness when applied to higher order problems.

6.1 The direct construction approach

Most transfer functions of practical importance can be manipulated into a general form involving a ratio of two polynomials in s and a gain factor. Consider the transfer function

$$\frac{Y(s)}{U(s)} = \frac{K(s^m + a_{m-1}s^{m-1} + \ldots\ldots\ldots + a_2 s^2 + a_1 s + a_0)}{s^n + b_{n-1}s^{n-1} + \ldots\ldots\ldots\ldots + b_2 s^2 + b_1 s + b_0} \qquad (14)$$

where $n > m$ and K is a simple gain factor. This is in the required form and the restriction that the order of the denominator should be greater than that of the numerator is connected with conditions for physical realisability.

The only transfer functions which are of general practical importance for control systems applications but which cannot be manipulated into the form shown in Equation (14) involve pure delay elements involving factors exp(-sT) or distributed delay elements involving factors exp(-$\sqrt{s}$T). However, such cases can be handled without difficulty if the delay elements are simple multiplicative factors which can be treated as a separate block in cascade with a block described by a transfer function involving a ratio of polynomials in s.

Dividing all terms in the numerator and denominator of the right hand side by s^n gives

$$\frac{Y(s)}{U(s)} = \frac{K(s^{m-n} + a_{m-1}s^{m-1-n} + \ldots\ldots + a_2 s^{2-n} + a_1 s^{1-n} + a_0 s^{-n})}{1 + b_{n-1}s^{-1} + \ldots\ldots\ldots\ldots + b_2 s^{2-n} + b_1 s^{1-n} + b_0 s^{-n}} \qquad (15)$$

It is now possible to rewrite the relationship between the transfer function output Y(s) and the input U(s) to involve an intermediate variable E(s). This new variable need not have any obvious physical significance and is defined from the following equation

$$\frac{Y(s)}{U(s)} = \frac{Y(s)}{E(s)} \cdot \frac{E(s)}{U(s)} \qquad (16)$$

The transfer function of Equation (15) may now be split into two parts as follows

$$\frac{Y(s)}{E(s)} = K(s^{m-n} + a_{m-1}s^{m-1-n} + \ldots\ldots + a_2 s^{2-n} + a_1 s^{1-n} + a_0 s^{-n}) \qquad (17)$$

and

$$\frac{E(s)}{U(s)} = \frac{1}{1 + b_{n-1}s^{-1} + \ldots\ldots + b_2 s^{2-n} + b_1 s^{1-n} + b_0 s^{-n}} \qquad (18)$$

Equation (18) may be re-arranged to give

$$E(s) = U(S) - (b_{n-1}s^{-1} + \ldots + b_2 s^{2-n} + b_1 s^{1-n} + b_0 s^{-n})\,E(s)$$

which corresponds to a signal flow graph of the form shown in Figure 4. Here the new variable E(s) is the input to a sequence of integrator elements, the output of each of which is fed back to the input through coefficient elements. The outputs of each of the integrators, taken in order from the left, are thus $s^{-1}E(s)$, $s^{-2}E(s)$, $s^{-3}E(s)$,....,$s^{-n}E(s)$. However from Equation (17) the output Y(s) is seen to be a weighted sum of m+1 quantities such as these and this gives a signal flow graph such as that shown in Figure 5.

Variables from each of the integrator blocks in Figure 5 may be assigned as state variables equations in state variable form may be written down from the signal flow graph or block diagram by inspection. If the output of the final integrator is x_1, with the other state variables taken as the outputs of each of the remaining integrator elements. The set of state equations is then as follows

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

.
.
.

$$\dot{x}_{n-1} = x_n$$

$$\dot{x}_n = -b_0 x_1 - b_1 x_2 - \ldots - b_{n-1} x_n + u(t) \qquad (19)$$

This set of equations is similar to Equations (9). A single algebraic equation relating the output y to the state variables must now be added. It is clear from the block diagram, that this equation is

$$y = K(a_0x_1+a_1x_2+....+a_{m-1}x_{n-m}+x_{n-m+1}) \qquad (20)$$

Equations (19) and (20) together provide a simple way for implementing the given transfer function within a simulation. The only disadvantage of this direct construction approach is that the properties of the resulting state space model can be highly sensitive to small numerical inaccuracies in coefficient values, especially in the case of high-order models.

6.2 The parallel construction approach

The parallel programming approach is appropriate when the given transfer function has a denominator in factored form as shown below

$$\frac{Y(s)}{U(s)} = \frac{K(s^m+a_{m-1}s^{m-1}+.....+a_2s^2+a_1s+a_0)}{(s+\beta_1)(s+\beta_2).....(s+\beta_n)} \qquad (21)$$

Using partial fractions it is then possible to express the right hand side of this equation as a sum of terms each of which involves a single pole. The resulting equation is

$$\frac{Y(s)}{U(s)} = \frac{\alpha_1}{s+\beta_1}+\frac{\alpha_2}{s+\beta_2}+.....+\frac{\alpha_n}{s+\beta_n} \qquad (22)$$

Each term on the right hand side of the equation has the same form and the corresponding signal flow graph involves a parallel structure as shown in Figure 6. The state variables are taken to be the outputs of each integrator. The state equations describing the signal flow graph are then of the form

$$\dot{x}_1 = -\beta_1x_1+u(t)$$

$$\dot{x}_2 = -\beta_2x_2+u(t)$$

$$\vdots$$

$$\dot{x}_n = -\beta_nx_n+u(t) \qquad (23)$$

An additional algebraic equation relates the output y to the n state variables. This has the form

$$y = \alpha_1x_1+\alpha_2x_2+.....\alpha_nx_n \qquad (24)$$

A model with a parallel structure having first-order (or second-order) blocks is generally more robust in terms of numerical sensitivity to coefficient inaccuracies than the equivalent model in direct construction form. The higher the order of the original transfer function description the greater are the benefits of the parallel approach.

6.3 The iterative construction approach

If a transfer function has only real poles and zeros the numerator and denominator may both be factorised into products of first-order factors to give a description of the following form:

$$\frac{K(s+\alpha_1)\,(s+\alpha_2)\,\ldots\,(s+\alpha_m)}{(s+\beta_1)\,(s+\beta_2)\,\ldots\,(s+\beta_n)} \qquad (25)$$

where K, α_i and β_j are all real constants. Grouping the factors into products of terms of the form

$$\frac{(s+\alpha_j)}{(s+\beta_j)} \qquad (26)$$

it is possible to construct a sub-model block diagram or signal flow graph which represents a single factor of the complete transfer function. This sub-model diagram has the form shown in Figure 7. Since the order of the denominator is always equal to or greater than the order of the numerator of the transfer function for physically realisable systems. Hence, when the process of grouping these factors in pairs has been completed, there may be some additional terms involving denominators of the form $(s + \beta_k)$ which cannot be combined with any numerator factors. In such cases the additional factors are dealt with by means of the sub-model signal flow graph element in Figure 8. The form of the diagram for the complete transfer function for the special case where the numerator is of the same order as the denominator is shown in Figure 9. The state variables are the output variables of the integrator blocks and the resulting state equations are as follows:

$$\dot{x}_n = -\beta_n x_n + u \qquad (27)$$

$$\dot{x}_{n-1} = -\beta_{n-1}x_{n-1} + \alpha_n x_n + \dot{x}_n = -\beta_{n-1}x_{n-1} + (\alpha_n - \beta_n)\,x_n + u \qquad (28)$$

. . . . . .

. . . . . .

$$\dot{x}_{n-j} = -\beta_{n-j}x_{n-j} + (\alpha_{n-j+1} - \beta_{n-j+1})\,x_{n-j+1} + \ldots\ldots + (\alpha_n - \beta_n)\,x_n + u \qquad (29)$$

. . . . . .

. . . . . .

$$\dot{x}_1 = -\beta_1 x_1 + (\alpha_2 - \beta_2)\,x_2 + \ldots\ldots\ldots + (\alpha_n - \beta_n)\,x_n + u \qquad (30)$$

As with the parallel construction approach the iterative method is often superior to the direct construction method in terms of coefficient sensitivity. A cascaded arrangement of first or second order transfer functions tends to be more robust to coefficient errors than the

equivalent structure involving multiple feedback loops.

6.4 An example of block diagram construction from a transfer function

Consider the following transfer function:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^2+9s+20}{s^3+6s^2+11s+6} \qquad (31)$$

Direct construction method

First of all the numerator and denominator polynomials on the right hand side of the equation should be divided by the highest power of s to give

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^{-1}+9s^{-2}+20s^{-3}}{1+6s^{-1}+11s^{-2}+6s^{-3}} \qquad (32)$$

Introducing a dummy variable E(s) gives

$$\frac{Y(s)}{E(s)} = s^{-1}+9s^{-2}+20s^{-3} \qquad (33)$$

and

$$\frac{E(s)}{U(s)} = \frac{1}{1+6s^{-1}+11s^{-2}+6s^{-3}} \qquad (34)$$

That is

$$E(s) = U(s) - (6s^{-1}E(s) + 11s^{-2}E(s) + 6s^{-3}E(s)) \qquad (35)$$

and

$$Y(s) = s^{-1}E(s) + 9s^{-2}E(s) + 20s^{-3}E(s) \qquad (36)$$

Equations (35) and (36) may be expressed in block diagram form by a model involving three integrators connected in cascade as shown in Figure 10. Taking the outputs of integrator blocks as state variables allows the following set of simultaneous first order differential equations to be established:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = x_3$$

$$\dot{x}_3 = -6x_1 - 11x_2 - 6x_3$$

or, in matrix notation,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \qquad (37)$$

The output equation is

$$y = 20x_1 + 9x_2 + x_3$$

or

$$y = [20 \;\; 9 \;\; 1] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad (38)$$

Parallel construction method

In this case the first step involves factorising the denominator of the given transfer function to give:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^2 + 9s + 20}{s^3 + 6s^2 + 11s + 6} = \frac{s^2 + 9s + 20}{(s+1)(s+2)(s+3)} \qquad (39)$$

Expressing the right hand side of this equation in partial fraction form gives:

$$\frac{Y(s)}{U(s)} = \frac{6}{s+1} - \frac{6}{s+2} + \frac{1}{s+3} \qquad (40)$$

and this may be represented as a block diagram with the parallel structure shown in Figure

9. Again the outputs of the integrator blocks are taken as the state variables to give the following set of first-order differential equations

$$\dot{x}_1 = -x_1 + u; \quad \dot{x}_2 = -2x_2 + u; \quad \dot{x}_3 = -3x_3 + u$$

and an algebraic equation relating the output, y(t), to the three state variables $x_1(t)$, $x_2(t)$ and $x_3(t)$ which has the form

$$y = -6x_1 - 6x_2 - x_3$$

In matrix form the state and output equations are thus

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u \qquad (41)$$

and

$$y = \begin{bmatrix} 6 & -6 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad (42)$$

Iterative construction method

There are two zeros and three poles in the given transfer function. Hence the two zeros can be associated with two of the poles in the standard way for the iterative approach and one pole must be treated separately, as shown in the transfer function below

$$\frac{Y(s)}{U(s)} = \frac{s^2 + 9s + 20}{s^3 + 6s^2 + 11s + 6} = \left(\frac{s+5}{s+1}\right)\left(\frac{s+4}{s+2}\right)\left(\frac{1}{s+3}\right) \qquad (43)$$

The resulting block diagram thus has two stages involving elements of the type shown in Figure 7 and one stage of the type shown in Figure 8. Figure 12 shows the complete block diagram. Taking outputs of integrator blocks as state variables gives a set of first order differential equations

$$\begin{aligned} \dot{x}_1 &= -3x_1 + 4x_2 + \dot{x}_2 \\ &= -3x_1 + 2x_2 + 4x_3 + u \qquad (44) \\ \dot{x}_2 &= -2x_2 + 5x_3 + \dot{x}_3 \end{aligned}$$

$$= -2x_2 - 4x_3 + u \qquad (45)$$

$$\dot{x}_3 = -x_3 + u \qquad (46)$$

and an algebraic output equation

$$y = x_1 \qquad (47)$$

7. Modelling of distributed parameter elements

7.1 Introduction

Distributed parameter models arise in the study of systems in which quantities are transmitted from one point to another, as in the case of electrical signals in a transmission line, flow of fluid in a pipe, phenomena associated with the conduction of heat etc.. The "signals" in such systems are distributed in space as well as time and the mathematical models are described by partial differential equations.

In situations in which the system reduces essentially to a single input variable and a single output variable of interest at one point in space, as often applies in control systems applications, it is possible to derive a transfer function type of description which involves a distributed parameter. Such distributed parameter transfer function models may then be used in the same way as conventional transfer functions for lumped parameter models. The two types of distributed parameter element considered in this section are the pure time delay and the distributed time delay.

7.2 Pure Time Delay

A pure time delay (known also as a transportation lag, or "dead time") arises in systems in which quantities are transmitted at finite velocity from one point to another. In the case of a pure delay this transmission does not involve any frequency dependent change of signal amplitude.

A system involving a pure time delay (due perhaps to physical phenomena such as heat flow but where losses can be neglected), may be described approximately by a partial differential equation of the form

$$\frac{\partial z}{\partial t} = v\frac{\partial z}{\partial x} \qquad (48)$$

where z(x,t) is the state of a point at position x and time t. The parameter v is the velocity of transmission.

Applying the Laplace transformation to this equation gives, assuming zero initial conditions, a new equation as follows

$$sZ(x,s) = v\frac{\partial Z(x,s)}{\partial x} \tag{49}$$

where

$$Z(x,s) = \int_0^\infty \exp(-st)\, z(x,t)\, dt \tag{50}$$

The solution of this equation is

$$Z(x,s) = A\exp(-s\frac{x}{v}) \tag{51}$$

where A is an arbitrary constant.

If U(s) is the Laplace transform of the input of the system at point x_1 and Y(s) is the transform of the output of the element at point x_2 then it follows that

$$U(s) = Z(x_1,s) = A\exp(-s\frac{x_1}{v}) \tag{52}$$

and

$$Y(s) = Z(x_2,s) = A\exp(-s\frac{x_2}{v}) \tag{53}$$

Thus the transfer function relating the output transform Y(s) to the input transform U(s) is

$$\frac{Y(s)}{U(s)} = \exp\left(-s\frac{x_2-x_1}{v}\right) \tag{54}$$

That is

$$\frac{Y(s)}{U(s)} = \exp(-sT) \tag{55}$$

where T is the transmission time from point x_1 to point x_2. The parameter T is thus the duration of the pure time delay.

7.3 Distributed time delay

A distributed time delay arises where quantities are transmitted from one point to another with a finite velocity but with an attenuation which varies with frequency. examples include the conduction of heat and the transmission of electrical signals in a medium which is not loss free. The partial differential equation describing this situation has the form

$$\frac{\partial z}{\partial t} = a \frac{\partial^2 z}{\partial x^2} \tag{56}$$

where z(x,t) represents the value of the quantity concerned at point x and time t. The parameter a is a constant and could represent, for example, thermal conductivity. Applying the unilateral Laplace transform for zero initial conditions gives

$$sZ(x,s) = a \frac{\partial^2 Z(x,s)}{\partial x^2} \tag{57}$$

where

$$Z(x,s) = \int_0^\infty \exp(-st)\, z(x,t)\, dt \tag{58}$$

The solution of this equation has the form

$$Z(x,s) = A\exp\left(-\sqrt{\frac{s}{a}}x\right) + B\exp\left(\sqrt{\frac{s}{a}}x\right) \tag{59}$$

It is now assumed that at infinite distance the value of the variable z(x,t) is always zero. This allows one of the terms in the above equation to be eliminated, giving

$$Z(x,s) = A\exp\left(-\sqrt{\frac{s}{a}}\right) \tag{60}$$

for $x \geq 0$.

If U(s) is the Laplace transform of the input to the system being modelled (at point x_1) and Y(s) is the Laplace transform of the output of the system (at point x_2) then it follows that

$$U(s) = Z(x_1,s) = A\exp\left(-\sqrt{\frac{s}{a}}x_1\right) \tag{61}$$

Similarly

$$Y(s) = Z(x_2,s) = A\exp\left(-\sqrt{\frac{s}{a}}x_2\right) \tag{62}$$

Hence the transfer function relating Y(s) to U(s) has the form

$$\frac{Y(s)}{U(s)} = \exp\left(-\sqrt{\frac{s}{a}}(x_2 - x_1)\right) \tag{63}$$

Thus

$$\frac{Y(s)}{U(s)} = \exp\left(-\sqrt{s\frac{(x_2 - x_1)^2}{a}}\right) = \exp(-\sqrt{sT}) \tag{64}$$

where the parameter T is an equivalent time.

It should be noted that elimination of the term involving the positive exponent in this derivation has physical significance. It is equivalent to rejecting the possibility of reflected waves in the system under consideration. The use of the transfer function resulting from this analysis is therefore restricted to cases in which reflected waves are not expected.

7.4 Simulation models involving pure and distributed delay elements

Most simulation languages incorporate facilities for the representation of pure delays. Pure delays also present few difficulties in a simulation which is developed using a general purpose high level language, provided the delay does not itself vary with time. Cases involving variable delays present additional problems [3]. An alternative approach, and more approximate representation for a pure delay, is based upon the properties of Padé approximations. These approximations, based upon truncated series for the exponential function, were widely used for the representation of pure delays in analog simulations and it is equally possible to use this approach in a digital simulation.

More complex problems involving distributed parameter elements can be approached in a very general way using finite differences or finite element methods or other numerical techniques for the solution of partial differential equations. Detailed treatments of these specialised topics can be found in appropriate textbooks and a review of some of the problems of simulation of such systemsmay be found in the book by Spriet and Vansteenkiste [4]. Simulations involving distributed parameter elements can be numerically intensive and for time-critical applications distributed parameter problems are often reduced to quite simple lumped-parameter approximations. One example of this type can be found in the work of Bryce et al. [5] which is concerned with the real-time simulation of a water pipeline as part of an investigation of water-turbine governing systems.

References

[1] Palm, W.J., "Control Systems Engineering", John Wiley & Sons, New York, U.S.A., 1986.

[2] Golten, J. and Verwer, A., "Control System Design and Simulation", McGraw-Hill, London, U.K., 1991.

[3] Doebelin, E.O. "System Modelling and Response", pp. 193-201, J. Wiley & Sons, New York, U.S.A., 1980.

[4] Spriet, J.A. and Vansteenkiste, G.C. "Computer-aided Modelling and Simulation", Academic Press, London, U.K., 1982.

[5] Bryce, G.W., Foord, T.R., Murray-Smith, D.J. and Agnew, P.W. 'Hybrid simulation of water-turbine governors', Simulation Councils Proceedings, Vol. 6, Part 1, pp. 35-44, 1976.

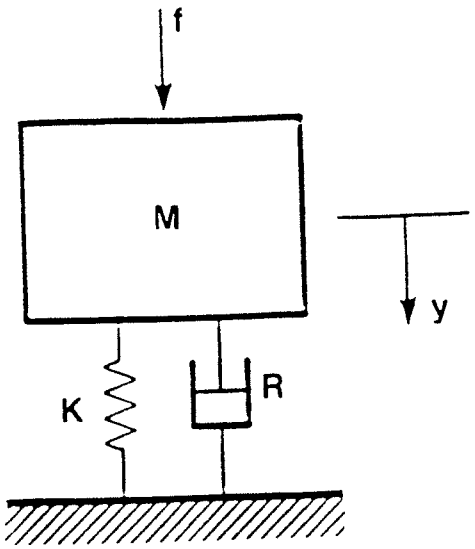


Figure 1: Mechanical system involving mass, spring and viscous damping elements.

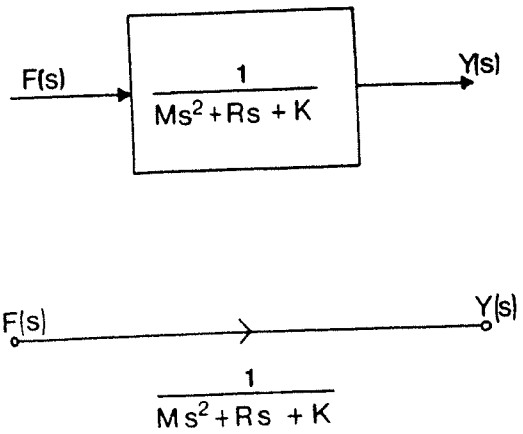


Figure 2: Transfer function based block diagram and signal flow graph for system of Figure 1.

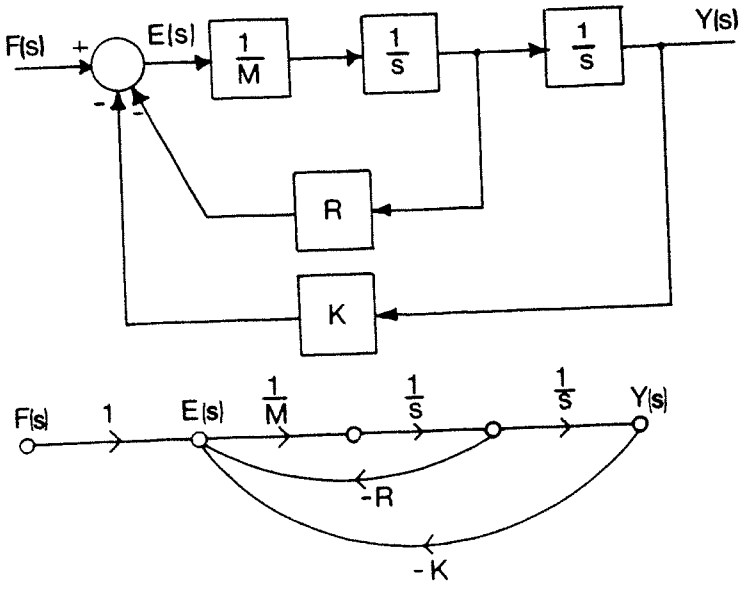


Figure 3: Detailed block diagram and signal flow graph for system of Figure 1.

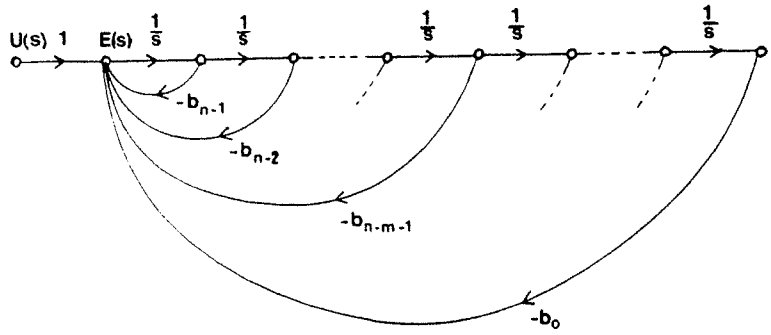


Figure 4: Direct construction signal flow graph for denominator terms.

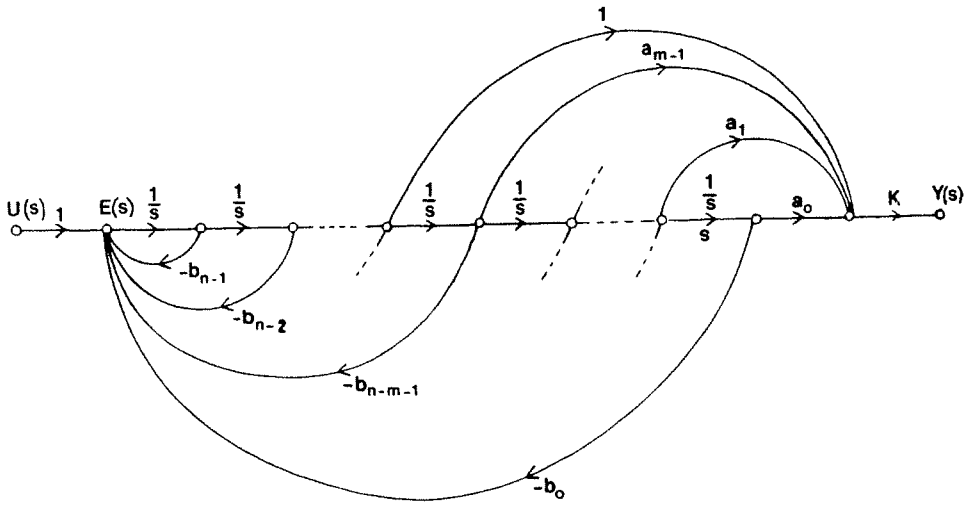


Figure 5: Complete signal flow graph for direct construction representation.

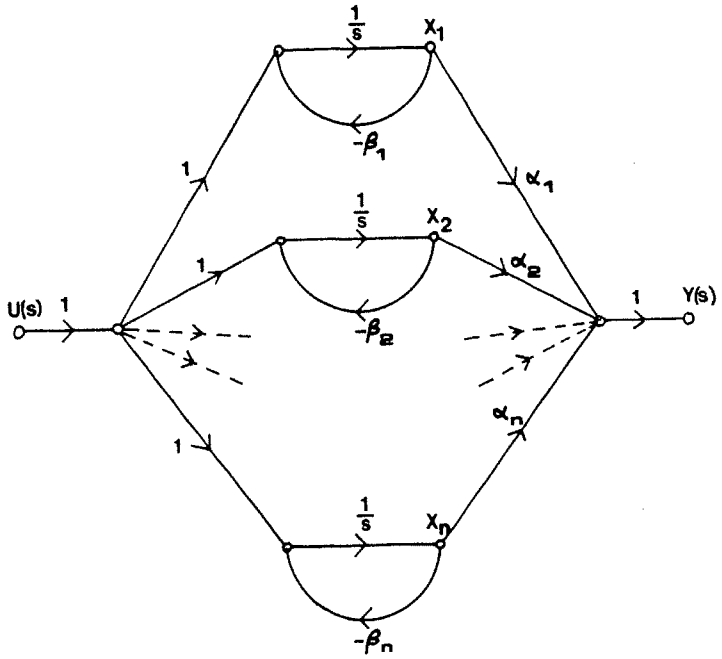


Figure 6: Signal flow graph illustrating parallel construction method.

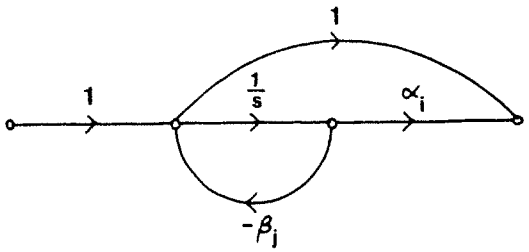


Figure 7: Signal flow graph for single pole-zero pair.

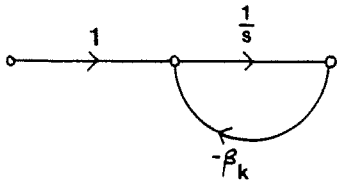


Figure 8: Signal flow graph for single pole.

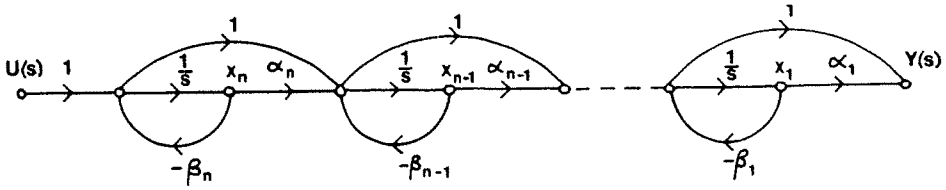


Figure 9: Signal flow graph illustrating iterative construction method.

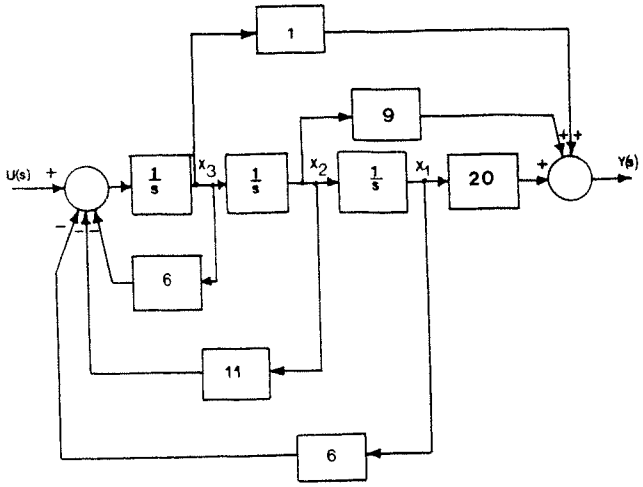


Figure 10: Block diagram for example system by direct construction method.

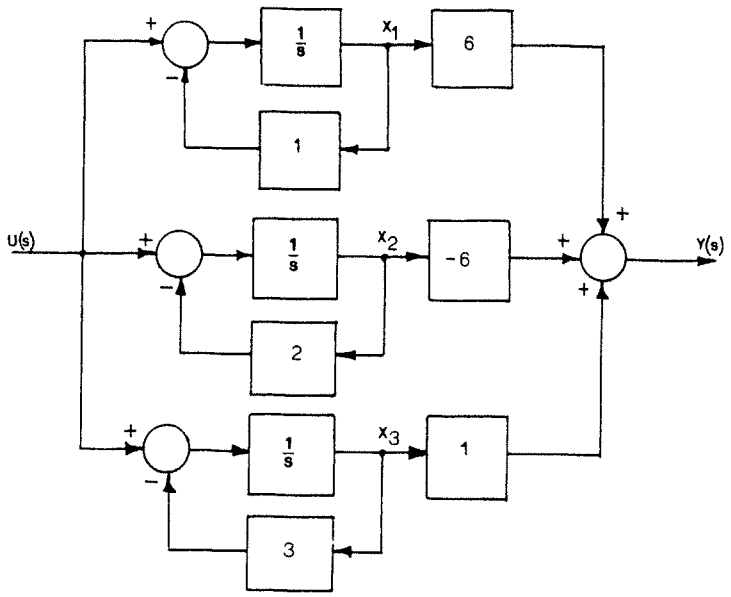


Figure 11: Block diagram for example system by parallel constructioin method.

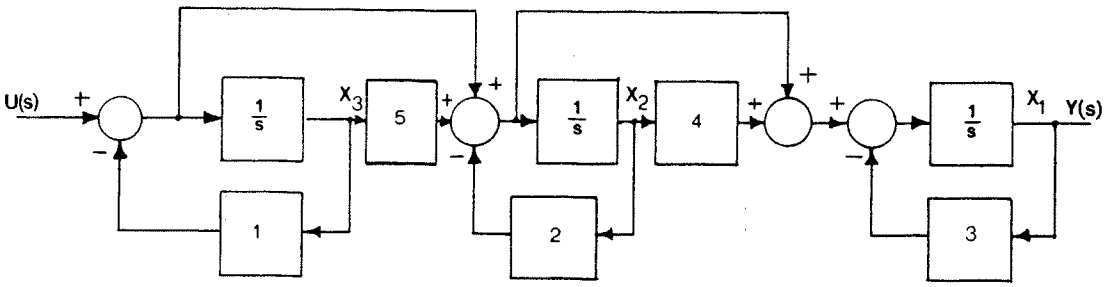


Figure 12: Block diagram for example system by iterative construction method.

Simple Examples of Control System Simulation Using Common Simulation Tools

1. Introduction

This section of the course provides examples of a number of different types of simulation problem involving control system applications. The models upon which the simulations are based are relatively simple and have been chosen to illustrate features of some typical simulation tools which are currently available. Further details of the SLIM simulation language may be found in the Users' Manual and in an associated textbook [1] which also provides some additional linear and nonlinear simulation examples. The underlying principles which can be applied both in simulation studies of systems developed using classical methods of control system design and systems designed using fuzzy control principles. Particular simulation techniques have been selected but this choice is not intended to suggest that other tools could not have provided an equally appropriate solution but strengths and weaknesses of some methods of approach are discussed. In each case suggestions are made of further investigations which could be made with these simple models.

2. A control system simulation: speed control of a water turbine

Control systems investigations form one of the most common application areas for continuous system simulation tools. This example is concerned with a simulation of a speed control system for a water turbine used for the generation of electricity. Figure 1 is a block diagram of a highly simplified description of a system of this kind. The turbine, which is of the impulse type, is represented by a linear transfer function which also incorporates dynamics of the pipeline system. Control of the turbine is accomplished through feedback of a signal proportional to turbine shaft speed and comparison with the desired (reference) speed signal. Any difference between the desired and actual speed provides an error signal which is processed by the controller block to provide the signal applied to the turbine actuator input to change the water flow in such a way that the error is continuously minimised. The controller may be implemented either in analogue (continuous) or digital form. The model is linear throughout, apart from backlash in the mechanical linkages associated with the turbine inlet actuator. The purpose of the model is to allow the performance of the model of the closed loop system to be investigated using both continuous and digital controllers. This is a useful illustration of continuous and discrete simulation techniques being used together to approach an important class of practical engineering problem.

A number of controller transfer functions can be considered for this type of application. Detailed consideration of possible forms of controller are not of primary interest in the present context but any readers interested in this aspect, or in the underlying model, may find further details elsewhere (e.g.[1-2]). For the present purposes the controller considered is of the "temporary droop" type implemented in continuous form. This type of controller has a transfer function of the form:-

$$C(s) = \frac{1 + sT_x}{\sigma + (\sigma + \mu)\,T_x s + T_x T_y s^2} \qquad (1)$$

In this controller transfer function only the parameters σ and μ may be regarded as adjustable quantities. The other parameters (T_x and T_y) are fixed quantities and are not available for controller tuning.

Appropriate parameter values for the plant and controller are as follows:

T_a = 7.0 sec. (inertial time constant)
T_w = 1.1 sec. (water time constant)
T_s = 0.2 sec. (actuator servo time constant)

All model variables, such as turbine speed are expressed as normalised (per unit) quantities.

Figure 2 shows the DYNAMIC segment of a SLIM program (TURB.SLI) for the simulation model and the full source may be found on the diskette. Parameter values for the nominal set of conditions are provided in the program listing. The simulation experiment in this case involves investigation of the response of the control system to a step change of the reference speed. It can be seen from Figure 3 that the response is stable but oscillatory in nature for the controller parameters values used. A program TURBI.SLI, which allows input from a data file, is also included in the Appendix to this section of the notes. This version of the simulation program, together with the appropriate data files in the same format as the test file TURBI.IN, provides a convenient means of experimenting with controller parameter values.

One interesting extension to this simulation program involves the inclusion of mechanical backlash between the servomotor and the turbine inlet. This is an important feature of real mechanical systems of this kind and is known to have a destabilising effect on the overall control system. Backlash is a double-valued form of nonlinearity which has a steady-state input-output characteristic of the form shown in Figure 3. It generally arises because of 'slack' in mechanical linkages and gears. With backlash present movement of the input in one direction produces a proportional movement of the output, but any reversal of the direction of the input will cause the output to stop before following the input again.

The representation of backlash and other similar double-valued nonlinearities which involve a form of "memory", such as hysteresis, can be difficult. One approach is to use principles first established for the modelling of backlash elements using analogue computers [3]. This involves the use of an integrator with feedback to provide the memory element. Figure 4 shows a block diagram for a representation of backlash by this type of method. Essentially the system works by comparing the integrator output, which is the output variable for the backlash element, with the input. The integrator input is controlled through comparator elements and logic blocks and when the input reverses direction (at the extremes of travel) the integrator input becomes zero. The integrator input remains zero until the input variable has changed by an amount equal to the width of the backlash element. It is then switched back to the output of the summing element. This causes the integrator output to remain fixed in value for a period of time and this corresponds to the flat top and bottom sections on the diagram showing the input-output characteristics for the backlash. Although presented here in block-diagram form the implementation of a backlash model of this kind is quite simple using the equation-oriented methods. Figure 5 shows the DYNAMIC segment of a SLIM program TURB2.SLI which is a version of the turbine speed control system model with backlash incorporated. This program and the corresponding data file TURB2.IN are listed in the Appendix.

The oscillation observed on the response shown in Figure 2 is influenced by the backlash parameter a_b as well as by the parameters of the controller. A well damped step response in the absence of backlash (a_b=0.0) can become a maintained oscillation if the backlash is increased sufficiently (Figure 6).

Figure 7 is an ACSL program listing for this example. Note that a special statement BCKLSH is available in ACSL to represent backlash effects and use has been made of this facility. Figure 8 shows results obtained from this ACSL program. In modern block-oriented tools such as XANALOG and SIMULINK special facilities are also available for the simulation of nonlinear elements such as backlash. Figure 9 shows an XANALOG block diagram for this problem.

Further investigations which could be carried out using this model and the SLIM simulation programs include determination of the minimum value of the backlash parameter b which gives rise to a maintained oscillation in terms of turbine speed. This "limit cycling" type of behaviour is clearly an undesirable situation in a speed control system and one that should be avoided in practice. For those with an appropriate level of understanding of control systems analysis techniques the simulation result for the critical value of the backlash parameter may be compared with predictions from theory, based upon describing function analysis. It is interesting to consider the reasons for any differences between the simulation result and the value predicted by theory. It should be remembered that the describing function approach involves some important simplifying assumptions and approximations; the significance of some of these can be investigated easily using the simulation.

3. Simulation of a simple digital control system

Figure 10 is a block diagram of a simple closed-loop digital control system. The difference between the reference signal and the plant output forms an error which is sampled by the digital-to-analogue converter. The digital processor carries out some form of numerical operation on the error samples and provides an actuating signal to the plant input through the digital-to-analogue converter. Figure 11 shows part of the corresponding SLIM program listing for the simplest possible situation in which the control computer simple samples the error signal and outputs the sampled error values periodically as input to the plant. The complete program file, named DIGCON.SLI, and the necessary input data file DIGCON.IN, can be found on the diskette. Examination of the DIGCON.SLI shows that, because SLIM does not have special facilities for mixed continuous and discrete system simulation, the sampling period is defined as a multiple of the communication interval parameter and the facilities of the DYNAMIC segment and DERIVATIVE section are used to emulate the discrete action of the controller. The sampled variable is held constant in a zero-order hold type of action by a loop within the DYNAMIC segment. Any discrete calculations representing the action of a control algorithm within the control computer must also be performed within the DYNAMIC segment. Values of variables of interest in the simulation may be written to the output file in the usual way, at times set by the communication interval.

Figure 12 shows results obtained for three different sampling periods. It can be seen that the system shows an increasing tendency to oscillate as the sampling period is increased and eventually becomes unstable. Sampled data theory (see, for example [14]) predicts that for this system instability occurs when the sampling period is greater than 0.549 sec.. This is consistent with the results presented in Figure 12 and detailed investigations using the

simulation program can confirm the theoretical result more precisely.

The DIGCON.SLI program can be modified easily to allow for more complex controller action. For example, if one wanted to simulate the system with a controller which implemented a difference equation of the form

$$O(kT) = O((k-1)T) + I(kT) - 0.5I((k-1)T) \qquad (2)$$

the changes to the simulation program would all be made in the initial part of the DYNAMIC segment. Figure 13 shows the relevant part of the listing and the complete program is listed in the Appendix as the file DIGCON1.SLI. Note how the discrete input and output variables are stored for one sample period and updated. The transfer function of the controller of Equation (2), expressed in terms of z transforms, is as follows:

$$\frac{O(z)}{I(z)} = \frac{1 - 0.5z^{-1}}{1 - z^{-1}} = \frac{z - 0.5}{z - 1} \qquad (3)$$

In the special case when the sample period is 0.347 sec. the controller should, from sampled-data theory, act as a "dead beat" compensator [4]. In such a situation the plant output should exhibit zero steady state error and should rise to its final value, in response to a step change of system reference input, in one sampling period. Figure 14 shows results from the simulation program which are consistent with theory for this special case.

Some other equation-oriented simulation languages, such as ACSL, include special facilities which can be very useful for the simulation of digital control systems. In ACSL, for example, DISCRETE sections representing the difference equations or z-transfer function of a digital controller may be inserted within the DYNAMIC segment. Such DISCRETE sections are thus similar to DERIVATIVE sections but communicate with the continuous parts of the simulation at regular predetermined times. Figures 15 and 16 show XANALOG and SIMULINK block diagrams for this digital control problem and illustrate some more of the specialised blocks and icons available with these simulation tools.

This example offers any reader interested in automatic control systems many opportunities for experimentation. It is clear from the listing of the SLIM program DIGCON1.SLI that, with some minor changes to the DYNAMIC segment, it would be very easy to replace the dead beat compensator by some other form of controller. Similarly any other form of plant transfer function could be used in place of the one given in Figure 10, with only some simple changes to the DERIVATIVE section of the program being necessary.

References

[1] Murray-Smith, D.J. "Continuous System Simulation", Chapman and Hall, London, 1995.

[2] Bryce, G.W., Foord, T.R., Murray-Smith, D.J. and Agnew, P., 'Hybrid simulation of water turbine governors', Simulation Council Proceedings, Vol.6, Part 1, pp. 35-44, 1976.

[3] Ricci, F.J. "Analog-Logic Computer Programming and Simulation", Spartan Books 1972.

[4] Leigh, J.R., "Applied Digital Control", Prentice-Hall Intl., Englewood Cliffs, N.J., U.S.A., 1984.

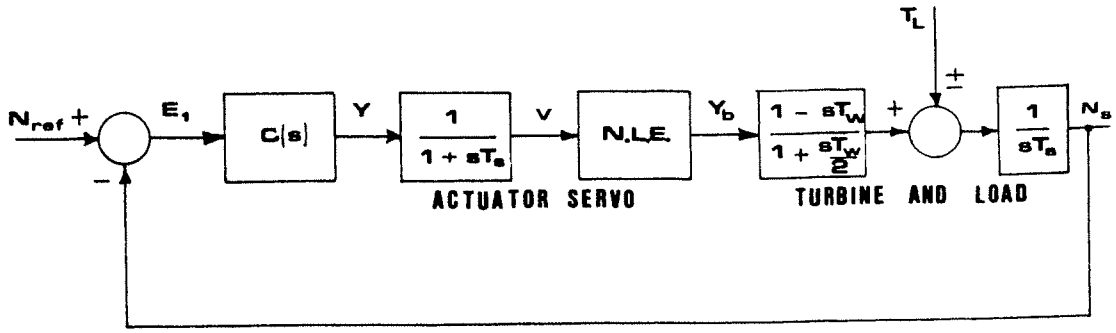


Figure 1: Block diagram of closed-loop system for automatic control of speed of water turbine connected to an electrical generator.

```
C
C  ******************Start of Dynamic Segment*************
C
      DYNAMIC
C
C  ******************Start of Derivative Segment***********
C
        DERIVATIVE
          DT1=(V-T1)*(2.0/TW)
          T1=INTEG(DT1,T10)
          T0=T1-TW*DT1
          TA=T0-TL
          DNS=TA/TIA
          ANS=INTEG(DNS,ANS0)
          E1=REF-ANS
          DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
          DY1=INTEG(DDY1,DDY10)
          Y1=INTEG(DY1,Y10)
          Y=Y1+TX*DY1
          DV=(Y-V)/TS
          V=INTEG(DV,V0)
        DERIVATIVE END
C
C  ********************End of Derivative Section*************
C
C  Values ot t, ref, tl and ans for current communication
C  interval output to results file
C
      TYPE T,REF,TL,ANS
C
C  Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10    DYNAMIC END
C
C  ********************End of Dynamic Segment***************
C
C  ************************Terminal Segment*****************
12    STOP
      END
```

Figure 2: Part of SLIM program TURB.SLI for simulation of the turbine speed control system with a temporary-droop type of governor.

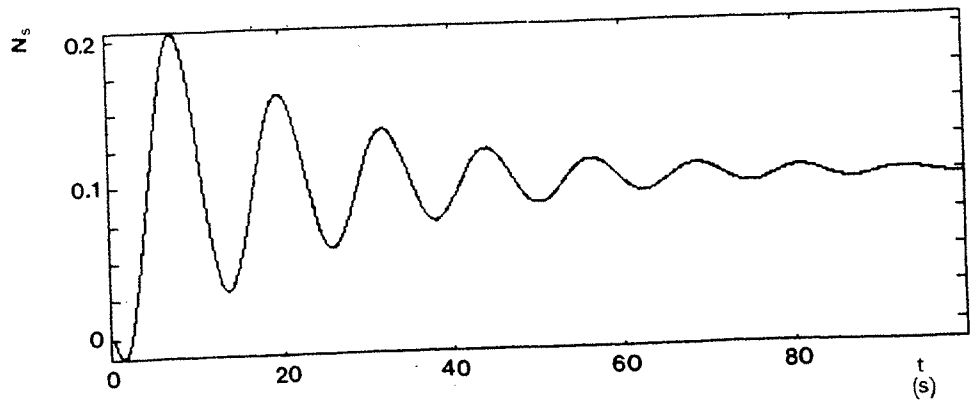


Figure 3: A typical simulated response of the speed control system to a step change in reference speed.

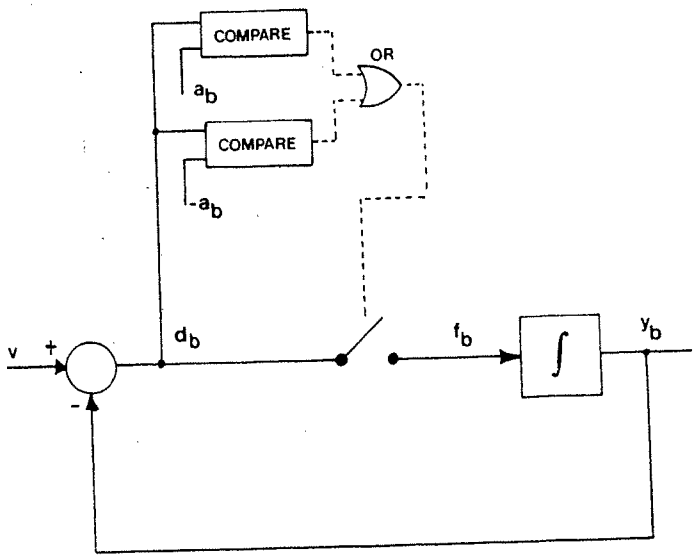


Figure 4: Block diagram illustrating one technique for representation of backlash element.

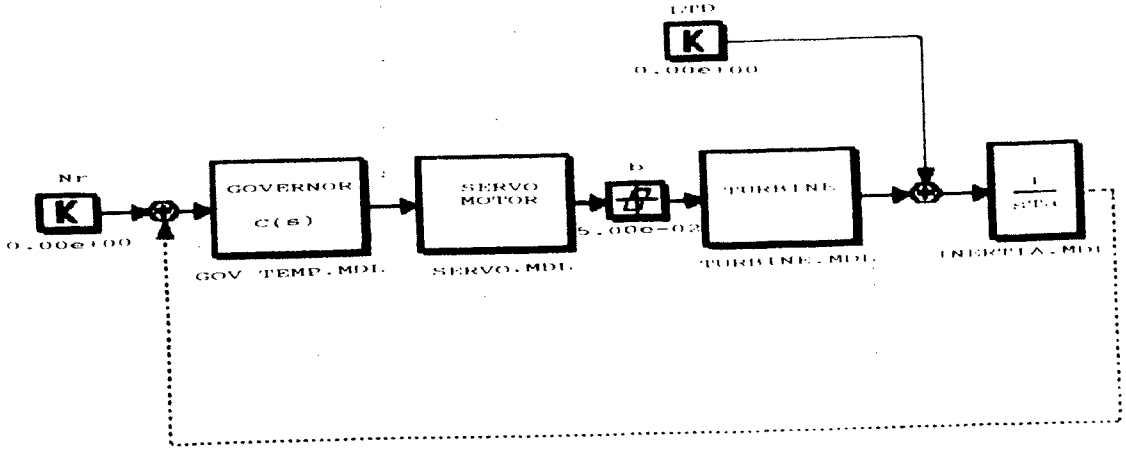


Figure 9: XANALOG block diagram of speed control problem.

```
**********Start of Dynamic Segment**********

   DYNAMIC

**********Start of Derivative Segment**********

      DERIVATIVE

Section for representation of backlash element

         IF(AB-0.001)40,45,45
         DB=V-YB
         IF(DB+AB)60,50,50
         FB=DB
         GOTO 100
         IF(DB-AB)80,70,70
         FB=DB
         GOTO 100
         FB=0.0
         YB=INTEG(10.0*FB,YB0)
         GOTO 110
         YB=V

End of section for backlash

         DT1=(YB-T1)*(2.0/TW)
         T1=INTEG(DT1,T10)
         T0=T1-TW*DT1
         TA=T0-TL
         DNS=TA/TIA
         ANS=INTEG(DNS,ANS0)
         E1=REF-ANS
         DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
         DY1=INTEG(DDY1,DDY10)
         Y1=INTEG(DY1,Y10)
         Y=Y1+TX*DY1
         DV=(Y-V)/TS
         V=INTEG(DV,V0)
      DERIVATIVE END

**********End of Derivative Section**********

Values of t, ref, tl, ans and yb for current communication
interval output to results file

   TYPE T,REF,TL,ANS,YB

Test for end of simulation run

   IF(T-TMAX)10,10,12
   DYNAMIC END

**********End of Dynamic Segment**********

**********Terminal Segment**********
   STOP
   END
```

Figure 5: Part of SLIM program TURB2.SLI for simulation of the turbine speed control system with a temporary-droop type of governor and backlash between the servomotor and the turbine inlet valve.

```
PROGRAM turb2.csl
          "Simulation of governed hydro-turbine system with"
          "a temporary-droop governor and with backlash between"
          "servo-motor and turbine inlet actuator"
    INITIAL
          "Data for plant and governor transfer functions"
        CONSTANT tw=1.1, ts=0.2, tia=7.0
        CONSTANT ab=0.06  $  "backlash parameter"
        CONSTANT tx=16.0, ty=0.3, sig=0.03, mu=0.25
          "Data for reference input and disturbance input"
        CONSTANT ref=0.1, tl=0.0
          "Data for experiment duration"
        CONSTANT tend=99.9
        ALGORITHM IALG=4    $ "Runge-Kutta second order"
        CINTERVAL CINT=1.0
          "Initial conditions"
        t1=0.0
        v=0.0
        y1=0.0
        dy1=0.0
        yb0=0.0
    END     $ "of INITIAL"

    DYNAMIC
      DERIVATIVE
            "Turbine"
          dt1=(yb-t1)*(2.0/tw)
          t1=INTEG(dt1,0.0)
          t0=t1-tw*dt1
            "Load"
          ta=t0+t1
          dns=ta/tia
          ns=INTEG(dns,0.0)
            "Governor"
          e1=ref-ns
          ddy1=(e1-sig*y1-((sig+mu)*tx+ty)*dy1)/(tx*ty)
          dy1=INTEG(ddy1,0.0)
          y1=INTEG(dy1,0.0)
          y=y1+tx*dy1
            "Servo motor"
          dv=(y-v)/ts
          v=INTEG(dv,0.0)
          yb=BCKLSH(yb0,ab,v)
      END    $  "of DERIVATIVE"
    TERMT (t.GE.tend)
    END      $  "of DYNAMIC"
    TERMINAL
    END      $  "of TERMINAL"
END      $  "of PROGRAM"
```

Figure 7: ACSL Program listing for the speed control problem.

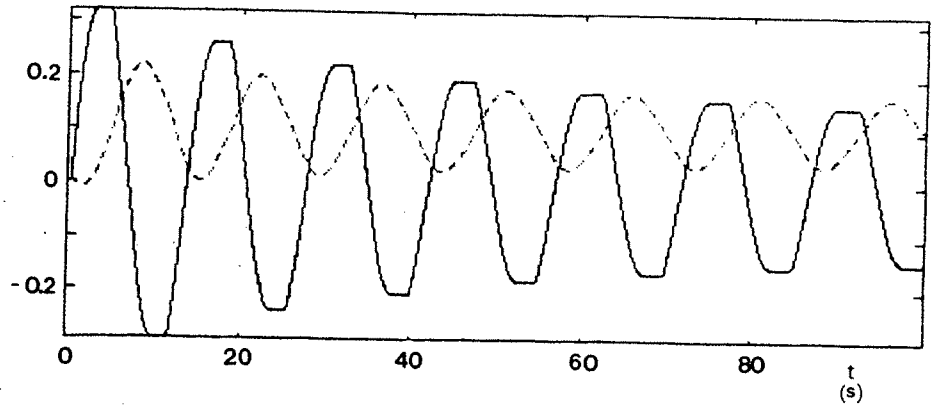


Figure 6: Response of the simulated speed control system with sufficient backlash to cause limit cycle oscillations. The continuous line is the control valve position while the dashed line is the output speed.

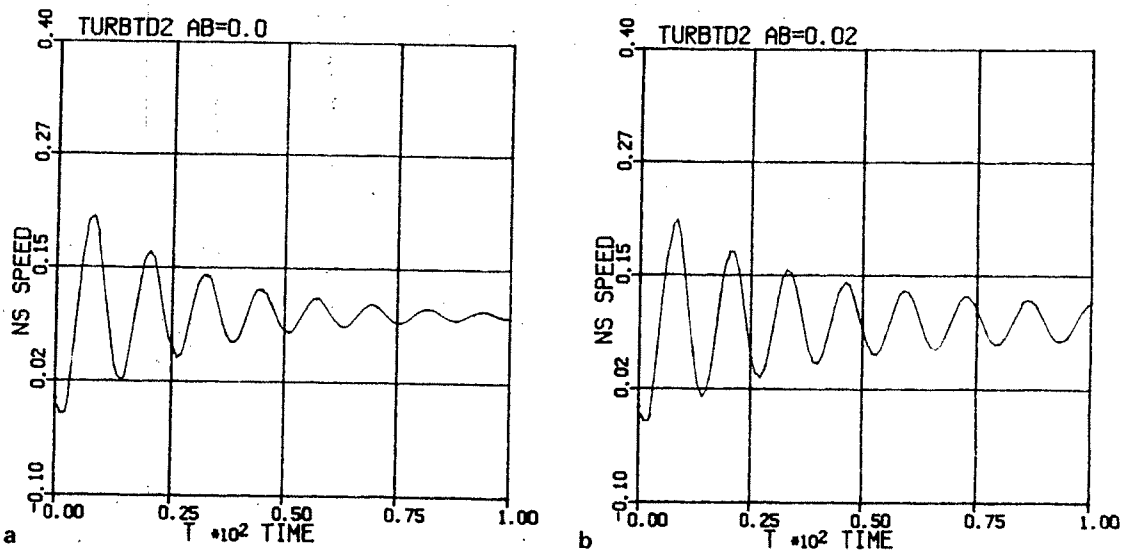


Figure 8: Results from ACSL simulation, a)with no backlash and b)with backlash of 0.02

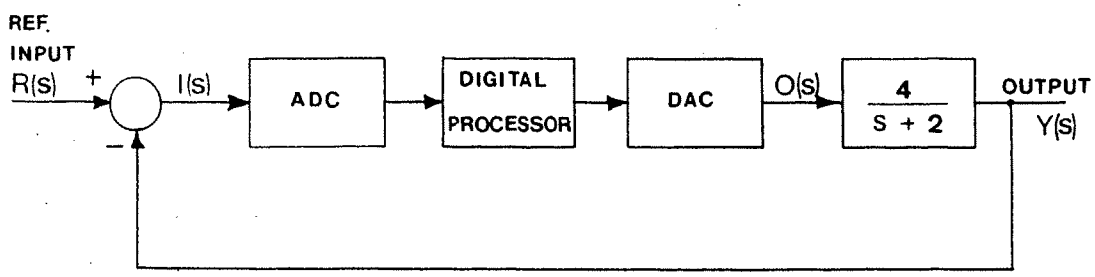


Figure 10: Block diagram of simple digital control system

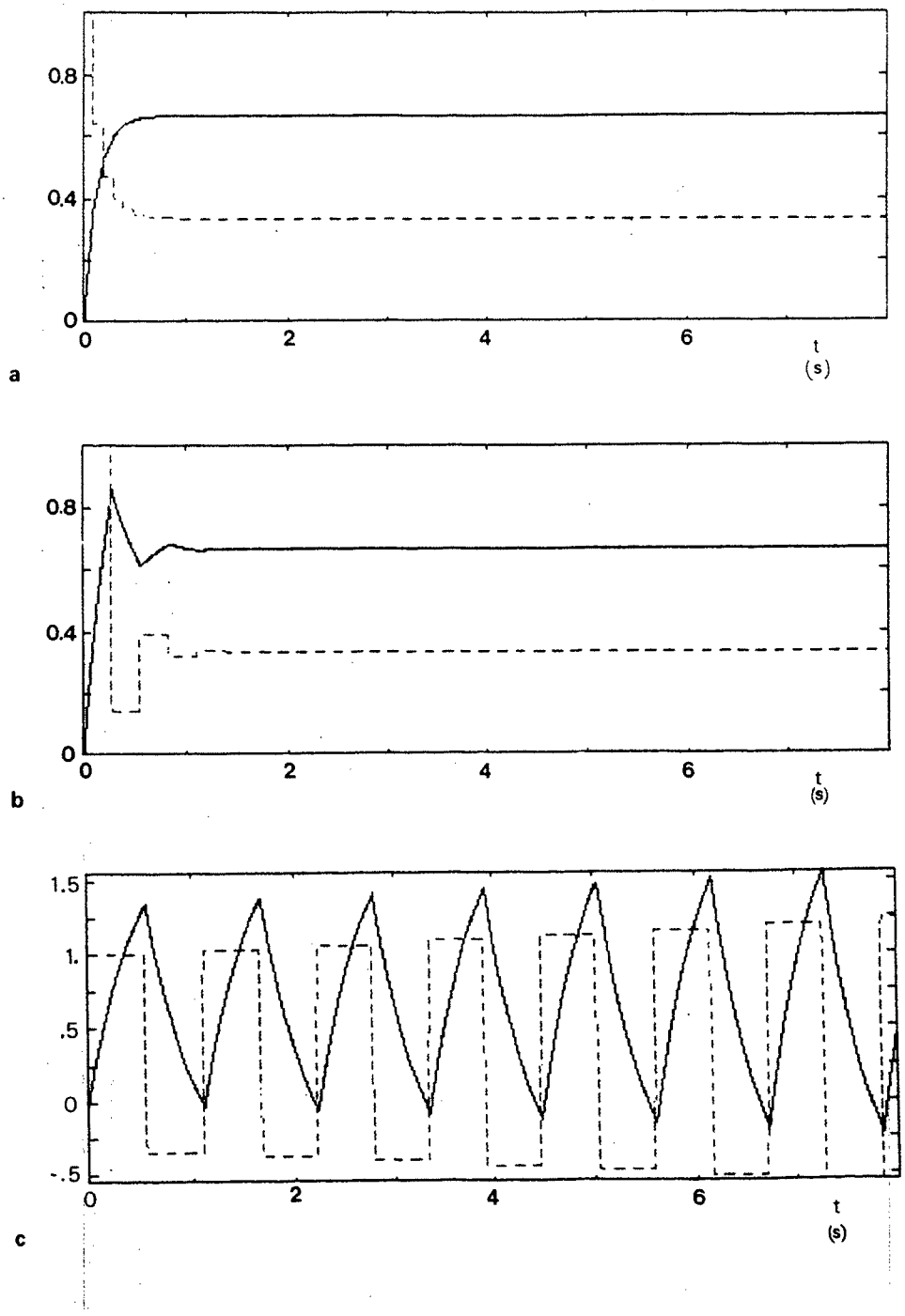


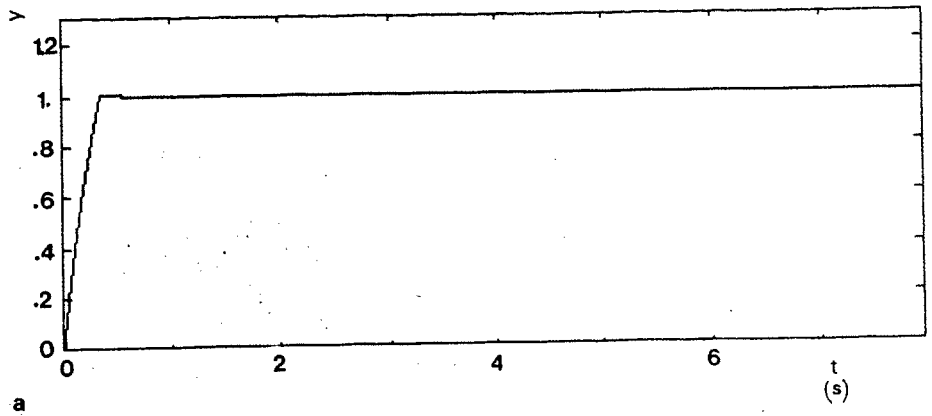
Figure 12: Responses {output (continuous line) and error (dashed line)} of digital control system for 3 values of controller sampling interval (a=0.1s, b=0.28s, c=0.56s).

```
    DYNAMIC
      I=I+1
      IF(I)15,25,15
      IF(I-MULT)20,25,25
      E=R-Y
      AIN=E
      OUT=OUT1+AIN-0.5*AIN1
      OUT1=OUT
      AIN1=AIN
      I=0
      TYPE T,V,E,Y

 ****************Start of Derivative Section*************

     DERIVATIVE
        DERIV=-(Y-AK*OUT)/TAW
        Y=INTEG(DERIV,0.0)
     DERIVATIVE END

*****************End of Derivative Section*****************

 Test for end of simulation run

    IF(T-TMAX)10,10,12
    DYNAMIC END

 *****************End of Dynamic Segment********************

 ********************Terminal Segment***********************

    STOP
    END
```

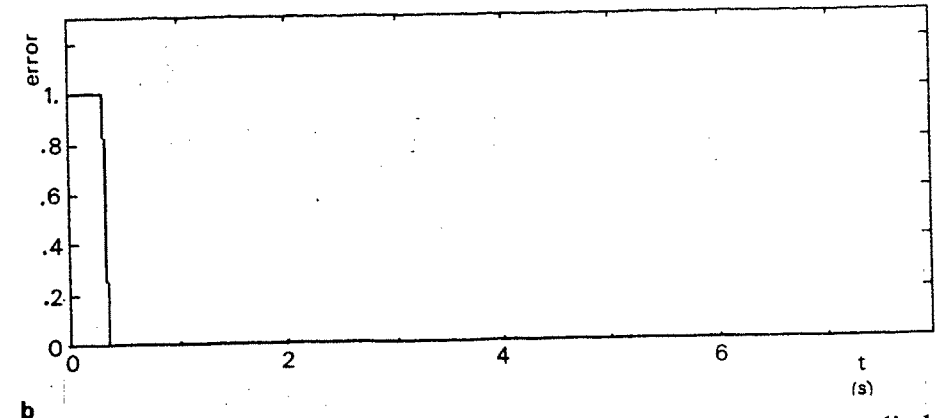
Figure 13: Part of SLIM program DIGCON1.SLI for simulation of digital control system.

```
C
C  ********************Start of Dynamic Segment***************
C
      DYNAMIC
        I=I+1
        IF(I)15,25,15
15      IF(I-MULT)20,25,25
25      AIN=R-Y
        OUT=AIN
        I=0
20      TYPE T,R,AIN,Y
C
C  ******************Start of Derivative Section**************
C
       DERIVATIVE
          DERIV=-(Y-AK*OUT)/TAW
          Y=INTEG(DERIV,0.0)
       DERIVATIVE END
C
C********************End of Derivative Section*****************
C
C  Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10    DYNAMIC END
C
C  ******************End of Dynamic Segment*********************
C
C  ***********************Terminal Segment***********************
C
12    STOP
      END
```

Figure 11: Part of SLIM program DIGCON.SLI for simulation of digital control system.



a



b

Figure 14: Responses from system with dead-beat control applied for step change of reference

```
    DYNAMIC
      I=I+1
      IF(I)15,25,15
      IF(I-MULT)20,25,25
      E=R-Y
      AIN=E
      OUT=OUT1+AIN-0.5*AIN1
      OUT1=OUT
      AIN1=AIN
      I=0
      TYPE T,V,E,Y

 ******************Start of Derivative Section**************

     DERIVATIVE
        DERIV=-(Y-AK*OUT)/TAW
        Y=INTEG(DERIV,0.0)
     DERIVATIVE END

******************End of Derivative Section*****************

 Test for end of simulation run

    IF(T-TMAX)10,10,12
    DYNAMIC END

 ******************End of Dynamic Segment*********************

 **********************Terminal Segment***********************

    STOP
    END
```

Figure 13: Part of SLIM program DIGCON1.SLI for simulation of digital control system.

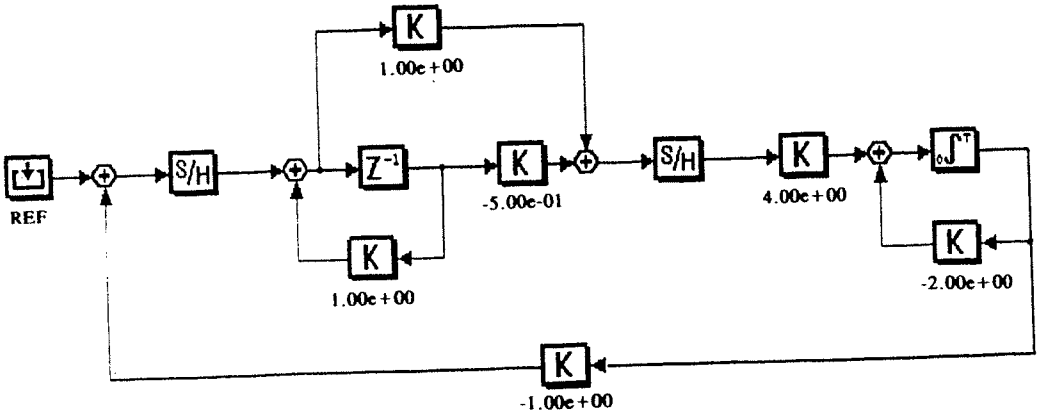


Figure 15: XANALOG block diagram for digital control system.

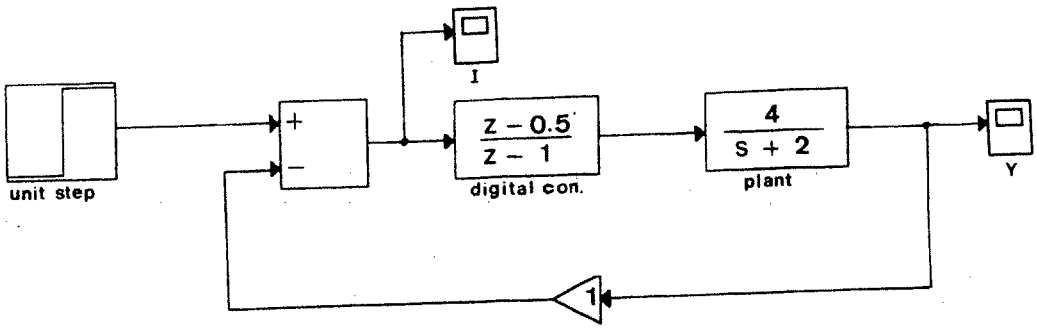


Figure 16: SIMULINK diagram for digital control system.

APPENDIX

Listings of SLIM Programs for Examples.

```
C
C  Simulation of hydro-turbine speed control system with temporary
C  droop type of governor. No backlash included in turbine model.
C
C  **********Initial Segment**********
C
C  First three lines of the program establish the number of
C  channels, the number of samples and the time interval
C  for outputs to the results file
      NCH=4
      NSA=400
      SAM=0.25
C
C  Parameter values for coefficients in the system equations
C
      TW=1.1
      TS=0.2
      TIA=7.0
      TX=16.0
      TY=0.3
      SIG=0.03
      AMU=0.25
C
C  Data for reference and disturbance inputs
C
      REF=0.1
      TL=0.0
C
C  Set communication interval and values for integration
C  parameters
C
      CINTERVAL(SAM)
      MINTERVAL(1.0E-6)
      MERROR(1.0E-6)
      XERROR(1.0E-6)
C
C  Set initial conditions
C
      T=0.0
      T10=0.0
      V0=0.0
      Y10=0.0
      DY10=0.0
      ANS0=0.0
C
C  Define the maximum value of the independent variable (time t)
C
      TMAX=100.0
C
C  Information about file organisation written to the results
C  file
C
      TYPE NCH
      TYPE NSA
      TYPE SAM
C
C  Information about initial values of variables written to
C  results file
C
      TYPE T,REF,TL,ANS0
```

```
C
C  ******************Start of Dynamic Segment**************
C
      DYNAMIC
C
C  ******************Start of Derivative Segment***********
C
        DERIVATIVE
          DT1=(V-T1)*(2.0/TW)
          T1=INTEG(DT1,T10)
          T0=T1-TW*DT1
          TA=T0-TL
          DNS=TA/TIA
          ANS=INTEG(DNS,ANS0)
          E1=REF-ANS
          DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
          DY1=INTEG(DDY1,DDY10)
          Y1=INTEG(DY1,Y10)
          Y=Y1+TX*DY1
          DV=(Y-V)/TS
          V=INTEG(DV,V0)
        DERIVATIVE END
C
C  ********************End of Derivative Section**************
C
C  Values ot t, ref, tl and ans for current communication
C  interval output to results file
C
      TYPE T,REF,TL,ANS
C
C  Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10    DYNAMIC END
C
C  **********************End of Dynamic Segment***************
C
C  ************************Terminal Segment*******************
12    STOP
      END
```

```
C
C  Simulation of hydro-turbine speed control system with temporary
C  droop type of governor. No backlash included in turbine model.
C  Typical data from file TURB1.IN.
C
C  ********************Initial Segment**********************
C
C  First three lines of the program establish the number of
C  channels, the number of samples and the time interval
C  for outputs to the results file
C
      NCH=4
      NSA=400
      SAM=0.25
C
C  Parameter values for coefficients in the system equations
C
      ACCEPT TW
      ACCEPT TS
      ACCEPT TIA
      ACCEPT TX
      ACCEPT TY
      ACCEPT SIG
      ACCEPT AMU
C
C  Data for reference and disturbance inputs
C
      ACCEPT REF
      ACCEPT TL
C
C  Set communication interval and values for integration
C  parameters
C
      CINTERVAL(SAM)
      MINTERVAL(1.0E-6)
      MERROR(1.0E-6)
      XERROR(1.0E-6)
C
C  Set initial conditions
C
      T=0.0
      T10=0.0
      V0=0.0
      Y10=0.0
      DY10=0.0
      ANS0=0.0
C
C  Define the maximum value of the independent variable (time t)
C
      TMAX=100.0
C
C  Information about file organisation written to the results
C  file
C
      TYPE NCH
      TYPE NSA
      TYPE SAM
C
C  Information about initial values of variables written to
C  results file
```

```
C
      TYPE T,REF,TL,ANS0
C
C  ******************Start of Dynamic Segment**************
C
      DYNAMIC
C
C  ******************Start of Derivative Segment***********
C
         DERIVATIVE
           DT1=(V-T1)*(2.0/TW)
           T1=INTEG(DT1,T10)
           T0=T1-TW*DT1
           TA=T0-TL
           DNS=TA/TIA
           ANS=INTEG(DNS,ANS0)
           E1=REF-ANS
           DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
           DY1=INTEG(DDY1,DDY10)
           Y1=INTEG(DY1,Y10)
           Y=Y1+TX*DY1
           DV=(Y-V)/TS
           V=INTEG(DV,V0)
         DERIVATIVE END
C
C  ********************End of Derivative Section*************
C
C  Values ot t, ref, tl and ans for current communication
C  interval output to results file
C
      TYPE T,REF,TL,ANS
C
C  Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10    DYNAMIC END
C
C  **********************End of Dynamic Segment***************
C
C  ************************Terminal Segment*******************
12    STOP
      END
```

```
C
C  Simulation of hydro-turbine speed control system with
C  temporary droop type of governor. Backlash included
C  in turbine model. Input data from file TURB2.IN.
C
C  **********************Initial Segment*******************
C
C  First three lines of the program establish the number of
C  channels, the number of samples and the time interval
C  for outputs to the results file
      NCH=5
      NSA=400
      SAM=0.25
C
C  Input parameter values for coefficients in the system equations
C  (from data file)
C
      ACCEPT TW,TS,TIA
      ACCEPT AB
      ACCEPT TX,TY
      ACCEPT SIG,AMU
C
C  Input data for reference and disturbance inputs
C
      ACCEPT REF,TL
C
C  Set communication interval and values for integration
C  parameters
C
      CINTERVAL(SAM)
      MINTERVAL(1.0E-5)
      MERROR(1.0E-4)
      XERROR(1.0E-4)
C
C  Set initial conditions
C
      T=0.0
      T10=0.0
      V0=0.0
      Y10=0.0
      DY10=0.0
      YB0=0.0
      ANS0=0.0
C
C  Define the maximum value of the independent variable (time t)
C
      TMAX=100.0
C
C  Information about file organisation written to the results
C  file
C
      TYPE NCH
      TYPE NSA
      TYPE SAM
C
C  Information about initial values of variables written to
C  results file
C
      TYPE T,REF,TL,ANS0,YB0
C
```

```
C  ********************Start of Dynamic Segment**************
C
      DYNAMIC
C
C  ******************Start of Derivative Segment***********
C
         DERIVATIVE
C
C  Section for representation of backlash element
C
           IF(AB-0.001)40,45,45
45         DB=V-YB
           IF(DB+AB)60,50,50
60         FB=DB
           GOTO 100
50         IF(DB-AB)80,70,70
70         FB=DB
           GOTO 100
80         FB=0.0
100        YB=INTEG(10.0*FB,YB0)
           GOTO 110
40         YB=V
C
C  End of section for backlash
C
110        DT1=(YB-T1)*(2.0/TW)
           T1=INTEG(DT1,T10)
           T0=T1-TW*DT1
           TA=T0-TL
           DNS=TA/TIA
           ANS=INTEG(DNS,ANS0)
           E1=REF-ANS
           DDY1=(E1-SIG*Y1-((SIG+AMU)*TX+TY)*DY1)/(TX*TY)
           DY1=INTEG(DDY1,DDY10)
           Y1=INTEG(DY1,Y10)
           Y=Y1+TX*DY1
           DV=(Y-V)/TS
           V=INTEG(DV,V0)
         DERIVATIVE END
C
C  ********************End of Derivative Section*************
C
C  Values of t, ref, tl, ans and yb for current communication
C  interval output to results file
C
      TYPE T,REF,TL,ANS,YB
C
C  Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10    DYNAMIC END
C
C  **********************End of Dynamic Segment***************
C
C  ************************Terminal Segment*****************
12    STOP
      END
```

```
1.1
0.2
7.0
16.0
0.3
0.03
0.25
0.1
0.0
```

Data file TURB2.IN

```
1.1
0.2
7.0
0.02
16.0
0.3
0.03
0.25
0.1
0.0
```

```
Simulation of a simple digital control system.
Input data from file DIGCON.IN.

**********************Initial Segment***************

First three lines of the program establish the number
of channels, the number of samples and the time interval
for outputs to the results file

   NCH=4
   NSA=400
   ACCEPT SAM

Input parameter value for coefficient in the system equations
(from data file)

   ACCEPT AK
   ACCEPT TAW

Read in value of reference

    ACCEPT R

Set initial value of the independent variable (time t)

   T=0.0
   AIN=0.0
   I=-1
   Y=0.0
   OUT=0.0

Define the maximum value of the independent variable (time t)

   ACCEPT TMAX

Set interval for output of variables to ouput file and
read in integer value representing the multiple
of this interval for sampling interval for digital control.
Also set values for integration parameters

   CINTERVAL(SAM)
   ACCEPT MULT
   MINTERVAL(1.0E-6)
   MERROR(1.0E-4)
   XERROR(1.0E-4)

Information about file organisation written to the results
file

   TYPE NCH
   TYPE NSA
   TYPE SAM


*******************Start of Dynamic Segment***************

   DYNAMIC
     I=I+1
     IF(I)15,25,15
```

```
15       IF(I-MULT)20,25,25
25       AIN=R-Y
         OUT=AIN
         I=0
20       TYPE T,R,AIN,Y
C
C  *****************Start of Derivative Section**************
C
       DERIVATIVE
          DERIV=-(Y-AK*OUT)/TAW
          Y=INTEG(DERIV,0.0)
       DERIVATIVE END
C
C*******************End of Derivative Section*****************
C
C  Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10    DYNAMIC END
C
C  *****************End of Dynamic Segment*******************
C
C  **********************Terminal Segment**********************
C
12    STOP
      END
```

```
C
C  Simulation of a simple digital control system with "dead-
C  beat" compensation.
C  Input data from file DIGCON1.IN.
C
C  **********************Initial Segment***************
C
C  First three lines of the program establish the number
C  of channels, the number of samples and the time interval
C  for outputs to the results file
C
      NCH=4
      NSA=230
      ACCEPT SAM
C
C  Input parameter value for coefficient in the system equations
C  (from data file)
C
      ACCEPT AK
      ACCEPT TAW
C
C  Read in value of reference
C
       ACCEPT R
C
C  Set initial value of the independent variable (time t)
C
      T=0.0
      E=R
      I=-1
      OUT=0.0
      AIN1=0.0
      OUT1=0.0
      Y=0.0
      V=1.3
C
C  Define the maximum value of the independent variable (time t)
C
      ACCEPT TMAX
C
C  Set interval for output of variables to ouput file and
C  read in integer value representing the multiple
C  of this interval for sampling interval for digital control.
C  Also set values for integration parameters
C
      CINTERVAL(SAM)
      ACCEPT MULT
      MINTERVAL(1.0E-6)
      MERROR(1.0E-4)
      XERROR(1.0E-4)
C
C  Information about file organisation written to the results
C  file
C
      TYPE NCH
      TYPE NSA
      TYPE SAM
C
C
C  ******************Start of Dynamic Segment***************
```

```
C
      DYNAMIC
        I=I+1
        IF(I)15,25,15
15      IF(I-MULT)20,25,25
25      E=R-Y
        AIN=E
        OUT=OUT1+AIN-0.5*AIN1
        OUT1=OUT
        AIN1=AIN
        I=0
20      TYPE T,V,E,Y
C
C  ******************Start of Derivative Section**************
C
       DERIVATIVE
          DERIV=-(Y-AK*OUT)/TAW
          Y=INTEG(DERIV,0.0)
       DERIVATIVE END
C
C*******************End of Derivative Section*****************
C
C  Test for end of simulation run
C
      IF(T-TMAX)10,10,12
10    DYNAMIC END
C
C  ******************End of Dynamic Segment*********************
C
C  **********************Terminal Segment***********************
C
12    STOP
      END
```

Data file DIGCON.IN

```
0.02
2.0
0.5
1.0
8.0
14
```

Data file DIGCON1.IN

```
0.0347
2.0
0.5
1.0
8.0
10
```

Internal Verification and External Validation of Simulation Models for Control Systems Analysis and Design

Topics considered include the following:

a) Terminology and definitions concerning simulation model verification validation.

Useful sources of information include a set of guidelines published in 1979 by the Technical Committee on Model Credibility of the Society for Computer Simulation [1] and various textbooks (e.g. [2-4]). An important distinction may be made between *internal verification* and *external validation* [2,5].

b) Internal Verification

The processes of internal verification involve checks for:

i) Internal consistency

and ii) Algorithmic validity

Appropriate methods involve *static checks* and *dynamic checks.*

c) External Validation

Criteria for external validation involve assessment of accuracy and suitability of a model for the intended application. They may include [6]:

Theoretical validity (consistent with accepted theories)

Empirical validity (adequate agreement with real system)

Pragmatic validity (satisfies application's requirements)

Heuristic validity (basis for explanation of system)

Empirical validation involves comparisons between behaviour of the model and behaviour of the real system. This can involve a number of methods including:

(i) Methods based on comparisons of response data (e.g. [4],[7])

(ii) Methods based on system identification methods (e.g. [7-12])

(iii) Methods based on sensitivity analysis (e.g. [4],[7])

(iv) Methods based on inverse models (e.g. [10])

Of the above four approaches to empirical validation methods (i) and (ii) are the most widely

used at present.

d) Robustness issues in external validation

This is especially important when system identification techniques are used as part of the external validation process. One must have confidence in the accuracy and reliability of the tools being used. These issues are explored in detail elsewhere (e.g. [13])

e) Possible outcomes of the external validation process

At least three possible outcomes arise. These are:

1. The measured data sets from the real system cannot be explained by any model structure and parameter set considered. The model structure and assumptions must be reviewed.

2. One or more models gives a satisfactory match to system response data but the uncertainty level for some parameters is high. The model may not be of much predictive value.

3. Satisfactory agreement is obtained with experimental test results and model parameter values are plausible. The model may be used for the intended application until new evidence falsifies the model in some way.

f) Documentation of the validation process

Model documentation should include the following:

1. A clear statement of the purpose of the model.

2. Descriptions of the model in conceptual and mathematical terms, including basic assumptions.

3. A statement concerning the range of conditions for which the model has been tested and the level of agreement throughout that range.

4. A description of all tests used for internal verification and external validation with relevant results.

References

[1] S.C.S. Technical Committee on Model Credibility, 'Terminology for model credibility', Simulation, Vol. 32, pp. 103-4, 1979.

[2] Shannon, R.E., "System Simulation. The Art and the Science", Prentice-Hall, Englewood Cliffs, N.J., U.S.A., 1975.

[3] Spriet, J.A. and Vansteenkiste, G.C. "Computer-Aided Modelling and Simulation", Academic Press, London, U.K., 1982.

[4] Murray-Smith, D.J. "Continuous System Simulation", Chapman & Hall, London, 1995.

[5] Murray-Smith, D.J., 'A review of methods for the validation of continuous system simulation models', in Nock, K.G. (editor), "Proceedings 1990 UKSC Conference on Computer Simulation", pp. 108-111, United Kingdom Simulation Council, Burgess Hill, 1990.

[6] Murray-Smith, D.J. and Carson, E.R., 'The modelling process in respiratory medicine', in Cramp, D.G. and Carson, E.R. (editors), "The Respiratory System", pp. 296-333, Croom Helm, London, 1988.

[7] Murray-Smith, D.J. Advances in simulation model validation: theory, software and applications, in Proceedings EUROSIM'95 Congress, Wien, September 1995.

[8] Unbehauen, H. and Rao, G.P., "Identification of Continuous Systems", North-Holland, Amsterdam, The Netherlands, 1987.

[9] Beck, J.V. and Arnold, K.J., "Parameter Estimation in Engineering and Science", John Wiley & Sons, New York, U.S.A., 1977.

[10] Bradley, R., Padfield, G.D., Murray-Smith, D.J. and Thomson, D.G., 'Validation of Helicopter Mathematical Models', Transactions of Institute of Measurement and Control, Vol. 12, pp. 186-196, 1990.

[11] AGARD Advisory Report No. 280, "Rotorcraft System Identification", AGARD, Neuilly sur Seine, France, 1991.

[12] Sinha, N.K. and Kuszta, B., "Modeling and Identification of Dynamic Systems", Van Nostrand Reinhold Company, New York, U.S.A., 1983.

[13] Murray-Smith, D.J., 'Modelling and robustness issues in rotorcraft system identification', AGARD Lecture Series No. 178, "Rotorcraft System Identification", AGARD, Neuilly sur Seine, France, 1991.

INTERNAL VERIFICATION

Assessment of consistency and accuracy of a simulation model compared with the underlying mathematical model in terms of :-

a) logical, mathematical and conceptual features.

b) algorithmic correctness (e.g. appropriate numerical methods for intended application).

EXTERNAL VALIDATION

Assessment of the mathematical model and its suitability for the intended application.

Must distinguish between:-

a) **Functional Validation**

and b) **Physical or Theoretical Validation**

EXTERNAL VALIDATION AS PART OF MODEL DEVELOPMENT PROCESS:

components → sub-models → complete model

Both functional and physical/theoretical validation processes involve elements of empirical validation

i.e. assessment of level of agreement between model variables and corresponding data from the real system.

EMPIRICAL VALIDATION

Complexity of instrumentation

$$\propto \frac{1}{\text{amount of a priori knowledge}}$$

METHODS BASED ON COMPARISONS OF RESPONSE DATA

a) **Overlaying of plots for model variables and corresponding system variables.**

METHODS BASED ON COMPARISONS OF RESPONSE DATA

b) Plots of differences between model and system responses.

THEIL'S INEQUALITY COEFFICIENT (TIC)

$$TIC = \frac{\sqrt{\sum_{i=1}^{n} (y_i - z_i)^2}}{\sqrt{\sum_{i=1}^{n} y_i^2} + \sqrt{\sum_{i=1}^{n} z_i^2}}$$

$$0 \leq TIC \leq 1$$

TIC ≤ 0 : time series almost identical

TIC ≤ 1 : time series differ significantly

FITNESS FUNCTION APPROACH

e.g. $$f(y) = \frac{1}{1 + e(y)}$$

where $$e(y) = \frac{1}{n} \sum_{i=1}^{n} (y_i - z_i)^2$$

$$0 \leq f(y) \leq 1$$

$f(y) \leq 1$: time series almost identical

$f(y) \leq 0$: time series differ significantly

METHODS BASED ON COMPARISONS OF RESPONSE DATA

c) Involving numerical measures of goodness-of-fit

$$\text{e.g. } J = \sum_{i=1}^{n} (y_i - z_i)^T w_i (y_i - z_i)$$

y_i = measured response at sample i

z_i = model response at sample i

w_i = weighting function

METHODS BASED ON COMPARISONS OF RESPONSE DATA

d) STATISTICAL METHODS

e.g. Histograms
Frequency distributions } **of complete data set**

Fitting of stochastic models

METHODS BASED ON COMPARISONS OF RESPONSE DATA

MODEL DISTORTION APPROACH

(Butterfield and Thomas)

Assessment in terms of PARAMETER DISTORTION needed to ensure that model achieves exact fit.

Compare parameter changes with uncertainty levels associated with the nominal parameter set.

METHODS BASED ON SYSTEM IDENTIFICATION TECHNIQUES

a) **Identifiability analysis**

b) **Estimation of paremeters in identifiable models**

c) **Cross-validation**

METHODS BASED ON SENSITIVITY ANALYSIS (Tomović et al.)

a) **Can help establish dependence of each part of response on each of the parameters of the model.**

b) **Can use to investigate whether parameter changes alone sufficient to provide required match of model and system.**

c) **Can suggest possible presence of fundamental structural errors in the model.**

METHODS BASED ON INVERSE MODELS

Steps:

a) Create inverse model

b) Drive inverse model using measured response data from the real system.

c) Compare input time series predicted by inverse model with input time series recorded from the real system.

Advantages

Avoids effects of small offset and bias components at system input.

Differences may be more apparent at the system input.

MODEL DOCUMENTATION

Should include:-

a) **Full details of purpose of model and the intended application.**

b) **Assumptions within the model.**

c) **Simplifications and approximations introduced.**

d) **Details of tests applied for internal verification.**

e) **Details of external validation processes.**

EXTERNAL VALIDATION viewed as a form of MODEL CALIBRATION PROCESS:-

Establishing useful range of model expressed in terms of

a) amplitude

b) frequency

i.e. range over which variables of model match corresponding quantities of real system to a stated level of accuracy.

Case Study I - Plant Modelling for a Two-Tank Liquid Level Control System

1. Introduction

For a vessel holding a mass of fluid M, the rate of change of mass in the container must equal the total mass inflow rate (Q_i) minus the total mass outflow rate (Q_o). That is

$$\frac{dM}{dt} = Q_i - Q_o \qquad (1)$$

The mass of fluid, M, is related to the volume of fluid in the vessel, V, by the equation

$$M = \rho V \qquad (2)$$

where ρ is the fluid density. For an incompressible fluid ρ is constant and thus

$$\dot{M} = \rho \dot{V} \qquad (3)$$

Figure 1 shows a vessel of rectangular cross section, with surface area A, it is possible to relate the mass of liquid, M, to the liquid height, H, through an equation

$$M = \rho AH \qquad (4)$$

The hydrostatic pressure at the base of the vessel is then

$$P = \rho gH \qquad (5)$$

where g is the gravitational constant. For the system of Figure 1, if the pressure at the surface of the liquid and at the outlet are the same and equal to P_a (say, atmospheric pressure) the pressure difference between the tank base and the outlet is given by $(P_a+P)-P_a$ which is simply P. The output flow rate Q_o is dependent on P and, for the case of laminar flow, is conventionally described by an equation of the form

$$Q_o = \frac{P}{R} \qquad (6)$$

where R is the fluid resistance.

Assume now that $Q_i(t)$ is known and that we want to predict the system behaviour in terms of the liquid height H(t). From Equations (1), (4), (5) and (6) we may write

$$\rho A \frac{dh}{dt} = Q_i - \frac{\rho g H}{R} \tag{7}$$

so that

$$A \frac{dH}{dt} = \frac{Q_i}{\rho} - \frac{gH}{R} \tag{8}$$

Note that Q_i is the mass flow rate and thus Q_i/ρ is the volume flow rate, (Q_{vi}), so that

$$A \frac{dH}{dt} = Q_{vi} - \frac{gH}{R} \tag{9}$$

Note that the relationship describing the flow at the outlet of the vessel is not always that shown in Equation (6) and the form of expression which is appropriate depends upon the nature of the outlet. For example, if the outlet is simply a hole in the side wall of the tank a condition known as **orifice flow** occurs. In this case, if the size of the orifice is small, and the pressure variation over the orifice area is thus negligible compared with the average orifice pressure, it can be shown (from the principle of conservation of energy) that the mass flow rate through the orifice is given by

$$Q_o = C_d a_o \sqrt{\frac{2P}{\rho}} = C_d a_0 \sqrt{2gH} \tag{10}$$

where a_o is the orifice area and C_d is the discharge coefficient. If, on the other hand, the outlet is through a pipe with turbulent flow, the appropriate relationship is

$$Q_o = \sqrt{\frac{P}{R_T}} \tag{11}$$

where R_T is a constant. Practical hydraulic components, such as valves, can be described by Equation (6) for small pressure drops but have to be described by Equation (11) in many cases due to turbulence at typical operating conditions.

2. Modelling of a pair of interconnected tanks

When a hydraulic system incorporates more than one liquid storage vessel the principle of conservation of mass (Equation (1)) may be applied to each element in turn. However, there is coupling between the vessels and the nature of this coupling depends upon the precise configuration of the vessels and upon the operating conditions. The interconnected tanks being modelled in this chapter are bench-top systems intended for use in teaching the principles of automatic control engineering [1].

Figure 2 is a schematic diagram of the system being considered. It consists of a

container of volume 6 litres having a centre partition which divides the container into two separate tanks. Coupling between the tanks is provided by a number of holes of various diameters near the base of the partition and the extent of the coupling may be adjusted through the insertion of plugs into one or more of these holes. The system is equipped with a drain tap, under manual control, and the flow rate from one of the tanks can be adjusted through this. The other tank has an inflow provided by a variable speed pump which is electrically driven. Both tanks are equipped with sensors which measure the pressure at the base of each tank and thus provide an electrical output voltage proportional to the liquid level.

2.1 A nonlinear mathematical model

Following the approach used in Section 1 the equation describing tank 1 in Figure 2 has the form

$$A_1 \frac{dH_1}{dt} = Q_{vi} - Q_{v1} \tag{12}$$

where H_1 is the height of liquid in tank 1, Q_{vi} is the input volume flow rate and Q_{v1} is the volume flow rate from tank 1 to tank 2 and A_1 is the cross-sectional area. Similarly for tank 2 we can write

$$A_2 \frac{dH_2}{dt} = Q_{v1} - Q_{vo} \tag{13}$$

where H_2 is the height of liquid in tank 2 and Q_{vo} is the flow rate of liquid out of tank 2. Considering the holes connecting the two tanks and the drain tap all as simple orifices allows the flow rates to be related to the liquid heights by the following two equations

$$Q_{v1} = C_{d_1} a_1 \sqrt{2g(H_1 - H_2)} \tag{14}$$

and

$$Q_{vo} = C_{d_2} a_2 \sqrt{2g(H_2 - H_3)} \tag{15}$$

where a_1 is the cross sectional area of the orifice between the two tanks, a_2 is the cross-sectional area of the orifice representing the drain tap, H_3 is the height of the drain tap above the base of the tank and g is the gravitational constant.

2.2 Linearisation of the model

For control system design studies it is appropriate to consider a linearised model in which the model variables represent small variations about steady state values. Thus the input flow variable is q_{vi} representing small variations about a steady flow rate Q_{vi}. Similarly the other variables represent small variations about steady values q_{v1} in Q_{v1}, q_{vo} in Q_{vo}, h_1 in H_1 and h_2 in H_2. In the steady state

$$Q_{vi} = Q_{v1} = Q_{v0} \tag{16}$$

$$q_{vi} - q_{v1} = A_1 \frac{dh_1}{dt} \tag{17}$$

$$q_{v1} - q_{vo} = A_2 \frac{dh_2}{dt} \tag{18}$$

From Equation (14) it is clear that Q_{v1} is a function of both H_1 and H_2. Hence the small variation in flow, q_{v1}, must depend on the steady levels H_1 and H_2 about which the system is operating. In general, one may therefore write

$$q_{v1} = \frac{\partial Q_{v1}}{\partial H_1} h_1 + \frac{\partial Q_{v1}}{\partial H_2} h_2 \tag{19}$$

Differentiating Equation (14) partially with respect to H_1 and H_2 in turn gives

$$q_{v1} = \frac{C_{d_1} a_1 \sqrt{2g}}{2\sqrt{H_1 - H_2}} (h_1 - h_2) \tag{20}$$

Similarly

$$q_{vo} = \frac{\partial Q_{vo}}{\partial H_2} h_2 = \frac{C_{d_2} a_2 \sqrt{2g}}{2\sqrt{H_2 - H_3}} h_2 \tag{21}$$

Substituting for q_{v1} and q_{vo} in Equations (17) and (18) gives

$$A_1 \frac{dh_1}{dt} = q_{vi} - \alpha_1 (h_1 - h_2) \tag{22}$$

$$A_2 \frac{dh_2}{dt} = \alpha_1 (h_1 - h_2) - \alpha_2 h_2 \tag{23}$$

where

$$\alpha_1 = \frac{C_{d_1} a_1 \sqrt{2g}}{2\sqrt{H_1 - H_2}} \tag{24}$$

and

$$\alpha_2 = \frac{C_{d_2} a_2 \sqrt{2g}}{2\sqrt{H_2 - H_3}} \tag{25}$$

Reorganisation of Equations (22) and (23) gives a second order state-space model as follows

$$\begin{bmatrix}\dot{h}_1\\ \dot{h}_2\end{bmatrix} = \begin{bmatrix}-\frac{\alpha_1}{A_1} & \frac{\alpha_1}{A_1}\\ \frac{\alpha_1}{A_2} & -\frac{(\alpha_1+\alpha_2)}{A_2}\end{bmatrix}\begin{bmatrix}h_1\\ h_2\end{bmatrix} + \begin{bmatrix}\frac{1}{A_1}\\ 0\end{bmatrix} q_{vi} \tag{26}$$

Taking Laplace transforms it is possible to obtain the transfer function descriptions relating the depth h_1 and the depth h_2 to the input flow rate q_{vi}. These are as follows:

$$h_2(s) = \frac{\frac{1}{\alpha_2}}{1 + \frac{(A_1\alpha_1 + A_2\alpha_1 + A_2\alpha_2)}{\alpha_1\alpha_2}s + \frac{A_1A_2}{\alpha_1\alpha_2}s^2}\, q_{vi}(s) \tag{27}$$

and

$$h_1(s) = \frac{\frac{(\alpha_1+\alpha_2)}{\alpha_1\alpha_2}\left(1 + s\frac{A_2}{\alpha_1+\alpha_2}\right)}{1 + \frac{(A_1\alpha 1 + A_2\alpha_1 + A_2\alpha_2)}{\alpha_1\alpha_2}s + \frac{A_1A_2}{\alpha_1\alpha_2}s^2}\, q_{vi}(s) \tag{28}$$

These both involve a pair of simple real poles and the characteristic equation in both cases is

$$(1 + sT_1)(1 + sT_2) = 1 + \frac{(A_1\alpha_1 + A_2\alpha_1 + A_2\alpha_2)}{\alpha_1\alpha_2}s + \frac{A_1A_2}{\alpha_1\alpha_2}s^2 = 0 \tag{29}$$

or

$$(1 + sT_1)(1 + sT_2) = 0 \tag{30}$$

where

$$T_1T_2 = \frac{A_1A_2}{\alpha_1\alpha_2} \tag{31}$$

and

$$T_1 + T_2 = \frac{A_1\alpha_1 + A_2\alpha_1 + A_2\alpha_2}{\alpha_1\alpha_2} \tag{32}$$

3. Programs for simulation of the nonlinear coupled tank system

A complete source program for simulation of the two-tanks system, using the SLIM simulation language and based on Equations (12) to (15), is included as a .SLI file (TANKS.SLI) in the Appendix to these notes. Nominal parameter values for this nonlinear model, corresponding to a real laboratory-scale coupled tank system, are as shown in Table 1. A data file (TANKS.IN) is provided for this nominal set of parameters. Figure 3 shows an XANALOG block diagram for this simulation model.

Some results based on the simulation program TANKS.SLI and the given parameter set are shown in Figure 4. In all cases the inflow was zero and the drain tap on the second tank was fully open.

3.1 Internal verification of the simulation program

The first stage of internal verification is concerned with checking that the structure of the simulation program is consistent with the mathematical model. This involves working backwards from the statements in the program, especially those within the derivative section, to ensure that when translated back to the form of differential equations they are the same as those of the original model. Checks should also be made of the parameter values used in the program or in the parameter input file to ensure that they correspond exactly to the parameter set of the model itself.

The second stage of internal verification is concerned with numerical accuracy. In the case of fixed step integration methods comparisons can be made of results obtained with a number of different sizes of integration step length and with different integration techniques. This provides the user with some understanding of the sensitivity of results to the step length and of the overall suitability of the numerical methods chosen. In the case of variable step integration algorithms tests can be carried out to compare results with different settings of the relative and absolute error limits and with different values of the minimum integration step to be allowed. In both cases some comparisons can be made using a number of different values of the communication interval to ensure that interesting events in the simulation model are not being hidden from the user simply because of an inappropriate choice of this parameter which determines the interval between output samples.

3.2 External validation of the simulation model

Statements about model validity must always be made in the context of an intended application. In the case of the coupled tank system the computer simulation is to be used as a basis for the design of an automatic control system for regulation of level in one of the tanks. There is particular interest therefore in the accuracy of the model in predicting steady state conditions and in predicting the form of small transients about any given steady operating point. Such comparisons are very easily made in the case of small-scale laboratory equipment of this kind and agreement between steady-state measurements and steady-state model predictions is generally quite good for most parts of the operating range. Table 2 shows some typical results obtained from measurements on the real system and corresponding tests on the simulation model. Differences between the steady-state liquid levels in the simulation model and in the real system, for a given value of input flow rate, are significant and vary slightly with operating point. Figures 5 shows discrepancies between the model and

system for a test involving a large change in operating conditions. Response data for small perturbation step tests carried out about one chosen operating point give time constants which are broadly in agreement with values determined from the linearised model in the form of Equations (27) and (28).

The discrepancies in the model exposed by the steady state tests and the large perturbation responses are believed to be due mainly to the limitations of Equations (14) and (15) in describing the relationships between output flow and the liquid level in each tank. These equations apply to an ideal simple orifice and the actual physical effects at the tank outlets do not agree exactly with this simplified model.

With closed-loop control added to the real system and to the simulation model the agreement can be shown to be significantly closer. This is important since the equipment is intended to be used for investigations of closed-loop control. In simulation studies involving control system design applications there is always particular interest in the overall robustness of the control system and the effect which modelling errors and uncertainties may have on the performance of the closed-loop system. Although control systems are normally designed using linearised models, simulation studies carried out on a proposed closed-loop system using a nonlinear model of the plant can often be highly illuminating. Such an investigation may reveal problems with the proposed design which would otherwise only come to light during the commissioning and testing of the real system .

4 Discussion

This case study provides an illustration of a relatively simple nonlinear system which can be modelled in a classical way using physical laws and principles. The simulation model is easily implemented using either equation-oriented or block-oriented tools. The relatively simple nature of the system and the variables which are accessible for measurement in the real hardware make this an interesting but straightforward system for the application of external validation methods.

Possibilities for using the simulation program TANKS.SLI as a basis for further investigations may be found elsewhere [2].

References

[1] Wellstead, P.E., "Coupled Tanks Apparatus: Manual", TecQuipment International Ltd, Long Eaton, Nottingham, NG10 2AN, U.K., 1981.

[2] Murray-Smith, D.J., "Continuous System Simulation", Chapman & Hall, London, 1995.

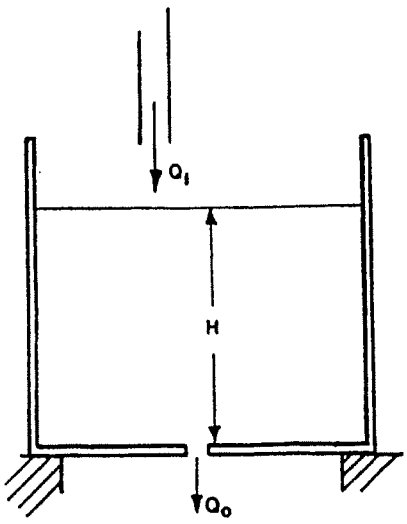


Figure 1: Simple tank of rectangular cross-section.

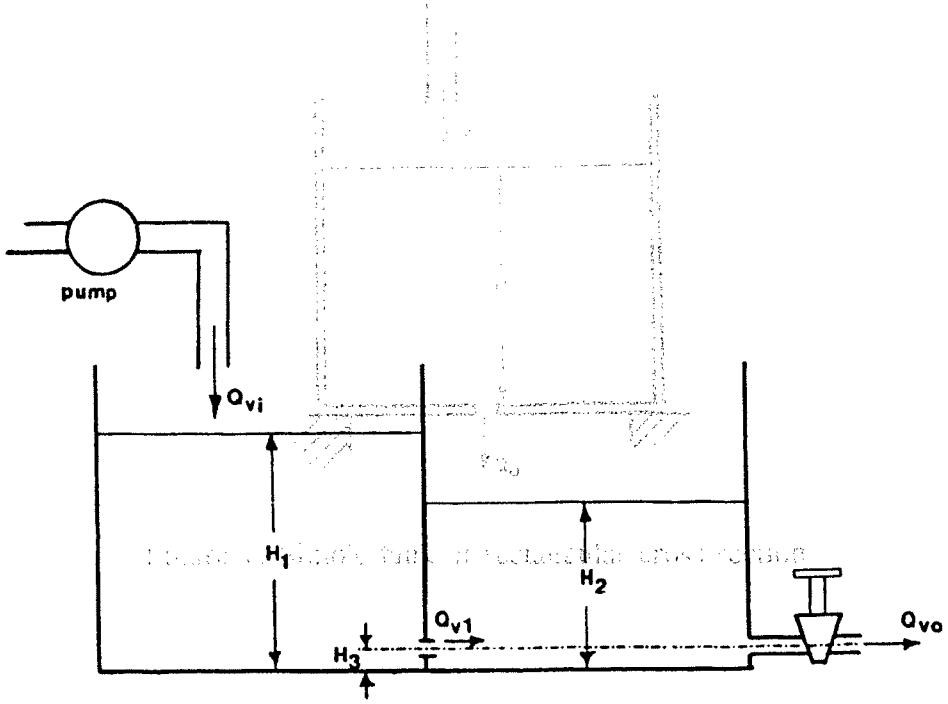
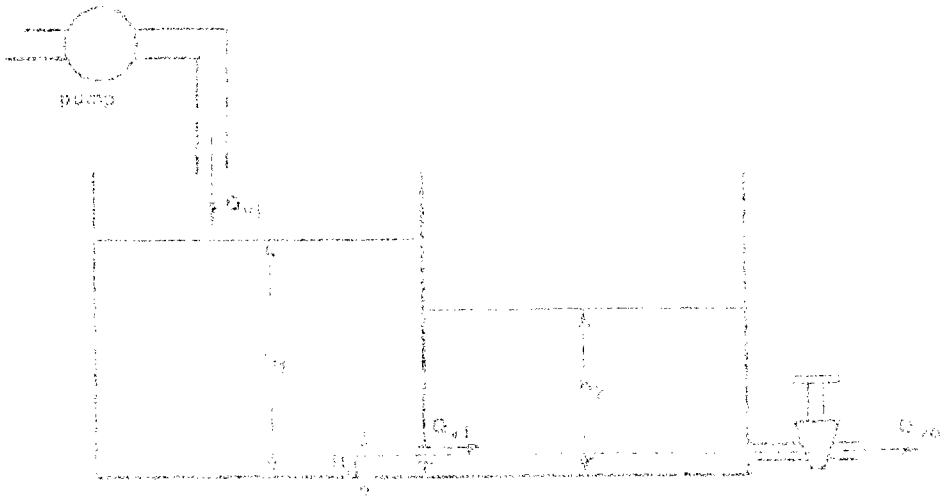


Figure 2: Pair of inter-connected tanks.



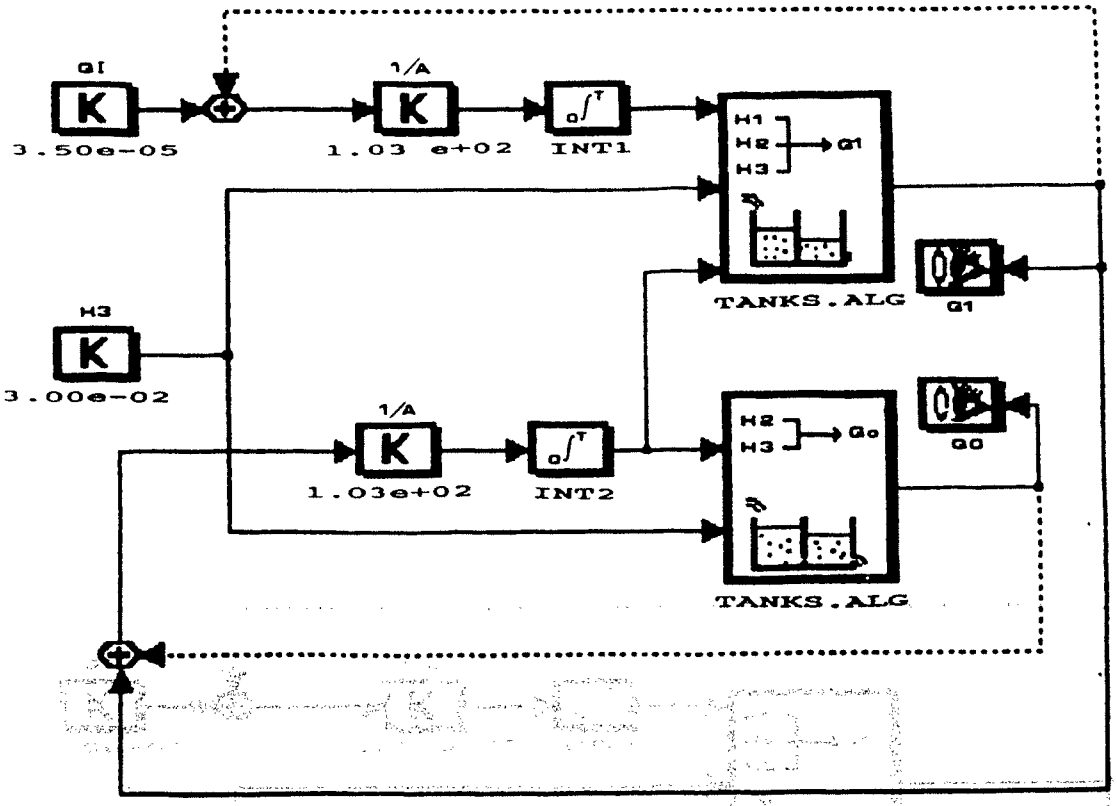


Figure 3: XANALOG block diagram for two-tank system simulation.

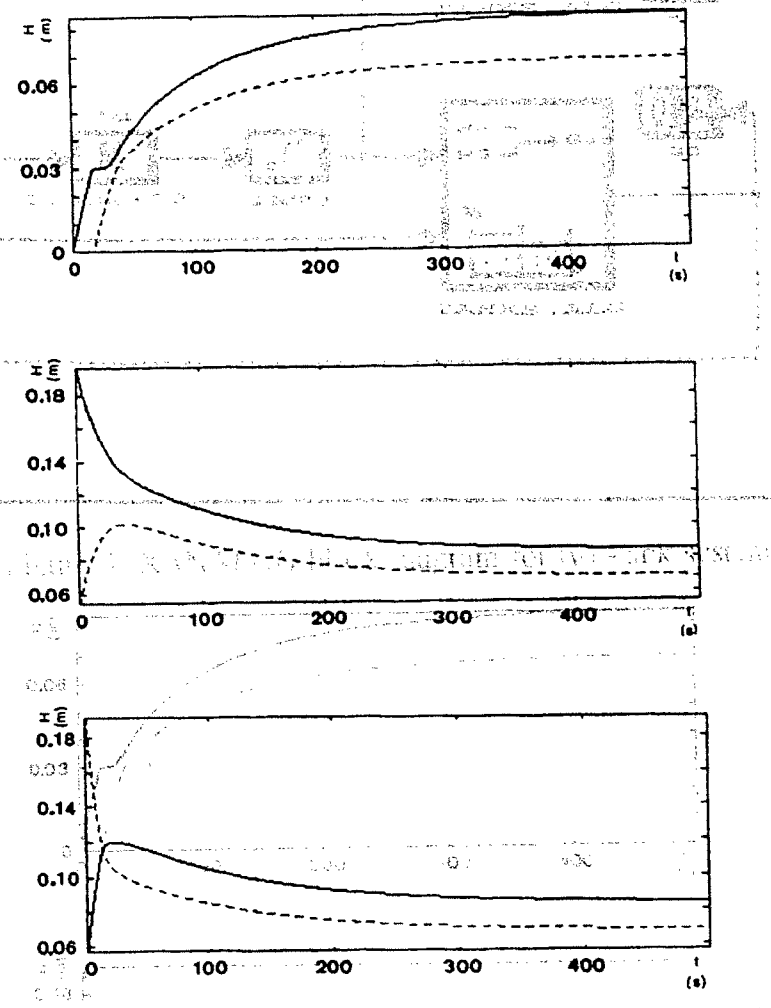


Figure 4: Simulation results for three sets of initial conditions.

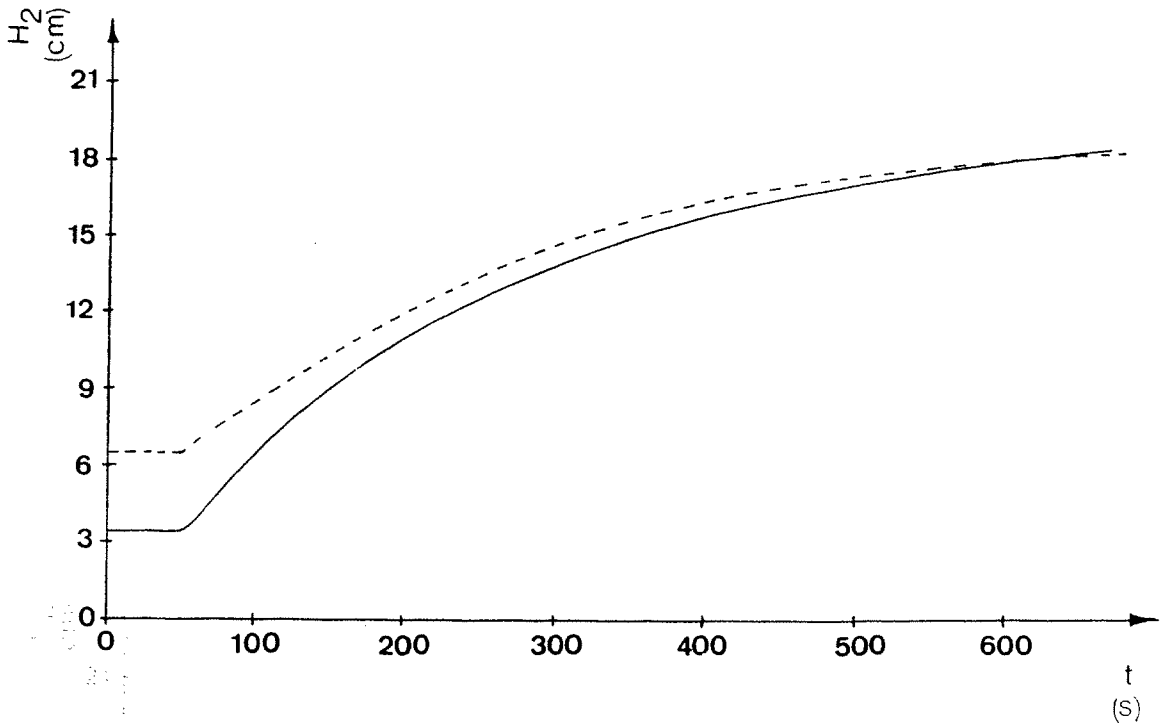


Figure 5: Simulation results for level in second tank, with measured response from the real system.

PARAMETER	SYMBOL	VALUE
Cross-sectional area, tank no. 1	A_1	0.0097 m^2
Cross-sectional area, tank no. 2	A_2	0.0097 m^2
Orifice area, between tanks	a_1	0.00003956 m^2
Orifice area, outlet from tank no. 2	a_2	0.0000385 m^2
Coefficient of discharge, inter-tank orifice	C_{d1}	0.75
Coefficient of discharge, outlet orifice from tank no. 2	C_{d2}	0.5
Gravitational constant	g	9.81 ms^{-2}
Pump calibration constant	G_p	0.0000072 $m^3s^{-1}V^{-1}$
Depth sensor calibration constant	G_d	33.33 Vm^{-1}
Height of outlet above base of tank	H_3	0.03 m

Table 1: Parameter values for the coupled-tank model

Q_1 (cm^3min^{-1})	H_1 measured (cm)	H_1 model (cm)	H_2 measured (cm)	H_2 model (cm)
1000	5.0	8.4	3.4	6.8
1500	13.1	15.2	9.7	11.6
2000	25.0	24.7	18.9	18.3

Table 2: System and model variables under steady-state conditions for three operating points

APPENDIX

Listing of SLIM program for Case Study

```
C
C  Simulation of the coupled tanks apparatus under
C  open-loop conditions. A typical input data set
C  is available in the file TANKS.IN.
C
      NCH=5
      NSA=500
      SAM=1.0
      TMAX=500.0
C
C  Information about output file organisation
C
      TYPE NCH
      TYPE NSA
      TYPE SAM
C
C  Input parameter values for model
C
      ACCEPT QI,H10,H20
      CD1=0.75
      CD2=0.5
      A1=3.956E-5
      A2=3.85E-5
      G=9.81
      H3=0.03
      A=9.7E-3
C
C  Calculation of equilibrium levels
C
      H2SS=H3+(1.0/(2.0*G))*((QI/(CD2*A2))*(QI/(CD2*A2)))
      H1SS=H2SS+(1.0/(2.0*G))*((QI/(CD1*A1))*(QI/(CD1*A1)))
      H1=H10
      H2=H20
      T=0.0
      CINTERVAL(SAM)
C
C  Values for integration parameters
C
      MINTERVAL(1.0E-6)
      MERROR(1.0E-4)
      XERROR(1.0E-4)
C
C  End of initial segment, start of DYNAMIC segment
C
      DYNAMIC
C         DERIVATIVE
C  Calculation
4               IF(H1-H3)4,2,2
3               IF(H2-H3)3,5,5
                Q1=0.0
                GOTO 7
5               H1=H10  Q1=-CD1*A1*SQRT(2.0*G*(H2-H1))
                H2=H20  GOTO 10
2               T=0.0   IF(H1-H2)5,6,6
6               CINTERVAL(SAM)  Q1=CD1*A1*SQRT(2.0*G*(H1-H2))
C               GOTO 10
10  Values for  IF(H2-H3)7,7,8
7               QO=0.0
                MINTERVAL(1.0E-6)
8               MERROR(1.0E-4)  QO=CD2*A2*SQRT(2.0*G*(H2-H3))
                XERROR(1.0E-4)  GOTO 20
C
C  End of initial segment, start of DYNAMIC segment
C
      DYNAMIC
          DERIVATIVE
              IF(H1-H3)4,2,2
4             H2=H20  Q1=0.0
              T=0.0   GOTO 7
```

```
20              CONTINUE
                DH1=(1.0/A)*(QI-Q1)
                DH2=(1.0/A)*(Q1-QO)
                H1=INTEG(DH1,H10)
                H2=INTEG(DH2,H20)
            DERIVATIVE END
            TYPE T, H1SS,H2SS, H1,H2
            IF(T-TMAX)50,50,60
50      DYNAMIC END
60      STOP
        END
```

Data file TANKS.IN

```
33.32E-6
0.05
0.2
```

SLIM - A Simple Continuous System Simulation Language

USER'S GUIDE

D.J. Murray-Smith
Department of Electronics and Electrical Engineering
University of Glasgow
Glasgow G12 8QQ
Scotland
United Kingdom

Version 1.0

July 1994

(as distributed on diskette with the book "Continuous System Simulation", by D.J. Murray-Smith, published by Chapman and Hall, London, 1994)

1. INTRODUCTION

The name SLIM stands for "A Simulation Language for Introductory Modelling". It involves a subset of the CSSL Standard established by the Sci Technical Committee in 1967. SLIM has been developed specifically as a language through which those new to continuous system simulation can be introduced to the principles of the subject.

The SLIM software included on the diskette which accompanies the book "Continuous System Simulation" by D.J. Murray-Smith is the most basic version of the SLIM family. Only one integration method is available (a variable-step method) and features such as function tables, pure delays, the use of arrays, the use of multiple DERIVATIVE sections, sub-model capabilities, sorting etc. are not provided. Such features may be found in many commercially-supported simulation languages. SLIM provides only the most basic simulation facilities but it is a very easily-used simulation tool and, with a little ingenuity, can be applied to a remarkably wide range of problems, as shown by the examples of Chapter 8 in the book and by the case studies in later Chapters.

Detailed practical information concerning the installation of the SLIM software and on the preparation and running of SLIM programs may be found in Section 4 of this Guide. This information is specific to the version of SLIM included on the diskette accompanying the book. Other versions of SLIM have been developed which incorporate additional facilities and are suitable for use with other operating systems, such as VMS and UNIX. A version has also been produced for use on a Parsytec parallel processor system under the PARIX system.

The SLIM software consists of a language processor which has been written in FORTRAN 77 which is capable of translating and interpreting programs written using the special SLIM instruction set. This instruction set involves a small subset of FORTRAN together with a number of the special CSSL instructions needed for simulation applications. SLIM, like many other examples of application-specific software systems, is based upon the use of a syntax-driven parser with a separate executor routine. The language processor first parses the user's program and detects any statements which are invalid. Once the validity of the complete program has been established, the software produces intermediate code which is not seen by the user but which can be interpreted by the executor routine.

2. THE SYNTAX OF THE SLIM LANGUAGE

SLIM follows the principles of the 1967 CSSL standard and the syntax has much in common with ACSL and other widely used languages. There are, however, no explicit INITIAL and TERMINAL statements and the structure of the program is defined only through DYNAMIC, DERIVATIVE, DERIVATIVE END and DYNAMIC END statements.

It should be noted that SLIM follows some of the conventions of FORTRAN and, in accordance with the standard for that language, all program statements in SLIM must begin at, or to the right of, the seventh column of a program line. The first five columns are used only for comments or labels. Comments are indicated by the character 'C' in the first column of the line and are

nored completely in the translation process. Labels must be numeric symbols
d may be entered in any or all of the first five columns. The sixth column
st always contain a space unless the line in question is a comment line.
per case characters must be used throughout, except for comments which may
volve upper or lower case characters, as desired. Blank lines should not
included in a SLIM program and tab facilities should not be used to insert
aces.

additional link with FORTRAN is the fact that the two types of variable
fined within SLIM, integer variables and real variables, are distinguished
cording to the letter of the alphabet with which the variable name starts.
l integer variable names must begin with the letters I,J,K,L,M and N
ereas real variable names must start with any other letter of the alphabet.
al variables and constants may have magnitudes in the range 1.18E-38 to
40E+38 while integer variables and constants must have magnitudes in the
nge 0 to 32768.

riable names may involve any combination of letters and numerals (after the
rst character which must of course be a letter) but can only have six
aracters in total.

amples of valid real variables are:

```
VALUE
VALUE1
GAIN
ACC2A
```

amples of valid integer variables are:

```
INVAL5
MPOINT
NFLAG
KP
```

al constants are exemplified by:

```
1.0
3.142
4.8E-7
6E4
```

id integer constants by:

```
1
567
22141
```

atements in SLIM may be divided conveniently into general-purpose
atements which provide the normal facilities for numerical programming
id which have counterparts in most high-level programming languages, and
ose statements which are of special significance for simulation.

t should be noted that in the SLIM language expressions are evaluated
cording to the normal operator precedence rules. The order of evaluation
s therefore as follows:

```
** raising to a power
*,/ multiplication or division
+,- addition or subtraction
```

It should be noted that SLIM statements must not extend beyond one line of code since there is no facility in the language for any form of continuation line. Complex expressions may therefore have to be split initially into a number of smaller expressions and combined at a later stage in the program.

i) GENERAL-PURPOSE STATEMENTS

a) INTEGER AND REAL ASSIGNMENTS

This type of statement takes the general form:

Real variable = Real expression
or Integer variable = Integer expression

Examples of this type of assignment are:

```
VELOCY=(7.5-DISTCE)/4.982
IVALUE=8+7*K
```

Note that mixed expressions involving combinations of integers with real variables or constants are not permitted and if detected will be rejected as a syntax error.The functions FLOAT and IFIX may be used for data type conversion to overcome this limitation.

Real intrinsic functions available within SLIM which may be included within Real Assignment statements are as follows:

ABS,ALOG,ATAN,COS,EXP,FLOAT,SIN,SQRT and RND.

All these functions except RND are quite standard and have the same role as the correspondingly named functions in FORTRAN. All except FLOAT involve real arguments and give real results. FLOAT requires an integer argument and produces a real result.

The function RND generates uniformly distributed random numbers having values which lie between 0.0 and 1.0. The seed value used for random number generation is determined by the argument used in calling the function. Ten different seed values are available. All values of the argument which are less than 2.0 lead to the use of one particular seed, values equal to or greater than 2.0 but less than 3.0 lead to use of a second predetermined seed value and so on for all other similar intervals in the range 3.0 to 10.0.. All argument values of 10.0 or greater lead to the use of the tenth seed value.

The only integer intrinsic function in SLIM is IFIX. It can be used in the same way as the corresponding function in FORTRAN. It requires a real argument and returns an integer result.

b)INPUT STATEMENT

Input of data in SLIM is from a file named by the user. The form of the Inpu statement is as follows:

ACCEPT variable list

The variable list contains variable names which can be of either real or integer type. An example of this type of SLIM assignment is:

ACCEPT VALUE,CONST,GAIN

OUTPUT STATEMENT

ıtput is always written to a named file on disk but may also appear in ıbular form on screen if desired. The form of the Output statement is:

TYPE variable list

.g.
```
TYPE VALUE,CONST,GAIN
TYPE J,IVAL
```

ıe variable list within an Output statement must only involve variables of ıe type. A mixture of integer and real variables in a single variable list s not permitted in an Output statement. The first few items in the output ile provide information about the conditions selected within the simulation un and are written to the file automatically. These include three integers roviding system programming information which are not of direct interest to he user followed by the values of the communication interval, the absolute nd relative error parameters used in the program, and the minimum interval.

he next items have to be included in the output file but require the nclusion of TYPE statements in the SLIM source program for the simulation. hey involve, in order, the number of channels in the output (say NCH), the umber of samples used in the simulation run (say NSA) and the communication nterval (say SAM). Every SLIM program must therefore include, at the end of he initial segment, three lines such as:

```
TYPE NCH
TYPE NSA
TYPE SAM
```

vhere NCH is the integer constant representing the number of output variables including the independent variable where appropriate), NSA is the integer constant representing the number of samples over the complete time history of each variable and SAM is the real constant representing the communication interval. It is, of course, important that these three quantities be written to the output file in this particular order and that they should be the first items output from the simulation program. It should be noted that in all cases the quantities NSA and SAM must be chosen to be consistent with the interval of the independent variable (usually time) over which the results are required. If, for example the independent variable has an initial value T=0.0 and a final value T=TMAX in the simulation program the quantities NSA and SAM must be such that their product is equal to TMAX.

Immediately after this it is normal to output the initial values of the independent variable and of the system variables of interest. Four TYPE statements therefore normally appear within the initial segment of any SLIM simulation program for which an output file is required.

d) ARITHMETIC IF STATEMENT

The syntax of an Arithmetic IF statement is as follows:

IF(arithmetic expression) Label 1, Label 2, Label 3

When this type of statement is executed the arithmetic expression is evaluated and control passes to Label 1 if the result is negative, to Label 2 if the result is zero and to Label 3 if the result is positive. Examples of such a statement are as follows:

```
IF(VELOCY-THRESH)175,90,90
IF(IVAL-K)10,20,30
```

In the first case the expression (VELOCY-THRESH) is evaluated and control is passed to the statements in lines labelled 175 or 90 depending on sign of the result. In the second case the integer result from the evaluation of the expression (IVAL-K) determines whether control passes to the statements with labels 10, 20 or 30. It should be noted that in the case of real expressions the condition corresponding to the result being zero is not of practical value due to problems of round-off and other inaccuracies in real arithmetic.

e) GOTO STATEMENT

This statement causes the normal sequential process of program statement execution to be interrupted. Control is passed unconditionally to the line which has the label contained in the GOTO statement. For example, the following section of program:

```
      X=1.0
40    TYPE X
      X=(X+10.0)/2.0
      GOTO 40
```

would produce an endless loop which, if executed, would produce an output sequence as follows:

```
5.50
7.75
8.875
9.4375
 etc.
```

f) CONTINUE STATEMENT

A CONTINUE statement is a dummy execution statement and simply causes control to be passed to the next statement in the program.

g) STOP STATEMENT

A STOP statement causes execution to cease. This is usually the last executable statement in a SLIM program. Like all other executable statements it can be given a label.

h) END STATEMENT

The purpose of an END statement is to indicate the end of the program. It must therefore always be the last statement and since it is not executable it cannot be given a label.

ii) SPECIAL-PURPOSE SIMULATION STATEMENTS

a) DYNAMIC STATEMENT

The DYNAMIC statement is used to indicate the start of the dynamic segment of the simulation program. In SLIM only one dynamic segment is allowed within a program. On entering the dynamic segment the processor assigns initial values to the integration variables which are evaluated within the derivative section. The dynamic segment contains the derivative section and the segment is therefore in two parts, one above the derivative section and the other below it.

)) DERIVATIVE STATEMENT

'he DERIVATIVE statement indicates the beginning of the derivative section /hich lies entirely within the dynamic segment. The differential equations ınd other equations which define the dynamic simulation model are normally :ontained within this section.

:) DERIVATIVE END STATEMENT

'he DERIVATIVE END statement defines the end of the derivative section. It .ndicates that all the integral assignments have been evaluated and the ıumerical integration routine is called. On returning from the integration :outine the problem variables are all updated. Once control has been passed ınto the derivative section it will not be passed beyond the DERIVATIVE END ;tatement until a new communication interval is about to start. If the ;tart of a new communication interval has not been reached control is passed ɔack to the start of the derivative section and the integration process is :epeated with the updated variables. This continues until, at the appropriate :ime determined by the communication interval, control is transferred from :he derivative section to the second part of the dynamic segment but without :e-assignment of initial values.

1) DYNAMIC END STATEMENT

'he DYNAMIC END statement signifies the end of the dynamic segment. In order :o enter the region of the SLIM program lying beyond the DYNAMIC END ;tatement control must be transferred explicitly by means of a GOTO or ırithmetic IF statement.

:)INTEGRAL ASSIGNMENT STATEMENT

Each first order differential equation gives rise to an Integral Assignment ;tatement. This type of statement has the form:

Real variable=INTEG(real expression, initial value)

/here initial value=real variable or real constant

ın example of a statement of this kind could have the following form:

X1=INTEG(V*W,X1INIT)

This indicates that the real expression V*W is to be evaluated and from this :he real variable X1 is to be calculated by integration with respect to the ındependent variable (normally time). The initial condition for this integration operation is provided by the real variable X1INIT which must have ɔeen defined at an earlier stage in the program (usually in the initial segment).

:) CINTERVAL STATEMENT

The CINTERVAL statement sets the communication interval which defines the increment of the independent variable at which control is passed from the derivative section to the dynamic segment. The format of this statement is as follows:

CINTERVAL(unsigned real constant or real variable)

For example the statement

CINTERVAL(0.2)

will cause control to be passed from the derivative section to the dynamic segment every 0.2 units of the independent variable (e.g. every 0.2 sec.). The default communication interval is 1.0 and this value is used in any program in which no CINTERVAL is defined explicitly.

It should be noted,incidentally, that the default initial value of the independent variable in a SLIM simulation is zero. Other initial values are possible but must be set by the user using an assignment statement such as:

T=1.6

g) MINTERVAL STATEMENT

This statement defines the smallest step size allowed for a variable step length method. If this lower limit is reached an error message is displayed on the screen and control of the program is returned to the user. The MINTERVAL statement has the general form:

MINTERVAL(unsigned real constant or real variable)

e.g. MINTERVAL(1.0E-5)

The default is MINTERVAL(1.0E-4).

h) XERROR STATEMENT

The XERROR statement is used to set the maximum absolute error that can be accepted in the results of the variable step length numerical integration. The value is defined on a basis of the error per integration step. The variable step length routine used within SLIM either satisfies the requirement defined in the XERROR statement or indicates to the user that the step length has reached the minimum. The form of the statement is:

XERROR(unsigned real constant or real variable)

A typical example is :

XERROR(1.0E-5)

The default value corresponds to XERROR(1.0E-4).

i) MERROR STATEMENT

This statement is similar to XERROR but sets the maximum relative error (per integration step). Again the requirement is satisfied automatically unless the user is otherwise informed through an error message. The format of the MERROR statement is similar to CINTERVAL, MINTERVAL and XERROR and a typical example is:

MERROR(1.0E-4)

which would set the value to the default.

3. OUTPUT FACILITIES AND GRAPHICS

Graphical output may be obtained from the SLIM program by means of a post-processing program which uses the results file from SLIM as input. This

rogram, called SLIMPLOT, requests the name of the results file, together ith information about the channels of the output to be plotted. Both time istory plots and x-y plots can be produced by SLIMPLOT.

he first few items in the output files produced by the SLIM program provide nformation about the simulation run and about the internal organisation of he output file, as specified above. This information, which must appear in he standard format described in Section 2 (c) above, provides the post- rocessing program with essential information which allows it to interpret he simulation data which follows.

. INSTRUCTIONS FOR INSTALLATION AND USE OF SLIM

he software included on this diskette includes the SLIM.EXE program and the ost-processing program SLIMPLOT.EXE for the display of results in graphical orm. Example programs discussed in Chapters 6, 8, 10, 11 and 12 of "Continuous System Simulation" by D.J. Murray-Smith may also be found on the iskette, together with the necessary input data files.

1.1 Hardware requirements

The SLIM and SLIMPLOT programs run on any IBM-compatible personal computer running the Microsoft DOS (MS-DOS) operating system. A computer with at least an 80286 or 80386 microprocessor is preferred, although not essential. It is also recommended that the computer has the 80287 or 80387 floating-point co-processor which will significantly reduce the time required to obtain results.

A hard disk is desirable and the computer must have CGA, EGA or VGA display capabilities. The computer should contain a minimum of 512k bytes of RAM memory. A standard ASCII text editor is necessary.

4.2 Software installation

This section of the User's Guide provides practical information about the steps to be followed in installing the SLIM software on your computer. The README.DOC file on the diskette also provides details of procedures for installation of the SLIM and SLIMPLOT software, including a list of the names of all the files which are provided and any updated features.

SLIM and SLIMPLOT will run successfully from a floppy disk drive but most users prefer to mount the software on their hard disk. To do this you should create a new directory (normally named SLIM) on the hard disk. All files on the distribution diskette should be transferred to this new directory. To create such a directory use the normal DOS command:

```
mkdir slim
```

If you wish you could then create separate path-accessible sub-directories for user files. However, for simplicity, it is assumed in the remainder of these notes that all SLIM files are in the SLIM directory, that the default hard disk drive is c: and that floppy disk drive is a:.

To transfer the files from the SLIM diskette to the hard disk place the diskette in the a: drive and copy all the distribution files into the c:\SLIM directory as follows:

```
copy a: *.* c:\slim
```

It is important to make back-up copies of all of the files supplied on the distribution diskette prior to attempting to make use of the software. The original diskette should be stored safely.

4.3 Starting and exiting SLIM

Make the SLIM directory the current directory by entering the DOS command cd\slim. When the DOS prompt (in this case C:\SLIM>) appears type:

```
slim
```

and the SLIM program should start to execute. It first requests the name of the SLIM program file to be run. From this point the sequence of operations follows the pattern of the example of Section 6.6.4 of the book. Any error messages relating to errors in translation or occurring at run time should be self explanatory. Syntax errors detected at the translation stage should provide a reference to the line in which the error was found but will not pinpoint the precise nature of the error. It is necessary for the user to examine the relevant section of code to determine why the translation has failed. Comment lines are ignored in the line numbering system for SLIM error messages.

It may be useful to use the VDPOLS.SLI program file from the example in Section 6.6.4 of the book to ensure that you are using SLIM correctly and that the results of Chapter 6 can be reproduced on your own computer.

At the end of a simulation run the user is provided with an opportunity to carry out a further run with the same program through a query of the form:

```
DO YOU WANT ANOTHER RUN? (Y/N)
```

Responding N returns the user to DOS and the DOS prompt appears.

4.4 Useful hints for using SLIM

In preparing simulation programs using the SLIM language features described in Chapter 6 and in this User's Guide it is important to take note of the following points:

a) A SLIM program must follow the conventions explained in Chapter 6 and Section 2 of this Guide. These conventions are similar in many respects to the conventions for FORTRAN 4 or FORTRAN 77 programming.
b) It should be noted that executable code must start in column 7 of each line of program. Columns 1 to 5 are reserved for labels or for use in comment lines. Comments must start with an upper case C in column 1.
c) In creating a SLIM program using a general-purpose text editor, use of the TAB key should be avoided. The space bar should be used to move the cursor to the required column (e.g. column 7).
d) No blank lines should be included in a SLIM program. The presence of a blank line in a SLIM source program causes problems at the translation stage and may cause the computer to 'hang'.
e) Line numbers quoted in connection with error messages relate to lines involving SLIM language statements only. Lines involving comments are not given line numbers.
f) Continuation lines are not permitted in SLIM programs and care must be taken to avoid lengthy statements which involve more than 70 characters

or spaces. Attempts to use longer statements in SLIM program may cause the computer to 'hang' at the translation stage. Long statements must be split into two or more statements, each occupying a separate line of code.

g) The distinction between integer and real quantities is important and involves the normal naming conventions of FORTRAN. Real and integer quantities cannot be used in direct combination within a single statement.

h) All executable statements in a SLIM program must involve only upper case letters. Comments may involve upper case and lower case letters but must include an upper case C in column 1.

i) When a syntax error is detected at the program translation stage a number of messages may be displayed which appear to relate to syntax errors in a number of different lines. If there is no apparent reason for an error in one particular line check the statements in the other lines for which syntax errors have been reported. Clearing the error reported in one line may eliminate the apparently meaningless errors reported in other lines.

.5 Starting and exiting SLIMPLOT

LIMPLOT is extremely easy to use. It is entered using the following command:

```
slimplot
```

he program then prompts the user for the file name of the SLIM output file hich is to be plotted. This is followed by a request to the user to select he type of output file (y versus time or an x-y plot), the number of ariables (in the case of an y-t plot) and the column numbers of the ariables.

he column numbers referred to in the SLIMPLOT prompts are the column numbers rising from use of the TYPE statement to generate the output data rom the SLIM program. For example, if the output is obtained from a tatement of the form:

```
TYPE T,X1,X2,X3
```

there will be four columns in the output file. Column 1 involves values of T (the independent variable) and columns 2 to 4 involve the corresponding values of the dependent variables X1, X2 and X3. If one wanted to plot X1 and X3 the columns to be entered in response to the prompt from the SLIMPLOT program would be 2 and 4. These column numbers may be entered as:

```
2,4
```

or

```
2 4
```

Text can be inserted by the user to provide a main (top) title and up to two titles which appear below the graph. Both upper and lower case characters may be used in these titles.

When graphical output has been displayed on the screen and the user wishes to move on to a further stage of processing pressing the ENTER key clears the graphical output from the screen and causes a new menu to appear. This has one option which allows the user to exit from the SLIMPLOT program and others which allow further plots to be produced.

4.6 Useful hints for using SLIMPLOT

The following points should be noted:

a) Problems will be encountered if the number of time steps in a SLIM output file is greater than the maximum allowed by SLIMPLOT. If the maximum is exceeded an error message is displayed during the running of the SLIMPLOT program. The SLIM program should then be re-run with an increased communication interval (CINTERVAL) or a smaller range for the independent variable. For problems where the independent variable is time this means either increasing the time step for the plot or reducing the length of the simulation run.

b) Hard copy output from SLIMPLOT is most easily obtained using the DOS GRAPHICS command and the use of the Print Screen key. An alternative, if an appropriate printer is available, is to make use of the printer or plotter facilities in the list of output devices which is displayed during the running of SLIMPLOT.

5. EXAMPLES

Chapters 6, 8, 10, 11 and 12 of "Continuous System Simulation" include examples of SLIM programs. SLIM program files and data files for these examples are provided on this diskette. The program files all involve file names with a .SLI extension, while the input data files all involve files with a .IN extension. Filenames are the same as those used in the relevant chapters. It should be possible in each case to reproduce results presented in tabular or graphical form in the book.

6. COMPARISONS OF SLIM WITH WIDELY-USED SIMULATION LANGUAGES

SLIM is intended to provide beginners with an introduction the principles of continuous system simulation and has been designed as an inexpensive but reasonably versatile teaching tool. It is not intended to compete directly with commercially-supported languages. SLIM is most suitable for relatively small simulation problems. It does conform to many of the conventions of the CSSL specification and provides a convenient, efficient and easily-used tool. Having gained some initial experience with SLIM the user should be able to move on without difficulty to make use of other more powerful languages, such as ACSL.

The maximum number of state variables which can be accommodated within a SLIM simulation program using the version of the SLIM software which accompanies the book "Continuous System Simulation" is 40. This is not seen as imposing any serious restriction on the average beginner. All of the examples and case studies considered involve less than ten state variables. The maximum number of statements allowed within a SLIM program is 250. Once again typical applications involve much smaller programs. Some other simulation languages, such as ACSL, do not impose limitations of this kind.

While the restrictions in terms of program size are not thought to represent limits which are likely to be approached by the beginner, it is important to understand that languages such as CSSL IV, ACSL and DESIRE provide many features which are not available within SLIM. Some of these restrict the type of system which can be modelled using SLIM and others limit the options available to the user at run time. The sub-set of facilities incorporated in SLIM has been chosen to allow the beginner to gain experience rapidly, without having to deal with the complexities associated with features in mor comprehensive languages. A careful examination of the user manual for each specific language allows the potential user to establish the precise differences in every case.

e absence of facilities to model a pure time delay (transport delay) is one
ıportant example of a feature, available in many simulation tools, which has
›t been included within the version of SLIM provided with "Continuous
'stem Simulation". The lack of any facility to represent a function in
ıbular form is another example of a potentially useful feature which has
›t been included. Sorting facilities and facilities for sub-models are also
›t provided within SLIM. Simulation problems which require the use of
ıcilities such as transport delays and function tables cannot therefore be
ıckled using SLIM, unless the problem can be reformulated in some way to
.low it to be solved within the limitations of the language. Some other
ınguages, such as ACSL, provide a much more extensive set of facilities
ıd therefore provide much more flexibility at the model definition stage.

ı comparison with ACSL and other similar languages SLIM has limited
ıpabilities in terms of run-time options. There is no real separation of
ıe model definition and the experiment in SLIM. Changes of parameters
equire editing of the simulation program itself or changes of an associated
ıta file. However, flexibility is available in terms of output display
ıcilities in SLIM because of the fact that the graphical output options
ıvolve a separate post-processing program.

LIM can be extremely effective when learning about the principles of
ımulation and when used for the solution of problems for which its
ıcilities are appropriate. It is important, however, to understand its
ımitations and to have an appreciation of the additional facilities of the
›ftware products which are available commercially.

. DISCLAIMER

ll parties must rely on their own skill and judgement when making use of
ıe SLIM and SLIMPLOT software. Neither the author nor the publishers
ssume any liability to anyone for any loss or damage caused by any error
: omission in the work, whether such error or omission is the result of
egligence or any other cause. Any and all such liability is disclaimed.

. ACKNOWLEDGEMENTS

ıe features included in this version of the SLIM language are, as explained
ı Section 1 of this User's Guide, a subset of the features considered
esirable for equation-oriented continuous system simulation languages, as
ecommended in 1967 by the relevant Simulation Councils inc. technical
›mmittee. In considering the features to incorporate in this simulation
anguage, which has been designed specifically for teaching applications,
number of existing simulation tools were reviewed. One existing language
hich has had a particularly strong influence on SLIM is the MIMESIS
anguage which was developed by Ian Ricketts at the University of Dundee and
as described in his Ph.D. dissertation in 1977 (Ricketts,I.W. "MIMESIS - A
›ntinuous System Simulation Language", Ph.D. Thesis, University of Dundee,
977). SLIM and MIMESIS are both based on the principles of syntax-driven
arsers as described by R.L. Gauthier and R.D. Ponto in their text entitled
Designing Systems Programs" (Gauthier, R.L. and Ponto, R.D. "Designing
ystem Programs", Prentice-Hall, New Jersey, 1970). Many features of the
nstruction set of SLIM are similar to those of MIMESIS, although the internal
tructure of the SLIM program and the detailed coding are different.
xperience gained in using the MIMESIS software has been of particular value
n the development of SLIM and the author is very grateful to Dr. Ricketts
or making MIMESIS available.

he author also wishes to record his thanks to present and former students
n the Department of Electronics and Electrical Engineering at the University

of Glasgow who have assisted in the development of the SLIM software. A number of student projects have been associated with the application of SLIM to practical engineering problems and these have provided valuable information about the range of problems which can be tackled conveniently using this software. Very helpful feedback has also been provided freely by students who have used SLIM in connection with courses in computer simulation methods, in control engineering and in avionic systems at undergraduate and post-graduate level.

The software has undergone much revision and modification during the course of its development. Work on SLIM began in 1991 during a period of sabbatical leave which was spent at the Technical University of Vienna. The author is very grateful to Professor Dr. Felix Breitenecker of the Department of Simulation Techniques at the Technical University of Vienna for his assistance and support in making that visit possible and to the University of Glasgow for providing the necessary leave of absence. Without the opportunity provided by this period of uninterrupted work it is unlikely that SLIM would have been developed.

The graphical routines included in the SLIMPLOT program are part of the NAG Graphics Library. This product contains a collection of over 100 carefully designed routines which provides FORTRAN and C programmers with a versatile means of producing a graphical representation of numerical and statistical results. It is used worldwide in commerce, financial modelling, industry academic research and many other areas, and is available for a wide spectrum of machines. Interfaces are also supplied to the most commonly used plotting packages. NAG also provide a wide range of other products such as mathematical and statistical routines in FORTRAN, C, Pascal and Ada; FORTRAN 90 compilers and tools; symbolic solvers; data visualization software; linear programming software and a gateway generator which automatically generates MATLAB gateway. For further information, a quotation or details of your local distributor please contact:

Sales Department
NAG Ltd.
Wilkinson House
Jordan Hill Road
Oxford OX2 9BX
United Kingdom.

SLIM - A Simple Continuous System Simulation Language

USER'S GUIDE

D.J. Murray-Smith
Department of Electronics and Electrical Engineering
University of Glasgow
Glasgow G12 8QQ
Scotland
United Kingdom

Version 1.0

July 1994

(as distributed on diskette with the book "Continuous System Simulation",
by D.J. Murray-Smith, published by Chapman and Hall, London, 1994)

1. INTRODUCTION

The name SLIM stands for "A Simulation Language for Introductory Modelling". It involves a subset of the CSSL Standard established by the Sci Technical Committee in 1967. SLIM has been developed specifically as a language through which those new to continuous system simulation can be introduced to the principles of the subject.

The SLIM software included on the diskette which accompanies the book "Continuous System Simulation" by D.J. Murray-Smith is the most basic version of the SLIM family. Only one integration method is available (a variable-step method) and features such as function tables, pure delays, the use of arrays, the use of multiple DERIVATIVE sections, sub-model capabilities, sorting etc. are not provided. Such features may be found in many commercially-supported simulation languages. SLIM provides only the most basic simulation facilities but it is a very easily-used simulation tool and, with a little ingenuity, can be applied to a remarkably wide range of problems, as shown by the examples of Chapter 8 in the book and by the case studies in later Chapters.

Detailed practical information concerning the installation of the SLIM software and on the preparation and running of SLIM programs may be found in Section 4 of this Guide. This information is specific to the version of SLIM included on the diskette accompanying the book. Other versions of SLIM have been developed which incorporate additional facilities and are suitable for use with other operating systems, such as VMS and UNIX. A version has also been produced for use on a Parsytec parallel processor system under the PARIX system.

The SLIM software consists of a language processor which has been written in FORTRAN 77 which is capable of translating and interpreting programs written using the special SLIM instruction set. This instruction set involves a small subset of FORTRAN together with a number of the special CSSL instructions needed for simulation applications. SLIM, like many other examples of application-specific software systems, is based upon the use of a syntax-driven parser with a separate executor routine. The language processor first parses the user's program and detects any statements which are invalid. Once the validity of the complete program has been established, the software produces intermediate code which is not seen by the user but which can be interpreted by the executor routine.

2. THE SYNTAX OF THE SLIM LANGUAGE

SLIM follows the principles of the 1967 CSSL standard and the syntax has much in common with ACSL and other widely used languages. There are, however, no explicit INITIAL and TERMINAL statements and the structure of the program is defined only through DYNAMIC, DERIVATIVE, DERIVATIVE END and DYNAMIC END statements.

It should be noted that SLIM follows some of the conventions of FORTRAN and, in accordance with the standard for that language, all program statements in SLIM must begin at, or to the right of, the seventh column of a program line. The first five columns are used only for comments or labels. Comments are indicated by the character 'C' in the first column of the line and are

nored completely in the translation process. Labels must be numeric symbols
d may be entered in any or all of the first five columns. The sixth column
st always contain a space unless the line in question is a comment line.
per case characters must be used throughout, except for comments which may
volve upper or lower case characters, as desired. Blank lines should not
included in a SLIM program and tab facilities should not be used to insert
aces.

additional link with FORTRAN is the fact that the two types of variable
fined within SLIM, integer variables and real variables, are distinguished
cording to the letter of the alphabet with which the variable name starts.
l integer variable names must begin with the letters I,J,K,L,M and N
ereas real variable names must start with any other letter of the alphabet.
al variables and constants may have magnitudes in the range 1.18E-38 to
40E+38 while integer variables and constants must have magnitudes in the
nge 0 to 32768.

riable names may involve any combination of letters and numerals (after the
rst character which must of course be a letter) but can only have six
aracters in total.

xamples of valid real variables are:

```
VALUE
VALUE1
GAIN
ACC2A
```

xamples of valid integer variables are:

```
INVAL5
MPOINT
NFLAG
KP
```

eal constants are exemplified by:

```
1.0
3.142
4.8E-7
6E4
```

nd integer constants by:

```
1
567
22141
```

tatements in SLIM may be divided conveniently into general-purpose
tatements which provide the normal facilities for numerical programming
nd which have counterparts in most high-level programming languages, and
hose statements which are of special significance for simulation.

t should be noted that in the SLIM language expressions are evaluated
according to the normal operator precedence rules. The order of evaluation
s therefore as follows:

```
** raising to a power
*,/ multiplication or division
+,- addition or subtraction
```

It should be noted that SLIM statements must not extend beyond one line of code since there is no facility in the language for any form of continuation line. Complex expressions may therefore have to be split initially into a number of smaller expressions and combined at a later stage in the program.

i) GENERAL-PURPOSE STATEMENTS

a) INTEGER AND REAL ASSIGNMENTS

This type of statement takes the general form:

Real variable = Real expression
or Integer variable = Integer expression

Examples of this type of assignment are:

```
VELOCY=(7.5-DISTCE)/4.982
IVALUE=8+7*K
```

Note that mixed expressions involving combinations of integers with real variables or constants are not permitted and if detected will be rejected as a syntax error.The functions FLOAT and IFIX may be used for data type conversion to overcome this limitation.

Real intrinsic functions available within SLIM which may be included within Real Assignment statements are as follows:

ABS,ALOG,ATAN,COS,EXP,FLOAT,SIN,SQRT and RND.

All these functions except RND are quite standard and have the same role as the correspondingly named functions in FORTRAN. All except FLOAT involve real arguments and give real results. FLOAT requires an integer argument and produces a real result.

The function RND generates uniformly distributed random numbers having values which lie between 0.0 and 1.0. The seed value used for random number generation is determined by the argument used in calling the function. Ten different seed values are available. All values of the argument which are less than 2.0 lead to the use of one particular seed, values equal to or greater than 2.0 but less than 3.0 lead to use of a second predetermined seed value and so on for all other similar intervals in the range 3.0 to 10.0.. All argument values of 10.0 or greater lead to the use of the tenth seed value.

The only integer intrinsic function in SLIM is IFIX. It can be used in the same way as the corresponding function in FORTRAN. It requires a real argument and returns an integer result.

b)INPUT STATEMENT

Input of data in SLIM is from a file named by the user. The form of the Input statement is as follows:

```
ACCEPT variable list
```

The variable list contains variable names which can be of either real or integer type. An example of this type of SLIM assignment is:

```
ACCEPT VALUE,CONST,GAIN
```

)OUTPUT STATEMENT

utput is always written to a named file on disk but may also appear in abular form on screen if desired. The form of the Output statement is:

```
TYPE variable list

.g.   TYPE VALUE,CONST,GAIN
      TYPE J,IVAL
```

he variable list within an Output statement must only involve variables of ne type. A mixture of integer and real variables in a single variable list s not permitted in an Output statement. The first few items in the output ile provide information about the conditions selected within the simulation un and are written to the file automatically. These include three integers roviding system programming information which are not of direct interest to he user followed by the values of the communication interval, the absolute nd relative error parameters used in the program, and the minimum interval.

he next items have to be included in the output file but require the nclusion of TYPE statements in the SLIM source program for the simulation. hey involve, in order, the number of channels in the output (say NCH), the umber of samples used in the simulation run (say NSA) and the communication nterval (say SAM). Every SLIM program must therefore include, at the end of he initial segment, three lines such as:

```
TYPE NCH
TYPE NSA
TYPE SAM
```

where NCH is the integer constant representing the number of output variables (including the independent variable where appropriate), NSA is the integer constant representing the number of samples over the complete time history of each variable and SAM is the real constant representing the communication interval. It is, of course, important that these three quantities be written to the output file in this particular order and that they should be the first items output from the simulation program. It should be noted that in all cases the quantities NSA and SAM must be chosen to be consistent with the interval of the independent variable (usually time) over which the results are required. If, for example the independent variable has an initial value T=0.0 and a final value T=TMAX in the simulation program the quantities NSA and SAM must be such that their product is equal to TMAX.

Immediately after this it is normal to output the initial values of the independent variable and of the system variables of interest. Four TYPE statements therefore normally appear within the initial segment of any SLIM simulation program for which an output file is required.

d) ARITHMETIC IF STATEMENT

The syntax of an Arithmetic IF statement is as follows:

```
IF(arithmetic expression) Label 1, Label 2, Label 3
```

When this type of statement is executed the arithmetic expression is evaluated and control passes to Label 1 if the result is negative, to Label 2 if the result is zero and to Label 3 if the result is positive. Examples of such a statement are as follows:

```
IF(VELOCY-THRESH)175,90,90
IF(IVAL-K)10,20,30
```

In the first case the expression (VELOCY-THRESH) is evaluated and control is passed to the statements in lines labelled 175 or 90 depending on sign of the result. In the second case the integer result from the evaluation of the expression (IVAL-K) determines whether control passes to the statements with labels 10, 20 or 30. It should be noted that in the case of real expressions the condition corresponding to the result being zero is not of practical value due to problems of round-off and other inaccuracies in real arithmetic

e) GOTO STATEMENT

This statement causes the normal sequential process of program statement execution to be interrupted. Control is passed unconditionally to the line which has the label contained in the GOTO statement. For example, the following section of program:

```
     X=1.0
40   TYPE X
     X=(X+10.0)/2.0
     GOTO 40
```

would produce an endless loop which, if executed, would produce an output sequence as follows:

```
5.50
7.75
8.875
9.4375
 etc.
```

f) CONTINUE STATEMENT

A CONTINUE statement is a dummy execution statement and simply causes control to be passed to the next statement in the program.

g) STOP STATEMENT

A STOP statement causes execution to cease. This is usually the last executable statement in a SLIM program. Like all other executable statements it can be given a label.

h) END STATEMENT

The purpose of an END statement is to indicate the end of the program. It must therefore always be the last statement and since it is not executable it cannot be given a label.

ii) SPECIAL-PURPOSE SIMULATION STATEMENTS

a) DYNAMIC STATEMENT

The DYNAMIC statement is used to indicate the start of the dynamic segment of the simulation program. In SLIM only one dynamic segment is allowed within a program. On entering the dynamic segment the processor assigns initial values to the integration variables which are evaluated within the derivative section. The dynamic segment contains the derivative section and the segment is therefore in two parts, one above the derivative section and the other below it.

) DERIVATIVE STATEMENT

he DERIVATIVE statement indicates the beginning of the derivative section hich lies entirely within the dynamic segment. The differential equations nd other equations which define the dynamic simulation model are normally ontained within this section.

:) DERIVATIVE END STATEMENT

he DERIVATIVE END statement defines the end of the derivative section. It ndicates that all the integral assignments have been evaluated and the umerical integration routine is called. On returning from the integration outine the problem variables are all updated. Once control has been passed nto the derivative section it will not be passed beyond the DERIVATIVE END statement until a new communication interval is about to start. If the start of a new communication interval has not been reached control is passed back to the start of the derivative section and the integration process is repeated with the updated variables. This continues until, at the appropriate time determined by the communication interval, control is transferred from the derivative section to the second part of the dynamic segment but without re-assignment of initial values.

d) DYNAMIC END STATEMENT

The DYNAMIC END statement signifies the end of the dynamic segment. In order to enter the region of the SLIM program lying beyond the DYNAMIC END statement control must be transferred explicitly by means of a GOTO or Arithmetic IF statement.

e)INTEGRAL ASSIGNMENT STATEMENT

Each first order differential equation gives rise to an Integral Assignment statement. This type of statement has the form:

Real variable=INTEG(real expression, initial value)

where initial value=real variable or real constant

An example of a statement of this kind could have the following form:

X1=INTEG(V*W,X1INIT)

This indicates that the real expression V*W is to be evaluated and from this the real variable X1 is to be calculated by integration with respect to the independent variable (normally time). The initial condition for this integration operation is provided by the real variable X1INIT which must have been defined at an earlier stage in the program (usually in the initial segment).

f) CINTERVAL STATEMENT

The CINTERVAL statement sets the communication interval which defines the increment of the independent variable at which control is passed from the derivative section to the dynamic segment. The format of this statement is as follows:

CINTERVAL(unsigned real constant or real variable)

For example the statement

CINTERVAL(0.2)

will cause control to be passed from the derivative section to the dynamic segment every 0.2 units of the independent variable (e.g. every 0.2 sec.). The default communication interval is 1.0 and this value is used in any program in which no CINTERVAL is defined explicitly.

It should be noted,incidentally, that the default initial value of the independent variable in a SLIM simulation is zero. Other initial values are possible but must be set by the user using an assignment statement such as:

T=1.6

g) MINTERVAL STATEMENT

This statement defines the smallest step size allowed for a variable step length method. If this lower limit is reached an error message is displayed on the screen and control of the program is returned to the user. The MINTERVAL statement has the general form:

MINTERVAL(unsigned real constant or real variable)

e.g. MINTERVAL(1.0E-5)

The default is MINTERVAL(1.0E-4).

h) XERROR STATEMENT

The XERROR statement is used to set the maximum absolute error that can be accepted in the results of the variable step length numerical integration. The value is defined on a basis of the error per integration step. The variable step length routine used within SLIM either satisfies the requirement defined in the XERROR statement or indicates to the user that the step length has reached the minimum. The form of the statement is:

XERROR(unsigned real constant or real variable)

A typical example is :

XERROR(1.0E-5)

The default value corresponds to XERROR(1.0E-4).

i) MERROR STATEMENT

This statement is similar to XERROR but sets the maximum relative error (per integration step). Again the requirement is satisfied automatically unless the user is otherwise informed through an error message. The format of the MERROR statement is similar to CINTERVAL, MINTERVAL and XERROR and a typical example is:

MERROR(1.0E-4)

which would set the value to the default.

3. OUTPUT FACILITIES AND GRAPHICS

Graphical output may be obtained from the SLIM program by means of a post-processing program which uses the results file from SLIM as input. This

:ogram, called SLIMPLOT, requests the name of the results file, together
ıth information about the channels of the output to be plotted. Both time
istory plots and x-y plots can be produced by SLIMPLOT.

ıe first few items in the output files produced by the SLIM program provide
ıformation about the simulation run and about the internal organisation of
ıe output file, as specified above. This information, which must appear in
ıe standard format described in Section 2 (c) above, provides the post-
rocessing program with essential information which allows it to interpret
ıe simulation data which follows.

. INSTRUCTIONS FOR INSTALLATION AND USE OF SLIM

he software included on this diskette includes the SLIM.EXE program and the
ost-processing program SLIMPLOT.EXE for the display of results in graphical
orm. Example programs discussed in Chapters 6, 8, 10, 11 and 12 of
Continuous System Simulation" by D.J. Murray-Smith may also be found on the
iskette, together with the necessary input data files.

.1 Hardware requirements

'he SLIM and SLIMPLOT programs run on any IBM-compatible personal computer
unning the Microsoft DOS (MS-DOS) operating system. A computer with at least
n 80286 or 80386 microprocessor is preferred, although not essential. It is
lso recommended that the computer has the 80287 or 80387 floating-point
o-processor which will significantly reduce the time required to obtain
esults.

hard disk is desirable and the computer must have CGA, EGA or VGA display
apabilities. The computer should contain a minimum of 512k bytes of RAM
ıemory. A standard ASCII text editor is necessary.

.2 Software installation

'his section of the User's Guide provides practical information about the
steps to be followed in installing the SLIM software on your computer.
'he README.DOC file on the diskette also provides details of procedures for
ınstallation of the SLIM and SLIMPLOT software, including a list of the
ıames of all the files which are provided and any updated features.

SLIM and SLIMPLOT will run successfully from a floppy disk drive but most
ısers prefer to mount the software on their hard disk. To do this you should
reate a new directory (normally named SLIM) on the hard disk. All files on
the distribution diskette should be transferred to this new directory. To
reate such a directory use the normal DOS command:

```
mkdir slim
```

If you wish you could then create separate path-accessible sub-directories
for user files. However, for simplicity, it is assumed in the remainder of
these notes that all SLIM files are in the SLIM directory, that the default
hard disk drive is c: and that floppy disk drive is a:.

To transfer the files from the SLIM diskette to the hard disk place the
diskette in the a: drive and copy all the distribution files into the c:\SLIM
directory as follows:

```
copy a: *.* c:\slim
```

It is important to make back-up copies of all of the files supplied on the distribution diskette prior to attempting to make use of the software. The original diskette should be stored safely.

4.3 Starting and exiting SLIM

Make the SLIM directory the current directory by entering the DOS command cd\slim. When the DOS prompt (in this case C:\SLIM>) appears type:

```
slim
```

and the SLIM program should start to execute. It first requests the name of the SLIM program file to be run. From this point the sequence of operations follows the pattern of the example of Section 6.6.4 of the book. Any error messages relating to errors in translation or occurring at run time should be self explanatory. Syntax errors detected at the translation stage should provide a reference to the line in which the error was found but will not pinpoint the precise nature of the error. It is necessary for the user to examine the relevant section of code to determine why the translation has failed. Comment lines are ignored in the line numbering system for SLIM error messages.

It may be useful to use the VDPOLS.SLI program file from the example in Section 6.6.4 of the book to ensure that you are using SLIM correctly and that the results of Chapter 6 can be reproduced on your own computer.

At the end of a simulation run the user is provided with an opportunity to carry out a further run with the same program through a query of the form:

```
DO YOU WANT ANOTHER RUN? (Y/N)
```

Responding N returns the user to DOS and the DOS prompt appears.

4.4 Useful hints for using SLIM

In preparing simulation programs using the SLIM language features described in Chapter 6 and in this User's Guide it is important to take note of the following points:

a) A SLIM program must follow the conventions explained in Chapter 6 and Section 2 of this Guide. These conventions are similar in many respects to the conventions for FORTRAN 4 or FORTRAN 77 programming.
b) It should be noted that executable code must start in column 7 of each line of program. Columns 1 to 5 are reserved for labels or for use in comment lines. Comments must start with an upper case C in column 1.
c) In creating a SLIM program using a general-purpose text editor, use of the TAB key should be avoided. The space bar should be used to move the cursor to the required column (e.g. column 7).
d) No blank lines should be included in a SLIM program. The presence of a blank line in a SLIM source program causes problems at the translation stage and may cause the computer to 'hang'.
e) Line numbers quoted in connection with error messages relate to lines involving SLIM language statements only. Lines involving comments are not given line numbers.
f) Continuation lines are not permitted in SLIM programs and care must be taken to avoid lengthy statements which involve more than 70 character

or spaces. Attempts to use longer statements in SLIM program may cause the computer to 'hang' at the translation stage. Long statements must be split into two or more statements, each occupying a separate line of code.

g) The distinction between integer and real quantities is important and involves the normal naming conventions of FORTRAN. Real and integer quantities cannot be used in direct combination within a single statement.

h) All executable statements in a SLIM program must involve only upper case letters. Comments may involve upper case and lower case letters but must include an upper case C in column 1.

i) When a syntax error is detected at the program translation stage a number of messages may be displayed which appear to relate to syntax errors in a number of different lines. If there is no apparent reason for an error in one particular line check the statements in the other lines for which syntax errors have been reported. Clearing the error reported in one line may eliminate the apparently meaningless errors reported in other lines.

4.5 Starting and exiting SLIMPLOT

SLIMPLOT is extremely easy to use. It is entered using the following command:

```
slimplot
```

The program then prompts the user for the file name of the SLIM output file which is to be plotted. This is followed by a request to the user to select the type of output file (y versus time or an x-y plot), the number of variables (in the case of an y-t plot) and the column numbers of the variables.

The column numbers referred to in the SLIMPLOT prompts are the column numbers arising from use of the TYPE statement to generate the output data from the SLIM program. For example, if the output is obtained from a statement of the form:

```
TYPE T,X1,X2,X3
```

there will be four columns in the output file. Column 1 involves values of T (the independent variable) and columns 2 to 4 involve the corresponding values of the dependent variables X1, X2 and X3. If one wanted to plot X1 and X3 the columns to be entered in response to the prompt from the SLIMPLOT program would be 2 and 4. These column numbers may be entered as:

```
2,4
```

or

```
2 4
```

Text can be inserted by the user to provide a main (top) title and up to two titles which appear below the graph. Both upper and lower case characters may be used in these titles.

When graphical output has been displayed on the screen and the user wishes to move on to a further stage of processing pressing the ENTER key clears the graphical output from the screen and causes a new menu to appear. This has one option which allows the user to exit from the SLIMPLOT program and others which allow further plots to be produced.

4.6 Useful hints for using SLIMPLOT

The following points should be noted:

a) Problems will be encountered if the number of time steps in a SLIM output file is greater than the maximum allowed by SLIMPLOT. If the maximum is exceeded an error message is displayed during the running of the SLIMPLOT program. The SLIM program should then be re-run with an increased communication interval (CINTERVAL) or a smaller range for the independent variable. For problems where the independent variable is time this means either increasing the time step for the plot or reducing the length of the simulation run.

b) Hard copy output from SLIMPLOT is most easily obtained using the DOS GRAPHICS command and the use of the Print Screen key. An alternative, if an appropriate printer is available, is to make use of the printer or plotter facilities in the list of output devices which is displayed during the running of SLIMPLOT.

5. EXAMPLES

Chapters 6, 8, 10, 11 and 12 of "Continuous System Simulation" include examples of SLIM programs. SLIM program files and data files for these examples are provided on this diskette. The program files all involve file names with a .SLI extension, while the input data files all involve files with a .IN extension. Filenames are the same as those used in the relevant chapters. It should be possible in each case to reproduce results presented in tabular or graphical form in the book.

6. COMPARISONS OF SLIM WITH WIDELY-USED SIMULATION LANGUAGES

SLIM is intended to provide beginners with an introduction the principles of continuous system simulation and has been designed as an inexpensive but reasonably versatile teaching tool. It is not intended to compete directly with commercially-supported languages. SLIM is most suitable for relatively small simulation problems. It does conform to many of the conventions of the CSSL specification and provides a convenient, efficient and easily-used tool. Having gained some initial experience with SLIM the user should be able to move on without difficulty to make use of other more powerful languages, such as ACSL.

The maximum number of state variables which can be accommodated within a SLI simulation program using the version of the SLIM software which accompanies the book "Continuous System Simulation" is 40. This is not seen as imposing any serious restriction on the average beginner. All of the examples and cas studies considered involve less than ten state variables. The maximum number of statements allowed within a SLIM program is 250. Once again typical applications involve much smaller programs. Some other simulation languages, such as ACSL, do not impose limitations of this kind.

While the restrictions in terms of program size are not thought to represent limits which are likely to be approached by the beginner, it is important to understand that languages such as CSSL IV, ACSL and DESIRE provide many features which are not available within SLIM. Some of these restrict the type of system which can be modelled using SLIM and others limit the options available to the user at run time. The sub-set of facilities incorporated in SLIM has been chosen to allow the beginner to gain experience rapidly, without having to deal with the complexities associated with features in mor comprehensive languages. A careful examination of the user manual for each specific language allows the potential user to establish the precise differences in every case.

ıe absence of facilities to model a pure time delay (transport delay) is one nportant example of a feature, available in many simulation tools, which has ɔt been included within the version of SLIM provided with "Continuous ystem Simulation". The lack of any facility to represent a function in abular form is another example of a potentially useful feature which has ɔt been included. Sorting facilities and facilities for sub-models are also ɔt provided within SLIM. Simulation problems which require the use of acilities such as transport delays and function tables cannot therefore be ackled using SLIM, unless the problem can be reformulated in some way to llow it to be solved within the limitations of the language. Some other anguages, such as ACSL, provide a much more extensive set of facilities nd therefore provide much more flexibility at the model definition stage.

n comparison with ACSL and other similar languages SLIM has limited apabilities in terms of run-time options. There is no real separation of he model definition and the experiment in SLIM. Changes of parameters equire editing of the simulation program itself or changes of an associated ata file. However, flexibility is available in terms of output display acilities in SLIM because of the fact that the graphical output options nvolve a separate post-processing program.

LIM can be extremely effective when learning about the principles of imulation and when used for the solution of problems for which its acilities are appropriate. It is important, however, to understand its imitations and to have an appreciation of the additional facilities of the oftware products which are available commercially.

. DISCLAIMER

ll parties must rely on their own skill and judgement when making use of he SLIM and SLIMPLOT software. Neither the author nor the publishers ssume any liability to anyone for any loss or damage caused by any error r omission in the work, whether such error or omission is the result of egligence or any other cause. Any and all such liability is disclaimed.

. ACKNOWLEDGEMENTS

'he features included in this version of the SLIM language are, as explained n Section 1 of this User's Guide, a subset of the features considered lesirable for equation-oriented continuous system simulation languages, as ecommended in 1967 by the relevant Simulation Councils inc. technical ommittee. In considering the features to incorporate in this simulation anguage, which has been designed specifically for teaching applications, number of existing simulation tools were reviewed. One existing language hich has had a particularly strong influence on SLIM is the MIMESIS anguage which was developed by Ian Ricketts at the University of Dundee and as described in his Ph.D. dissertation in 1977 (Ricketts,I.W. "MIMESIS - A ontinuous System Simulation Language", Ph.D. Thesis, University of Dundee, 977). SLIM and MIMESIS are both based on the principles of syntax-driven arsers as described by R.L. Gauthier and R.D. Ponto in their text entitled 'Designing Systems Programs" (Gauthier, R.L. and Ponto, R.D. "Designing ystem Programs", Prentice-Hall, New Jersey, 1970). Many features of the nstruction set of SLIM are similar to those of MIMESIS, although the internal tructure of the SLIM program and the detailed coding are different. Experience gained in using the MIMESIS software has been of particular value in the development of SLIM and the author is very grateful to Dr. Ricketts for making MIMESIS available.

The author also wishes to record his thanks to present and former students in the Department of Electronics and Electrical Engineering at the University

of Glasgow who have assisted in the development of the SLIM software. A number of student projects have been associated with the application of SLIM to practical engineering problems and these have provided valuable information about the range of problems which can be tackled conveniently using this software. Very helpful feedback has also been provided freely by students who have used SLIM in connection with courses in computer simulation methods, in control engineering and in avionic systems at undergraduate and post-graduate level.

The software has undergone much revision and modification during the course of its development. Work on SLIM began in 1991 during a period of sabbatical leave which was spent at the Technical University of Vienna. The author is very grateful to Professor Dr. Felix Breitenecker of the Department of Simulation Techniques at the Technical University of Vienna for his assistance and support in making that visit possible and to the University of Glasgow for providing the necessary leave of absence. Without the opportunity provided by this period of uninterrupted work it is unlikely that SLIM would have been developed.

The graphical routines included in the SLIMPLOT program are part of the NAG Graphics Library. This product contains a collection of over 100 carefully designed routines which provides FORTRAN and C programmers with a versatile means of producing a graphical representation of numerical and statistical results. It is used worldwide in commerce, financial modelling, industry academic research and many other areas, and is available for a wide spectrum of machines. Interfaces are also supplied to the most commonly used plotting packages. NAG also provide a wide range of other products such as mathematical and statistical routines in FORTRAN, C, Pascal and Ada; FORTRAN 90 compilers and tools; symbolic solvers; data visualization software; linear programming software and a gateway generator which automatically generates MATLAB gateway. For further information, a quotation or details of your local distributor please contact:

Sales Department
NAG Ltd.
Wilkinson House
Jordan Hill Road
Oxford OX2 9BX
United Kingdom.

1. Was ist Fuzzy-Logik?

FUZZY-LOGIK

Methodik des sog. unscharfen Schließens

Subjektive Unbestimmtheit von Begriffen

Mathematisch fundierte Methode; erlaubt graduierbare oder vage Prädikate

So far as the laws of mathematic refer to reality they are not cer-tain;
And so far as they are certain they do not refer to reality
(A. Einstein)

1. Beispiel:

Tatsache: Sokrates war ein Mensch

Wissen: Menschen sind sterblich

Schlußfolgerung: Sokrates ist gestorben

2. Beispiel:

Tatsache: Die Tomate hat eine leicht rötliche Farbe

Wissen: Eine rote Tomate ist reif

Schlußfolgerung: Die Tomate ist nicht ganz reif

Klassische Logik versus Fuzzy Logik

Scharfe Menge charakterisiert durch zweiwertige Funktion

f_A:(x) --> { 0,1 }

Unscharfe Mengen charakterisiert durch Zugehörigkeitsfunktion

/uA(X)

Die Funktion $f_A(x)$ wird als zweiwertig bezeichnet, da sie entweder den Wert 0 oder 1 annimmt, je nachdem ob das Element x zur Menge A gehört oder nicht. Die Menge { 0,1 } ist die Wertemenge von A.
Ist als Wartemenge das gesamte Intervall [0,1] zugelassen, geht die scharfe Menge A über in die unscharfe Menge A.
Eine unscharfe Menge A wird durch eine verallgemeinerte charakteristische Funktuion /uA gekennzeichmet: X: --> [0,1], die **Zugehörigkeitsfunktion** von A genannt wird und über einen (von Fall zu Fall geeignet festzulegenden) Grundbereich $_X$ definiert ist.
Je dichter der Wert /uA(X) zu dem Wert 1 tendiert, desto mehr entspricht x der Charakteristik der Menge A.

DEFINITION:

Sei X ein unscharfe Menge.

Eine unscharfe Menge (fuzzy set) A über X wird charakterisiert durch eine Zugehörigkeitsfunktion /uA(X), die jedem Element aus X eine relle Zahl aus dem Intervall | 0,1 | zuordnet.
Der Wert von /uA an der Stelle X wird *Zugehörigkeitsgrad* von X zur Menge A bezeichnet.

3. Beispiel:

Sie wollen sich ein Cello kaufen. Die von Ihnen favorisierten Instrumente liegen zwischen DM 3.500,00 und 8.000,00.

GRUNDMENGE:

X={3500,--,4500,--,5600,--,7200,--,8000,--,8500,--}

UNSCHARFE MENGE:

A={(3500,--,0.3),(4500,--,0.6),(5600,--,0.9),(7200,--,0.4),
(8000,--,0.3),(8500,--,0.1)}

FUZZY-LOGIK

ist ein expandierender Markt!

1988 begann in Japan die Verbreitung des industriellen Einsatzes einer neuartigen Logik, die nicht mehr auf den absoluten Aussagen der monokontexturellen Logik „wahr" und „falsch" bzw. „ja"und „nein" beruhte. Für diesen neuen Logiktyp wurde der Begriff FUZZY-LOGIK eingeführt, der am treffendsten mit „unscharfe Logik" übersetzt werden kann.

FUZZY-LOGIC Marktanteile

- Es ist zu erwarten, daß 1995 für mehr als 2,5 Mrd DM Geräte im Bereich der Konsumelektronik verkauft werden, z. B. Audio, Video, Haushalt, etc., mit einem Anteil von 2 - 15 DM an Bauteilen der Mikroelektronik

- Es ist zu erwarten, daß 1995 der Gesamtmarkt von Geräten und Systemen mit FUZZY-LOGIK-Bausteinen eine Größenordnung von ca. 6 Mrd DM haben wird.

- Im Jahr 2000 wird der Weltmarktanteil für „FUZZY-LOGIK-HALBLEITER" auf ca. 15 Mrd DM geschätzt

Der Erfolg der FUZZY-LOGIC erklärt sich aus den vielen Vorteilen, die diese Logik mit sich bringt. Die mit FUZZY-LOGIC realisierten Geräte sind beispielsweise wesentlich

- benutzerfreundlicher als Geräte, die auf den bisherigen klassischen Verfahren basieren

- robuster und erfordern weniger Entwicklungszeit, bevor sie in den Markt eingeführt werden könnten, so daß eine Just-in-Time-Entwicklung möglich ist

- weniger Regeln als vergleichbare Expertensysteme z. B. bei Diagnoseanwendungen (Performancevorteil)

FUZZY-LOGIC basierte Regelung

- FUZZY-LOGIC wird z. Z. in hochentwickelten Systemen eingesetzt und repräsentiert bereits heute den nächsten Schritt der Entwicklung des sogenannten „Embedded Control"

- Hochentwickelte Regler sind das erste Ziel für State-of-the-Art-Controller die auf FUZZY-LOGIC basieren

- Bereits für 1994 wird von einer Kostenreduzierung für FUZZY-LOGIC-Bauteile erwartet, die es erlauben, auch das Segment für die Routine-Regelung einzubeziehen, sogenanntes LOW-END-CONTROLLER-Segment

- Insgesamt kann der FUZZY-LOGIC-Anteil im Controller-Segment bereits heute auf ca. 40 % abgeschätzt werden

2. Fuzzy-Logik in USA,J,D!

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

USA

Realisierte Projekte:

Simulation der automatischen Ankopplung einer Station im Weltall mit FUZZY CONTROL bei der NASA

Roboteranwendungen maschinelles Sehen mittels FUZZY LOGIK bei der NASA

Zugangskontrolle mittels FUZZY LOGIK im Los Alamos National Laboratory

Reaktorsteuerung mittels FUZZY CONTROL beim MIT

TU Clausthal IfI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Forschergruppen die sich mit der FUZZY LOGIK beschäftigen

University of Alabama, Birmingham

AT&T Bell Laboratories, Whippany

University of California, Berkeley (L.A.Zadeh)

University of Cincinnati

Florida State University, Tallhassee

City University of New York

Iona College, New Rochalle

State University of New York, Binghamton

North carolina State University

University of Saskatchewan

University of Southern California (B.Kosko)

Wayne State University, Detroit

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

- Expertensysteme für *fuzzy pattern recognition* und *fuzzy scene analysis*,
- eine *Fuzzy*–Experten–Shell für allgemeine Anwendungen,
- unscharfe intelligente Differentialgleichungslöser,
- *Fuzzy Controller* für industrielle Prozesse,
- ein *Fuzzy*–Expertensystem, um die Produktivität bei der Software–Entwicklung zu steigern,
- unscharf vernetzte Datenbasen für "intelligent tutors"–Anwendungen,
- unscharfe Expertensysteme für Super–Computer–Anwendungen,
- theoretische Untersuchungen für unscharfe Schlußfolgerungs–Mechanismen,
- Studien über unscharfe Relationen zur Rettung verlorengegangener Informationen.

Die Aufgeschlossenheit der Japaner gegenüber Fuzzy–Logik im Vergleich zu Deutschland (Europa) kann durch folgende Zitate begründet werden:
Prof. Bart Kosko, University of Southern California: "*Fuzzyness begins where Western logic ends*".
Dr. Nobori Wakami, Seniorforscher bei Matsushita: "*The concept of fuzzyness is like soy sauce and sushi — a perfect match*".
Prof. L.A. Zadeh: "*LIFE will be the world's premier center of fuzzy logic development*".
Prof. M. Sugeno, Tokyo Institute of Technologie: "*In expert systems here in Japan we have about 2000 projects, but only two percent or about 40 were successful. In fuzzy logic we have already had 80 successful projects out of 100. About 90 percent of those are in control engineering and the others are ordinary expert and decision support systems.*"

Steuerungs- und Regelungssysteme

Sinterofen, Zementdrehofen, Wärmeaustauscher, Druckkessel, Atomkraftwerke, automatische Kurssteuerung, Kraftfahrzeuge, Zugverkehr, Wasseraufbereitungsanlagen, Trinkwasserversorgungssysteme, Glasschmelzer, elektrische Haushaltsgeräte, Gasabsperrventile, Selbstanlasser

Künstliche Intelligenz

Mustererkennung, semantische Analyse natürlicher Sprachen, Kanji-Erkennung, Experten-Systeme, Erkennen landwirtschaftlicher Produkte, Müllsammelautomat, Pflegeroboter (Domnestic Robot)

Konsultationssysteme

Häusliche Beratungssysteme, CAI, CAD, Entscheidung des günstigsten Seeweges, verschiedene Optima dynamischer Pläne (Durchlauf, Produktion, Kontrolle), Unterstützung der Betriebsleitung bei der Entscheidungsfindung, Sicherheitsbestimmung eines Gebäudes, Ausrüstungsdiagnose, Fehlersuche, Suchgeräte zum Auffinden fehlerhafter Metalle, Qualitätsfestlegungen für Nutzholz, Gesundheitspflegesystem in einer Gesellschaft mit überwiegend älteren Menschen, Hilfestellung

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Medizinische Systeme

Zahnmedizinische Diagnose, Diagnose und Behandlung von Herzproblemen, Diagnose von Leberkrankheiten, medizinische Bildübersetzung, Pankreas-Behandlung, Blutübertragungsberatung, Optimale Probeentnahme, Bestrahlungsmessung, Untersuchung medizinischer Behandlungsmethoden

Modelle menschlichen Verhaltens

Soziale Modelle

Analyse des in der Öffentlichkeit herrschenden Bewußtseins, Systeme zur Gesundheitspflege, ...

.
.
.

Datenbankauswertungen

Semantik

.
.
.

1. Laboratorium: *Fuzzy Control*

Unter der Leitung von Dr. Takeshi Yamanaka von der Omron Tateishi Electronics Co. befaßt sich dieses Laboratorium mit den folgenden Themen:

- Studium der Fuzzy Control–Theorie, mit dem Schwerpunkt auf der Untersuchung der Stabilität von Fuzzy Control–Systemen.
- Entwicklung und Aufbau von Hilfsmitteln zur Unterstützung des Entwurfs von Fuzzy Control–Systemen (Benutzeroberflächen, Entwicklungssysteme, o.ä.)
- Entwurf, Aufbau sowie Auswertung von konkreten Anwendungsbeispielen für die Fuzzy Logic in Form eines "unscharfen" Roboters und eines "unscharfen" Kraftwerk–Kontroll-Systems im Labormaßstab.

2. Laboratorium: *Fuzzy Intellectual Information Processing*

Die Arbeitsgebiete des zweiten Laboratoriums unter der Führung von Dr. Tomohiro Takagi von der Matsushita Electric Industrial Co., Ltd. gliedern sich wie folgt:

- Bildererkennung,
- Systeme zur Unterstützung von Entscheidungsfindungsprozessen,
- Fuzzy Expertensysteme.

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

3. Laboratorium: *Fuzzy Computer*

Dr. Seiji Yasonobu von der Firma Hitachi, Ltd. beschäftigt sich mit seinen Mitarbeitern im wesentlichen mit folgenden Aufgaben:

- Fuzzy Computer–Architektur,
- Fuzzy Software,
- Fuzzy Hardware.

TU Clausthal IfI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller
FG: FUZZY-LOGIK

DEUTSCHLAND

Aachen
Lehrstuhl für Operations-Research (H.J.Zimmermann)
Erforschung von Operatoren und linguistischen Modifikatoren, mathematische Ableitung von Operatoren, Entscheidungsfindung, Entscheidungsunterstützungssysteme, Fuzzy Set Theory, Fuzzy Control

MIT (Management Intelligenter Technologien)

ELITE (European Laboratory for Intelligent Techniques Engineering)

Braunschweig
Fuzzy Arbeitsgruppe für Fuzzy Information (R.Kruse)

Chemnitz
An der FhG IUW und der TU Fuzzy Klassifikation (U.Priber)

Clausthal-Zellerfeld
Lehrstuhl für Technische Informatik (D.P.F.Möller)
Prozeßautomatisierung, Medizintechnik, Datenbanken

Dortmund
Lehrstuhl für Elektrische Steuerung und Regelung (H.Kiendl)
Fuzzy Control, Stabilitätsuntersuchungen, automatische Regelgenerierung

Lehrstuhl Informatik I (B.Reusch)
Zielorientierte Entscheidungshilfe, Schaltungsentwurf mittels Fuzzy Logik, Expertensysteme

FDZ (Fuzzy DEmonstrationszentrum Dortmund)

Erlangen, München, Passau
Neuronale Netze und Fuzzy Logik (FORWISS), Fuzzy Mini Messen, Special Interest Groups (P.Protzel)

Leipzig
Fuzzy Set Theory, Fuzzifizierung von Petri Netzen, Datenanalyse mit Fuzzy-Methoden (S.Gottwald)

München
Siemens AG, ZFE Fuzzy Task Force Gruppe (D.Reinfrank)
Prüfung der Fuzzy Logik für die Belange des Siemens Konzerns

dto. Daimler-Benz Forschungszentrum Berlin (W.Lohnert)

versität Braunschweig,

- *Fuzzy Topology*, von U. Hoele, Bergische Universität Wuppertal,
- *Theory of Fuzzy Measures* von S. Weber, Fakultät für Mathematik der Universität Mainz,
- *Evidential Conditional Reasoning*, von M. Spies, IBM Scientific Center, Heidelberg,
- *Psychological Aspects of Fuzzy Reasoning*, von A. Zimmer und H. Koerudle, Institut für Psychologie der Universität von Regensburg,
- Unscharfe Modellbildung technischer Prozesse von H.P. Lipp, TH Chemnitz,
- Unscharfe Mengen von S. Gottwald, Univ. Leipzig

3. Strukturelle Gliederung der Fuzzy-Applikationen

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Jahr	JAPAN	EUROPA	USA
1991	450	225	150
1995	1500	680	750
2000	4500	5250	2250

Fuzzy-Märkte (in Mio. US $)

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Robotik	6.8
Entscheidungsunterstützung	6.8
Engineering	6.7
Prozeßregelung	6.6
Spracherkennung	6.4
Risikoanalyse	6.3
Natürliche Sprachinterpret.	6.3
Textinterpretation	6.3
Medizin	5.8
FuE	5.1
Finanzen u. Ökonomie	5.1
Soziologie	4.9
Luft- u. Raumfahrt	4.8
Energie	4.4
Mathematik	4.2
Transport	4.1
Geologie	3.6
Chemie	2.4

Strukturelle Gliederung der Fuzzy-Applikationen bezüglich ihrer kommerziellen Relevanz (10=dominant, 1=uninteressan)

Quelle Frost&Sullivan 1992

Strukturelle Gliederung der Fuzzy-Applikationen:

Umsetzungsfördernde Kriterien für Fuzzy-Control:

- einfacher Zugang zu meß- und regeltechnischen Problemstellungen auf der Basis menschlicher Anschauungsweise,
- Implementierung von Erfahrungswissen,
- potentiell Reduktion der Entwicklungszeit,
- Möglichkeit neuer oder verbesserter Funktionaliät,
- Breite der Anwendungsfelder.

Indikatoren zur Identifikation sinnvoller Anwendungsfelder:

- fehlende mathematische Modelle,
- nichtlineare Problemstellungen,
- eingeschränkte Sensorik,
- totzeitbehaftete Systeme.

Umsetzungshemmende Defizite:

- Fehlen allgemeiner Entwurfs- und Optimierungsstrategien,
- Fehlen von Kriterien zur Güte-beurteilung und zum Stabilitäts-nachweis,
- Fehlen von Normungs- und Zulassungskriterien.

trukturelle Gliederung
er Fuzzy-Applikationen:

usgewählte Anwendungsgebiete)

utomobilsektor

erfahrenstechnik

teuerungstechnik

aschinenbau

ideotechnik

hotoapparate

aushaltsgeräte

ustererkennung

ntwicklungstendenzen:

ntelligente Sensorik und Aktorik

nbindung an Standard-Schnitt-
tellen

ntegration in Werkzeugumgebung

ntegration in Datenbanken

Automobilsektor Im Jahre 1990, angeregt durch die Veröffentlichungen von Nissan, Subaru und Honda, beschäftigten sich alle deutschen Automobilfirmen und deren Zulieferer mit den Problemkreisen von Antiblockiersystemen, Antischlupfregelung, automatischen Schaltgetrieben, Klimaanlagen sowie mit Abstandsmeßsystemen in Verbindung mit automatischem Kolonnenfahren. Auch die aktive Fahrwerksdämpfung wurde untersucht. Es gab Versuche, wie die Stabilität der Straßenlage durch ein Fuzzy-Logik-System verbessert werden kann. Eine interessante Anwendung stellt die sogenannte "Kollisions-Vermeidung" dar, die von verschiedenen Herstellern mit unterschiedlichem Erfolg untersucht wurde.

Besondere Beachtung fand das von Mazda in Japan vorgestellte automatische Viergang-Schaltgetriebe, das auch die für eine Automatik problematischen Bereiche "Bergauf-Fahren", "Bergab-Fahren" hervorragend beherrschte. Siemens berichtete anläßlich des 1. Berliner FAN-Treffens von den Erfolgen in einem umgebauten Serienfahrzeug. Das Fuzzy-System wählt zwischen den beiden Kennlinienfeldern "wirtschaftlicher Fahrer" und "sportlicher Fahrer" aus, und ist insbesondere am Berg durch das im Vergleich zur konventionellen Automatik wesentlich effizienter schaltende Getriebe deutlich überlegen.

Im Bereich der Last- und Nutzfahrzeuge wurde insbesondere die Problematik der beliebig variierbaren Gewichtsverteilung in Schaufeln, Anhängern oder Transportmedien untersucht. Potentielle Anwendungen sind z.B. das Transportieren von Mülltonnen (die unterschiedlich gefüllt sind) oder das Transportieren von Baggerschaufeln (ähnliche Anwendung).

Die Implementierungen reichen hier von der reinen Softwarelösung auf den traditionellen Meßsystemen in Prototyp-Fahrzeugen bis zu bislang nur "angedachten" Implementationen auf Mikrocontroller-Ebene. Daneben gibt es Studien zu Kombinationen von Fuzzy-Logik und Neuronalen Netze, z.B. auf dem Gebiet der automatische Anfahrhilfe, des führerlosen Fahrzeugs (Volkswagen) oder für das Kolonnenfahren innerhalb des Projektes Prometheus (Opel). Hier wurden sowohl der Tempomat als auch der Abstandsregler mittels Fuzzy-Logik sehr effizient gelöst. Dabei handelt es sich hier noch um einen Versuch, vom Einsatz in der Serie sind diese Regler noch weit entfernt. Dies ist nicht zuletzt auch darauf zurückzuführen, daß die heute realisierte Abstandsmessung mit Laser noch nicht kostengünstig für die Serie zu produzieren ist.

Die Automobil-Firmen und ihre Zulieferer sind sowieso mit der Bekanntgabe ihrer Ergebnisse recht zurückhaltend, und da die Zeit bis zum Einführen eines neuen Automobilmodells relativ lange ist, werden wohl auch in der nächsten Zeit noch keine weitreichenden Ergebnisse bekannt werden.

Verfahrenstechnik in der chemischen Industrie Im chemischen Anlagenbau gibt es eine Vielzahl von Prozessen, die durch einen bewährten Anlagenfahrer bedient werden, und die mit konventionellen Expertensystemen nicht oder nur sehr schwer zu beherrschen sind. Z.B. tritt in chemischen Anwendungen eine sogenannte exotherme Reaktion auf, die mit konventionellen Reglern bedient, aber in diesem Falle nicht oder kaum noch beherrschbar ist. In diesem Falle tritt der Fuzzy-Regler parallel zum konventionell aufgebauten Regler auf und korrigiert bzw. führt den konventionellen Regler wieder in den stabilen Bereich.

Wenn man sich das Prinzip des Brennofens vor Augen führt (dieses Verfahren wurden bereits vor ca. 10 Jahren in Dänemark mit Erfolg eingesetzt), so gibt es eine Vielzahl von ähnlichen Anwendungen, wie die Ergebnisse der Firmen BASF, AKZO, BÖHRINGER und BAYER zeigen. Der Stand in diesen Unternehmen ist heute so, daß z.T. Pilotanwendugen vorliegen, über deren Vorteile auf vielen Tagungen bereits gesprochen wird. In einem solchen Pilotprojekt versucht die EC-Erdölchemie GmbH, Köln in Kooperation mit MIT (Management Intelligenter Technologien GmbH) die Entkockung der Öfen mit Hilfe von Fuzzy Datenanalyse Methoden zu automatisieren. Darüber hinaus soll das Energiemanagement des Krackers mit diesen Methoden verbessert werden.
Aber noch sind nicht alle Unternehmen bereits auf einem für die allgemeine Anwendung wünschenswerten Wissensstand. Das weite Aufgabenfeld der "wissensgeführten" Anlagentechnik läßt in Zukunft eine breite Anwendung der Fuzzy-Logik erwarten.

Hersteller von Steuerungen Daß die japanische Firma OMRON die Fuzzy-Logik bereits vor Jahren forciert hat, um ihre eigenen Produkte zu verbessern, darf als bekannt vorausgesetzt werden. In Deutschland war die Interkama 1992 für einige Firmen der Start in die Fuzzy-logik. Während Siemens bereits seit ca. 2 Jahren mit Togai InfraLogic mit einem Fuzzy-Hersteller intensiv zusammenarbeitet und auf der Interkama für ihre Systeme Teleperm-M fertige Fuzzy-Systeme mit bereits realisierten Anwendungen präsentierte, zeigten die Firmen Hartmann & Braun und Klöckner-Möller in Zusammenarbeit mit Herstellern von Systemen die Integration von existierenden Entwicklungsoberflächen (TILShell bei Hartmann & Braun, fuzzyTECH bei Klöckner-Möller) in ihre Systeme. Daneben sind weitere Projekte vorgesehen. Der Vorteil für diese Art der Integration liegt darin, daß der Kundenkreis der Steuerungshersteller eine erweiterte Möglichkeit hat, komplexe Systemproblematiken zu programmieren.

Eine weitere auf diesem Gebiet aktive Firma ist ABB in Mannheim, die die Fuzzy-Logik ihren Kunden als neues Leistungsmerkmal anbietet. Zusammen mit der Universität Zittau wurde ein Fuzzy-Controller mit wissensbasiertem analytischen Regler (WAR) entwickelt, der die Vorteile des konventionellen Reglers nutzt und gleichzeitig dessen Nachteile umgeht, indem statt der "Fuzzy-Sets" verschiedene Stellgrößen analytisch ermittelt werden.

Siemens sieht die Fuzzy-Technologie als eine der wichtigsten Kerntechnologien für den Konzern an. In einer von Dr. Reinfrank im Zentralbereich ZFE geleiteten Arbeitsgruppe von 20 Personen wird Applikationsunterstützung für alle Unternehmensbereiche angeboten. Die Gruppe Automatisierungstechnik in Erlangen hat beispielsweise eine Steuerung von Zellstoffkochern in Portugal entwickelt, und zwar nicht als reine Fuzzy-Logik-Steuerung, sondern als Erweiterung konventioneller Systeme auf der Basis der Prozeßautomatisierungssysteme Teleperm-M. In Planung ist die Realisierung für die Serie Simatic S5. Siemens hat überdies mit der Fuzzy-Steuerung SIFLOC ein eigenes Produkt als Projektierungswerkzeug für Labor und reale Prozeße entwickelt.

Es zeichnet sich ab, daß die Hersteller industrieller Steuerungen Fuzzy-Komponenten in ihre Produkte integrieren und kompakte Fuzzy-Steuerungen einsetzen werden. Von den SPS-Hersteller wird oftmals bemängelt, daß die Anwender zwar die reine Steuerung mit den SPS-Geräten erledigen, aber komplexe Regelungen mit anderen Hilfsmitteln realisieren. Da oft auf die Erfahrung von praxiserprobten Regelungstechnikern zurückgriffen wird, was sich natürlich im Preis für komplexe Anlagen auswirkt, reizt natürlich die Fuzzy-Logik die SPS-Hersteller, ihren Anwendern die Möglichkeit zu verkaufen, die Regelung durch vor Ort vorhandene Techniker einstellen zu lassen, da ja für die Fuzzy-Logik-basierten Systeme kein Systemwissen, sondern nur Anwendungswissen notwendig ist.

eine Reihe von Anwendungen in diesem Bereich realisiert:

1. Automatische Belichtungssteuerung, die insbesondere den kritischen Bereich mit Gegenlicht, Streulicht usw. abdeckt.

2. Autofokus: Fokussieren auf bewegte Objekte, insbesondere sich schnell aus dem Zentrum entfernende Objekte.

3. Kompensation von Verwackelungen: Diese elektronische Korrektur war am Anfang wohl die interessanteste Fuzzy-Anwendung. Es existieren bei den vorhandenen Systemen Methoden, den Bildinhalt elektronisch zu vergrößern, um einen Bereich für die elektronische Korrektur zu haben. Leider ergibt sich hier eine deutliche Verschlechterung der Bildqualität, da die Auflösung entsprechend dem gewählten Ausschnitt verschlechtert wird. Damit waren auch insgesamt die Testergebnisse für den elektronischen Verwackelungsschutz nicht zufriedenstellend ("man schaltet die Verwackelungsautomatik am besten ab"). Inzwischen sind allerdings gut funkionierende Verfahren bekannt, die die Verwackelung mit einem mechanischen Spiegel durch elektronische Korrektur kompensieren. Eine Verschlechterung der Bildqualität ergibt sich dadurch nicht.

4. Farbkorrektur: Die Fuzzy-Logik wird als Hilfsmittel benutzt, um eine optimale Umstellung der Farben vorzunehmen, beispielsweise wenn von einer Kunstlichtumgebung in eine Tageslichtszene gefilmt wird. Jeder Videoamateur kennt die Problematik des Blau- oder Rotstichs bei falsch gewählter Farbtemperatur.

Maschinenbau Der Maschinenbau-Markt, in Deutschland einer der wichtigsten Märkte, hat bislang noch nicht die Vorteile der Fuzzy-Logik verstanden, hauptsächlich deswegen, weil die konventionelle Regelungstechnik eine Technik darstellt, die etabliert ist und als beherrschbar gilt. Deshalb werden zunächst die Fuzzy-Systeme daraufhin untersucht, ob sie die konventionellen Systeme ersetzen können. Da die Fuzzy-Technik als Ergänzung und Erweiterung für die konventionelle Regelungstechnik zu sehen ist, insbesondere da, wo man es mit strukturinstabilen Systemen zu tun hat, oder wo kein vernünftiges Modell existiert, ist dies zunächst ein unglücklicher, eher sogar falscher Ansatz.

Der zweite Grund, warum die Maschinenbauer noch nicht von den Vorteilen der Fuzzy-Logik profitieren können, ist darin zu sehen, daß die Maschinenbauer traditionell ihre Elektronik zukaufen, und zwar weniger in Spezial-Lösungen, sondern bevorzugt durch die SPS-Anbieter. Die notwendigen Programmierung geschieht auf der SPS-Seite, und hierfür gibt es derzeit noch kaum Fuzzy-Logik-Unterstützung. Die einzige uns bislang bekannte Ausnahme ist die Teleperm-M von Siemens (Anwendungsbereich chemischer Anlagenbau). Eine Portierung der FPL (Fuzzy Programming Language) von Togai auf die Programmiersprache STEP5 der Siemens S5 wurde angekündigt.

Da aber die Fuzzy-Logik in den Steuerungsgeräten mehr und mehr berücksichtigt wird, dürfte hier bald eine Verbesserung der Situation erfolgen.

Haushaltsgeräte In Japan gibt es den intelligenten Reiskocher, der aus der Reis- und Wassermenge sowie der vom Bediener gewünschten Körnigkeit des Reises die optimale Garzeit einstellt - via Fuzzy-Logik mit einem kundenspezifischen Schaltkreis.

Bei Staubsaugern paßt die Fuzzy-Logik die Saugleistung dem jeweiligen Verschmutzungsgrad des zu saugenden Objekts an. Das zu saugende Objekt (Teppich, Fliesen, Polstermöbel u.a.) wird über spezielle Sensorik erkannt. Ein Modell wurde mit etwa 40.000 verschiedenen Teppichen und Untergründen trainiert, um daraus automatisch die optimale Einstellung der Saugdüsen, des Motors etc. zu erhalten. Ein einfacher Fotowiderstand mit Infrarotdiode ermittelt den augenblicklichen Staubgrad der angesaugten Luft. Damit wird automatisch die Motorleistung verringert oder erhöht.

Mit Blick auf die Verkaufserfolge in Japan werden die Hersteller von Konsumprodukten voraussichtlich auf der nächsten Domotechnika (Februar 1993) ihre Produkte auf dem Markt zu plazieren versuchen: dazu gehören Mikrowellen-Geräte, Geschirrspüler, Waschmaschinen, Staubsauger, Elektronik-Herde und ähnliche Objekte. Da in diese Produkten der Kostenanteil der Elektronik als äußerst niedrig anzusetzen ist, wird die Implementation entweder als reines ASIC realisiert oder auf 4-Bit-Mikrocontrollern basieren.

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Mustererkennung mit Fuzzy-Logik Zur Untersuchung mechanischer Systeme wird die akustische Analyse als Stand der heutigen Technik eingesetzt. Während Erfassung und Darstellung der Daten im allgemeinen heute kein Problem mehr ist, erfordert die Interpretation und Auswertung der Messungen meist ein hohes Maß an Expertenwissen und Erfahrung, um Fehlaussagen zu vermeiden.

Bei der Firma IDS wurde ein Analysesystem für akustische Signale im Audiofrequenzbereich entwickelt. Während ein digitaler Signalprozessor das Spektrum des Meßsignals berechnet, wird die Klassifikation durch einen Fuzzy-Prozessor übernommen.

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Intelligente Sensorik und Aktorik Die Sensorik an sich stellt oftmals eine große Herausforderung dar. Die meisten Sensoren ermitteln eigentlich nicht den exakt vorliegenden Wert, sondern einen anderen, oftmals verfälschten Wert. Durch Interpretation der Meßergebnisse, häufig in Verbindung mit einer Korrelation anderer Daten, versucht man eine Schätzung vorzunehmen. Ein gutes Beispiel hierfür sind die chemischen Sensoren, die die Konzentration eines bestimmten Stoffes erkennen und meßtechnisch erfassen sollen.

Der heute oft beschrittene Weg ist meist recht aufwendig, indem man die analogen Meßdaten verstärkt, mit einem präzisen Analog-Digitalwandler in binäre Form umwandelt, um sie anschließend mit einer mathematischen Funktion über einen Rechner zu verarbeiten, wie etwa im Beispiel der schnellen Fourier-Transformation, oder mittels statistischer Verfahren. Das Ergebnis der Berechnungen wird schließlich einem Komparator zugeführt, der wiederum nur feststellt, ob das Meßergebnis innerhalb einer geforderten Toleranz liegt.

Die Integration von Fuzzy-Systemen zur Auswertung von Meßdaten in Sensoren oder zur vereinfachten Ansteuerung in Aktoren kann zu verringerten Datenübertragungsraten führen und damit die Systemkosten senken.

Anbindung an Standard-Schnittstellen Die heute vielfach gelieferte Schnittstelle in Form eines C-Quellprogrammes erfordert die richtige Parameterübergabe an das Fuzzy-Subsystem und ist für den Software-Entwickler recht transparent. Wir sind oftmals mit potentiellen Anwendern ins Gespräch gekommen, die eine Anbindung an ihr Visual Basic, ihr Kalkulationsprogramm EXCEL oder Einbindung in ihr Prozeßdatenerfassungssystem forderten.

Es gibt viele Schnittstellen in der heutigen Technik, die sich immer weiter entwickeln, oder durch neue ergänzt bzw. ersetzt werden. Man denke z.B. an die Schnittstelle IEC-Bus, die diversen Feldbus-Systeme oder aber auch an die Steckkartensysteme wie VME-Bus, ISA-Bus und SBUS. Für den Fuzzy-unerfahrenen Anwender ist es umso einfacher, diese Technologie mit einzubinden, je besser ihm die Werkzeughersteller mit der Unterstützung seiner Schnittstelle helfen. Natürlich machen nicht alle Schnittstellen in dieser Form Sinn, aber es lassen sich von der Anwenderseite diverse Schnittstellen als Schwerpunkte erkennen.

Integration in Werkzeugumgebung Der Entwickler eines Fuzzy-Systems wird vor der eigentlichen Realisierung sein Modell simulieren. Nach der Validierung seiner Lösung möchte er möglichst sofort ohne große Umwege die bislang erzielten Ergebnisse in die Praxis umsetzen. Bei einigen Werkzeugen steht der Anwender nun vor der Aufgabe, die entsprechenden Interface-Routinen selbst zu programmieren, da der Werkzeughersteller die Schnittstelle zu seiner Anwendung nicht standardmäßig implementiert hat. Findet er allerdings hier eine verbreitete Schnittstelle vor, weil er sich vielleicht bei seinem System für eine standardisierte Lösung entschieden hat, so ist die Wahrscheinlichkeit deutlich höher, vom Werkzeughersteller auch eine geeignete Schnittstelle erhalten zu können.

Integration in Datenbanken Unscharfe Datenbankabfragen können deren Verwendung in vielen Situationen vereinfachen. Man stelle sich einen reisewilligen Kunden vor, der eine Urlaubsunterkunft in Strandnähe, nicht zu weit vom Einkaufszentrum entfernt sucht.

Im Bereich der Regelungstechnik wurde schon in den 70'ger Jahren ein Zementdrehofen in Dänemark mit FUZZY Logik gesteuert. In den 80'ger Jahren wurde die Technik durch die japanischen Ingenieure umgesetzt in die FUZZY Waschmaschine, den FUZZY Camcorder, den FUZZY Staubsauger. Frühzeitig wurde FUZZY Logik im Bereich des Fahrzeugbaus eingesetzt, wie dies u.a. durch eine Getriebesteuerung dokumentiert ist. Bekannt wurde auch die Steuerung der U-Bahn in Sendai, bei der durch die FUZZY Logik ein weicheres Anfahren wie auch ein weicheres Bremsen mit einer einhergehenden Energieersparnis realisiert wurde. Omron als ein Unternehmen mit intensiver FUZZY Logik Forschung integrierte diese Technik in seine SPS Steuerungen. In Deutschland finden sich die ersten Applikationen im Bereich der Hydraulikssteuerung beim KFZ, zur Stabilisierung von Schiffen, einer Enteisungsanlage von Flugzeugen, einer Kransteuerung u.v.m. Es läßt sich absehen, daß durch die FUZZY Logik in dem Segment der Regelungs- und Steuerungstechnik vorhandene Problemlösungen optimiert und zukünftige schneller zum Erfolg führen wird. Hierbei sind adaptive Parameteroptimierungen ein erster Einsatz z.B. bei der Anfahrcharakteristik von Autoklaven.

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller
FG: FUZZY-LOGIK

In der Meßdatenverarbeitung wird sich die FUZZY Logik auch durch den Einsatz von hybriden Hardwarelösungen durchsetzen. Unscharfen Meßdaten werden über einen FUZZY Logik Ansatz mit anderen Daten in Bezug gesetzt und liefern bei verteilten Prozeßstrukturen dem Zentralrechner nur die relevanten Meßwerte. Bei automatischer Kalibrierung in temperatursensiblen Bereichen mit hochgradig nichtlinearen Abhängigkeiten ist so eine Vorverarbeitung ohne Belastung des Hauptrechners zu realisieren.

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Im Bereich der Prozeßsteuerung findet die FUZZY Logik durch ihre Kombination mit den Petri Netzen Zugang zur Steuerung von Fertigungsprozessen wie sie z.B. in der Nahrungsmittelindustrie und in der chemischen Industrie zu finden sind. Unscharf zu bewertende Größen wie sie z.B. ein Speicherinhalt oder eine optimale Chargengröße darstellen, können in eine automatische Prozeßsteuerung integriert werden. Hierbei kommt durch die unscharfe Bewertung das Fachwissen des Anlagenfahrers zum Tragen, welches klassisch nicht eingebunden werden konnte. Durch geeignete Simulation läßt sich nachweisen, daß ähnlich zu der Prozeßsteuerung in der Verkehrsleittechnik durch die FUZZY Logik eine Harmonisierung von Verkehrsströmen herbeiführen läßt. Im Hinblick auf die Umweltbelastungen sowie auch auf die Verkehrssicherheit sind dadurch wirtschaftliche Verbesserungen erzielbar.

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Im Bereich der Klassifikation sind durch die unscharfe Mustererkennung Appliationen z.B. im Bereich der Bildverarbeitung, der Erkennung von Rohrleckagen, der Qualitätssicherung bei Elektromotoren zu finden. Hierbei werden auftretende Signifikanzen einer unscharfen Bewertung unterworfen. Diese Bewertungen werden durch ein das Fachwissen repräsentierendes Regelwerk zueinander in Beziehung gesetzt. Die resultierende Größe ist dann ein Größemaß für den untersuchten Gegenstand. So kann durch eine FUZZY Logik gestützte Bewertung der R-, G- und B- Signale einer Farbkamera eine Farbzuordnung von Gegenständen erfolgen, bei denen eine konstante Farbgebung nicht gewährleistet ist. Mit einem ähnlichen Prinzip ist es möglich, die unterschiedlichen Schadstoffzusammensetzungen von Bodenproben durch eine Spektralbewertung zu bestimmen. In der Qualitätskontrolle bietet die FUZZY Logik die Möglichkeit, das Spektrum von GUT und SCHLECHT durch den Zwischenbereich zu erweitern, wobei bei geeigneter Struktur des Regelwerkes auch Tendenzen z. B. von Maschinenverschleiß frühzeitig erkannt werden.

Im Bereich der Expertensysteme, wie es z. B. ein System zur Prognose des Dollarkursverhaltens ist, erweitert die FUZZY Logik die zur Zeit auf Neuronalen Netzen basierende Expertensoftware. In Kombination mit den Neuronalen Netzen wie auch mit genetischen Algorithmen liefern die Systeme zufriedenstellende Ergebnisse. Erst durch die FUZZY Logik war es möglich, spezielles, nicht in Zahlen darstellbares Expertenwissen, wie es u.a. auch ein
Mediziner besitzt, mit in ein Computerprogramm zu integrieren.

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller

FG: FUZZY-LOGIK

Durch die offensichtliche Affinität zu den Neuronalen Netzen gibt es Entwicklungen, die eine Verbindung beider Techniken derart anstreben, daß Vorwissen über einen zu lernenden Vorgang durch die FUZZY Logik in das Neuronale Netz integriert wird. Durch diese Vorbelegung des Netzes läßt sich die Trainingszeit erheblich reduzieren. Weiterhin ist es möglich, sogenannte lokale Minima durch einen FUZZY Ansatz zu umgehen, um so eine Qualitätsverbesserung des Netzes und des Erkennens zu ermöglichen.

Mit Hilfe von Neuronalen Netzen lassen sich auf der anderen Seite FUZZY Logik Ansätze optimieren. Hierbei wird ein erster FUZZY Logik Ansatz realisiert. Mit den im realen Prozeá erhaltenen Datensätzen ist es möglich, durch ein Neuronales Netz eine neue, optimierte Regelbasis zu generieren. Diese Art des Vorgehens bieten sich bei der Umsetzung von FUZZY Logik in HARD REAL TIME Systemen an, wo Zeitrestriktionen zu beachten sind.

TU Clausthal IfI TI Lehrstuhl für Technische Informatik Prof. Dr.-Ing. D.P.F. Möller
FG: FUZZY-LOGIK

Bedarfsanalyse/Anwendungsgebiete

- Messen, Steuern, Regeln

Unschärfe ist in dynamischen Systemen darin begründet, daß exakte mathematische Beschreibungen entweder gar nicht existieren oder für das Anwendungsgebiet zu kompliziert sind.
Mittels Fuzzy-Logik kann diese Unschärfe modelliert werden. Die Konzepte der Fuzzy-Logik werden im MSR-Anwendungsgebiet angewendet, indem aktuell vorliegende Fakten mit einer Wissensbasis verglichen und daraus gültige Schlüsse gezogen werden. Die Wissensbasis wird in Form von WENN...DANN...-Regeln notiert. Diese gründen sich auf die Fähigkeiten und die Erfahrung des Anwenders, manchmal auch auf einfache Dimensionierungsregeln. Ihre Formulierung ist an die natürliche Sprache angelehnt.

- Entscheidungsfindung

Für eine Vielzahl von Entscheidungsproblemen liegen keine befriedigenden analytisch-numerischen Lösungen vor, da die Quantifizierung der zu berücksichtigenden Kriterien unangemessen erscheint. Mittels Fuzzy-Logik können Ziele und Randbedingungen, wie sie bei Entscheidungsproblemen vorliegen, qualitativ formuliert werden. Hierfür sind Hilfsmittel entwickelt worden, mit den Zusammenhänge zwischen Zielen, Aktionen und Wirkungen qualitativ beschrieben werden können, und die den Entscheider bei der Evaluierung möglicher Alternativen unterstützen.

-Bildverarbeitung/Mustererkennung

Anforderungen steigen hinsichtlich Leistungsfähigkeit und Genauigkeit, werden z.Z. jedoch nicht vollständig erfüllt und sind auch in naher Zukunft durch die klassischen Verfahren und Entwicklungen nicht erfüllbar. So werden z.B. klassische Kompressionsalgorithmen der Notwendigkeit einer qualitativen Auswertung der Bildinformationen nicht gerecht. Fuzzy-Logik kann in diesem Anwendngsgebiet hilfreich angewandt werden, indem sie mehrfach vorkommende Subbilder zu erkennen versucht, oder durch Vergröberungen, die durch das menschliche Auge adaptiv ausgeglichen werden, diese Datenmengen reduzieren. Fuzzy-Logik kann somit auch in der Erkennung von Abstufungen der Helligkeit oder der Strukturierung eingesetzt werden. Rein quantitative Methoden besitzen diese Möglichkeiten in der Regel nicht.

- Datenanalyse/Klassifikation

Bei der Rechnerunterstützten Überwachung und Diagnose muß in der Regel eine Vielzahl von Meßwerten (Daten) beobachtet (erfaßt) und für Klassifikations- und Steuerungsaufgaben bewertet werden. Dabei sind verschiedene Merkmale zu beobachten, deren Ausprägungen in Klassen eingeteilt werden. Bei der Beobachtung bzw. der späteren Analyse von Meßwerten werden zumeist Toleranzbereiche für die einzelnen Merkmale angegeben, wobei aber keine eindeutige Klassenzuordnung erreicht wird. Durch qualitative Formulierung der Toleranzbereiche mittes Fuzzy-Logik (unscharfe Mengen) kann das Anwendungsproblem gelöst werden.

- Optimierung

Optimierung ist ein breites Anwendungsgebiet, z.B. Optimierung von Produktionsprozessen.
Klassische Optimierungsverfahren orientieren sich an der numerischen Bestimmung von Extremwerten einer im Reglfall analytisch beschriebenen Funktion bei gleichzeitiger Einhaltung eines Gütekriteriums (Fehlerfunktional). In der praktischen Anwendung sind qualitative Charakterisierungen wie z.B. geringe Kosten oder hohe Auslastung häufig ausreichend und analytische oder quantitative Aussagen über die Optimierungskriterien häufig nicht möglich. Die Fuzzy-Logik erlaubt es hier die Anwendungsproblemstellungen einfacher zu beschreiben und vorhandene "Toleranzräume" auszuschöpfen.

- Produktionsplanung

In der Produktionsplanung gibt es eine Vielzahl von Einzelproblemen, die je nach Anwendungsgebiet des Produktionsprozesses zu lösen sind, wie z.B. bei der Auftragsvergabe im Falle der Fertigungsleittechnik oder das operative Produktionnsmanagement für unterschiedliche Zeithorizonte und Entscheidungsebenen.
In Abhängigkeit des Anwendungsgebietes kann Fuzzy-Logik, unter Einbindung von Expertenwissen, z.B. in Form von WENN...DANN...-Regeln oder wissensbasierten Komponenten die Lösungsfindung erleichtern. Auch sind Fuzzy-Hybridlösungen unter Einbezug klassischer Verfahren, wie z.B. Petri-Netze bei stark strukturierten Prozesen einsetzbar.

3.1 Fuzzy Control

3. FUZZY-REGELUNGSSYSTEME (FUZZY CONTROL)

Die Regelungstechnik befaßt sich mit Methoden, Ausgangsgrößen und Zustände eines dynamischen Systems auf vorgegebenen Werten zu halten oder einer Solltrajektorie nachzuführen. Daher wird bei "klassischen" Regelungssystemen für deren Synthese die Übertragungsfunktion zwischen den Ein- und Ausgängen des Reglers **quantitativ** mit Hilfe mathematischer Gleichungen beschrieben. Die Idee, Fuzzy-Logik zur **qualitatitven** Beschreibung von Regelungssystemen zu verwenden, wurde zuerst von Zadeh formuliert, die Umsetzung geht auf Mamdami zurück. Mitte der 70er Jahre untersuchte er die Anwendbarkeit unscharfer Verfahren zur Automatisierung komplexer, nichtlinearer Prozesse mit großen Totzeiten, die von erfahrenen Anlagenbetreibern gesteuert wurden. sich aber einer (quantitativen) mathematischen Beschreibung entzogen. Eine der ersten erfolgreichen Realisierungen einer Fuzzy-Logik-Regelung war die Automatisierung eines Zementbrennprozesses mit einem Fuzzy-Regler vom Mamdani-Typ. Seither hat das Gebiet der Fuzzy-Regelungen vielfältige Anwendungen aufzuweisen. Bevor die spezifischen Methoden der Fuzzy-Regelung dargestellt werden, sind die regelungstechnischen Grundlagen in kurzen Zügen zu skizzieren.

3.1 Grundlegende Begriffe und Definitionen der Regelungstechnik

Ein *Regelkreis* besteht, wie aus Bild 3.1 ersichtlich, aus der Reihenschaltung eines *Reglers* und eine *Strecke* in einer *Regelschleife*. Die zu regelnden Systemgrößen werden als *Regelgrößen* bezeichnet, die Sollwertvorgaben als *Führungsgrößen*. Der momentane Wert der Führungsgröße und die aktuelle Regelgröße bilden die *Regeldifferenz*. Der Regler generiert einen *Stelleingriff*, der über Aktoren auf die Regelstrecke einwirkt.

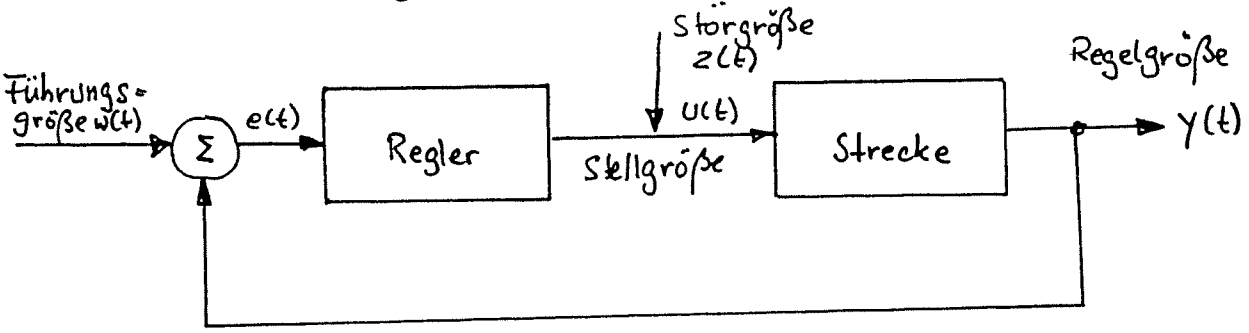


Bild 3.1 Regelkreis

Bei einer *Festwertregelung* muß die Regelgröße möglichst genau auf einen vorgegebenen Führungswert gehalten werden. Dabei müssen Störungen (die prinzipiell an jedem beliebigen Punkt angreifen können) ausgeglichen werden. Sprungförmige Änderungen der Sollwertvorgabe sollen schnell und ohne starke Oszillationen (Einschwingverhalten) ausgeregelt werden.

Bei einer *Folgeregelung* (Servoregelung) wird die Regelgröße dem Verlauf der Führungsgröße kontinuierlich nachgeführt, wobei der Entwurf auf ein geringes Schwingverhalten abzielt.

3.2 Beschreibungsformen linearer Regelungssysteme

3.2.1 Grundlagen

In der Theorie linearer Regelungssysteme kann man auf eine Fülle bewährter Verfahren zur Synthese stabiler und robuster Regelungen zurückgreifen. Voraussetzung für die Anwendung dieser Methoden ist die Verfügbarkeit eines linearen mathematischen Modelles der Regelungsstrecke, dessen dynamisches Verhalten hinreichend genau durch eine lineare Differentialgleichung (DGL) beschrieben wird.

Setzt man die **Linearität** als hervorragendes Systemmerkmal voraus, da muß bei dynamischen Regelungssystemen das **Superpositionsprinzip** gelten, welches besagt, daß die Reaktion eines linearen Systems auf eine Summe von eingangssignalen gleich der Summe der Reaktionen auf die Einzelsignale ist. Daraus folgt:

Eingangsgröße $u_1(t)$ führt zur Ausgangsgröße $y_1(t)$

Eingangsgröße $u_2(t)$ führt zur Ausgangsgröße $y_2(t)$

und damit:

$u(t) = u_1(t) + u_2(t)$ führt zur Ausgangsgröße $y(t) = y_1(t) + y_2(t)$.

Darüber hinaus muß das **Verstärkungsprinzip** erfüllt sein:

Eingangsgröße $u_1(t)$ führt zur Ausgangsgröße $y_1(t)$

und damit:

Eingangsgröße $u(t) = c_1(t)$ führt zur Ausgangsgröße $y(t) = c_1(t)$.

Superpositionsprinzip und Verstärkungsprinzip lassen sich bei linearen Systemen zusammenfassen!

Beschreibungsformen für dynamische Regelungssystems sind DGLn im Zeitbereich z.B. der Form:

$$y^{(n)} + a_{n-1}y^{(n-1)} + \ldots + a_1 y' + a_0 y = b_m u^{(m)} + b_{m-1}u^{(m-1)} + \ldots + b_1 u' + b_0 u$$

mit $m < n$, den rellen Koeffizienten a_i, b_i und $y^{(i)}$ als i-te Ableitung von y(t) nach der Zeit. Die höchste auftretende Ableitung (hier n) wird **Ordnung** der DGL genannt.

Eine weitere Beschreibungsform ist die Modellierung dynamischer linearer Systeme im Laplace- bzw. Frequenzbereich. Dabei geht die lineare DGL durch Laplacetransformation über in die zugehörige **Übertragungsfunktion**:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + s_0}$$

Bislang wurden Beschreibungsformen für dynamische Systeme betrachtet, die daß Eingangs-Ausgangsverhalten des Systems beschreieben. Mit diesen Verfahren kann nur der Verlauf der Ausgangsgröße bei einer bestimmten Eingangsgröße ermittelt werden, jedoch nicht der das im inneren des System ablaufende dynamische Verhalten, welches durch die Zustandsgrößen des Systems beschrieben wird. Hierzu müssen **Zustandsraummodelle** angewandt werden, welche Systemen von DGLn 1. Ordnung enstprechen. Für lineare Systeme haben sie die Form:

$$\underline{X}\cdot = \underline{AX} + \underline{BU}$$

$$y = \underline{c}^T \underline{X}$$

3.2.2 Klassische Reglertypen

In der regelungstechnischen Praxis haben sich drei Kategorien bewährt:

- PID-Regeler (und Unterklassen)
- Kennlinien- bzw. Kennfeldregler
- Zustandsregler

3.2.2.1 PID-Regler

Der PID-Regler besteht aus einem P-Regler, dessen Übertragungsverhalten durch das Reglerfunktional

$$u = K_P e$$

beschrieben wird, d.h. die Stellgröße u ergibt sich aus der Regelabweichung e durch Multiplikation mit der Reglerverstärkung K_p (Proportional-Regler), wobei der Parameter K_p der einzige Freiheitsgrad beim Entwurf ist, sowie einem Integral-Anteiel, was zum PI-Regler führt mit dem Reglerfunktional

$$u = K_P(e + 1/T_N \text{ INT } edt)$$

wobei T_N die Nachstellzeit ist, die eine Gewichtung des Integralterms erlaubt. Dieser sorgt dafür, daß sich auch geringe Regelabweichungen mit zunehmender Zeit immer stärker in der Stellgröße auswirken und die bleibende Regelabweichung verschwindet.

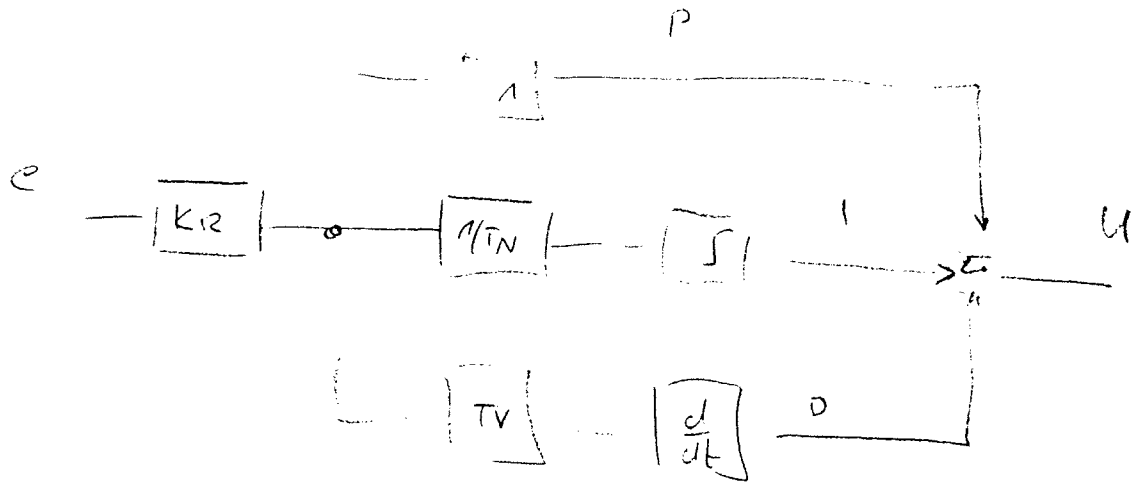
Um die Schnelligkeit des Regelrs zu verbessern wird ein D-Term zugefügt und man hat den PID-Regler in der Form:

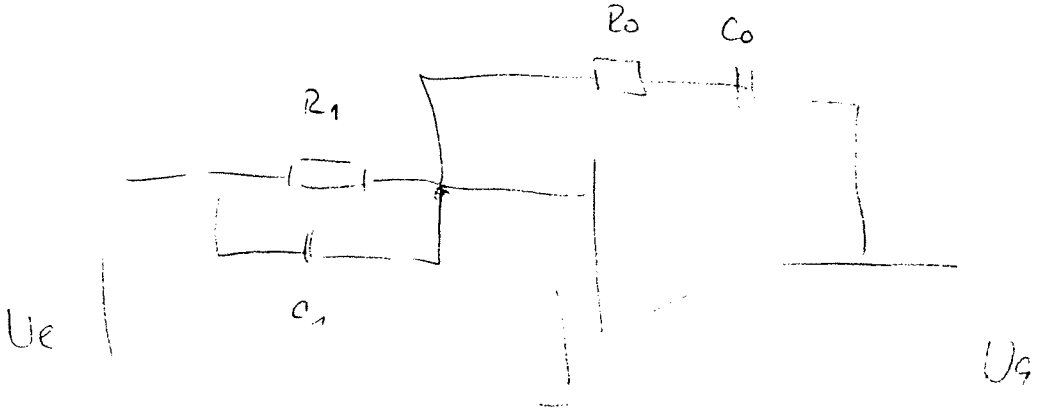
$$u = K_P(e + 1/T_N \text{ INT } edT + T_V e\cdot)$$

wobei T_V die Vorhaltzeit des D-Anteils ist. Dabei bewirkt der D-Anteil, daß sich plötzliche Änderungen der Führungsgröße, die einen unmittelbaren Einfluß auf die Regelabweichung haben, vom Regler schnell erkannt und verarbeitet werden können.

Somit kann zum PID-Regler allgemeingültig festgehalten werden:

- Der P-Anteil sorgt für allgemein günstiges Regelverhalten,
- der I-Anteil sorgt für stationäre Genauigkeit,
- der D-Anteil sorgt für schnelle Ausregelung.





$$u_a = -\left(\frac{R_0}{R_1} + \frac{C_1}{C_0}\right) u_e - \frac{1}{C_0 R_1} \int_0^t u_e \, d\tau - C_1 R_0 \dot{u}_e$$

3.3 Grundstruktur der Fuzzy-Regelungssysteme

Das Blockschaltbild in Bild 3.2. zeigt an einem Beispiel die prinzipielle Struktur eines Reglers der auf **Fuzzy-Control** basiert. Der besseren Verständlichkeit halber wird ein single input/single output fuzzy-controller (SISO) gewählt -andere Fuzzy-Regler wie z.B. MIMO, SIMO, MISO, arbeiten nach dem gleichen Prinzip-. Es gibt nur **eine Stellgröße**, also nur eine vom Regler vorzugebende Eingangsgröße des Prozesses. Die Aktionen des Reglers hängen nur von einer einzigen Eingangsgröße ab, der **Regelabweichung e**.

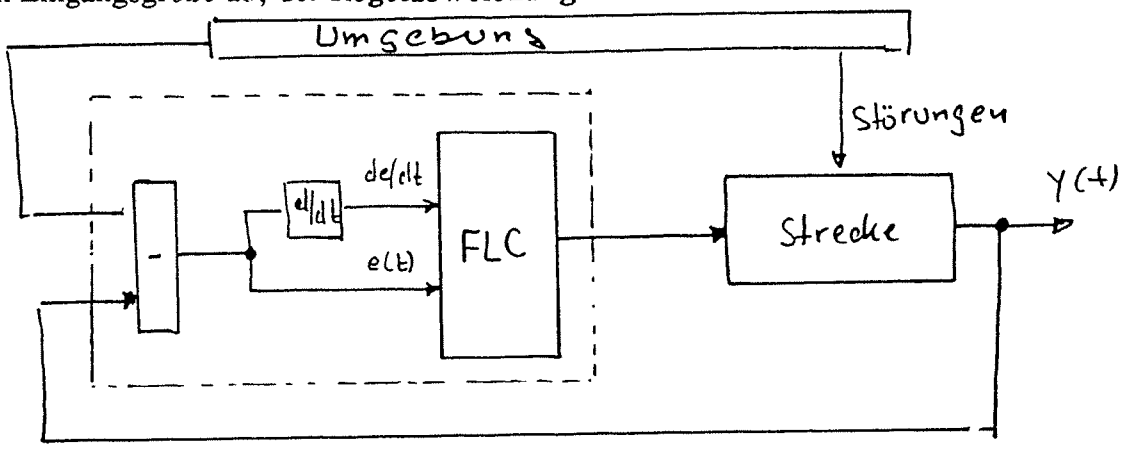


Bild 3.2. Regelung durch fuzzy controller mit e und de/dt als Eingangsgrößen

Der fuzzy controller berechnet seine Ausgangsgröße(n) aus seinen Eingangsgröße(n) anhand linguistischer Regeln. Jede linguistische Regel verknüpft:

- eine Prämisse, formuliert durch eine linguistische Aussage oder eine logische Verknüpfung mehrerer linguistischer Aussagen über die Eingangsggrößen des Reglers,

- mit einer Konklusion, ausgedrückt durch eine linguistische Aussage über die Ausgangsgröße (n) des Reglers.

Ein Fuzzy-Regler kann eine **linguistische Regel** nur dann nutzen, wenn die Aussagen in Prämisse und Konklusion folgender Beschränkung genügen: *Jede Aussage gilt dem momentanen Wert einer Größe - und nicht ihrem zeitlichen Verlauf*. Ein Fuzzy-Regler berechnet grundsätzlich den aktuellen Wert seiner Ausgangsgröße aus den aktuellen Werten seiner Eingangsgrößen arbeitet in Bezug auf seine Eingangsgrößen stets als reiner Proportionalregler.

Um dem Regler eine zeitliche Dynamik zu geben, kann man in den **linguistischen Regeln** die Ausgangsgröße vom Zeitverhalten der Eingangsgröße(n) abhängig machen. Dann ist der Regler so auszulegen, daß der Fuzzy-Regler eine Kenngröße des Zeitverhaltens dieser Eingangsgröße(n) als zusätzliche Eingangsgröße(n) erhält. Die zeitliche Ableitung der Regelabweichung sie wird mit **de/dt** bezeichnet, wird außerhalb des Fuzzy-Reglers gebildet, denn nur dann kann der Fuzzy-Regler eine linguistische Regel auswerten, in deren Prämisse eine Aussage über **de/dt** gemacht wird. Damit kann davon ausgegangen werden, daß die gewünschte zeitliche Dynamik erreicht wird. Für den Fuzzy-Regler sind **e** und **de/dt** damit zwei unabhängige Eingangsgrößen **X** und **Y**. Die Einbettung des Fuzzy-Reglers in seine Umgebung zeigt Bild 3..... Der Regler besteht aus dem eigentlichen Fuzzy-Regler und einem Differenzierer. Ein analoges Vorgehen ist erforderlich, wenn die Regelung einen integralen Anteil haben soll.

3.3.1 Aufbau und Wirkungsweise des Fuzzy Controller

Nachfolgend wird der Aufbau und die Wirkungsweise eines Reglers auf Fuzzy-Logik-Basis, der als Fuzzy Logic Controller **FLC** bezeichnet wird, beschrieben werden. Bild 3.3 zeigt das Blockschaltbild eines FLC mit zwei Eingangs- und einer Ausgangsgröße.

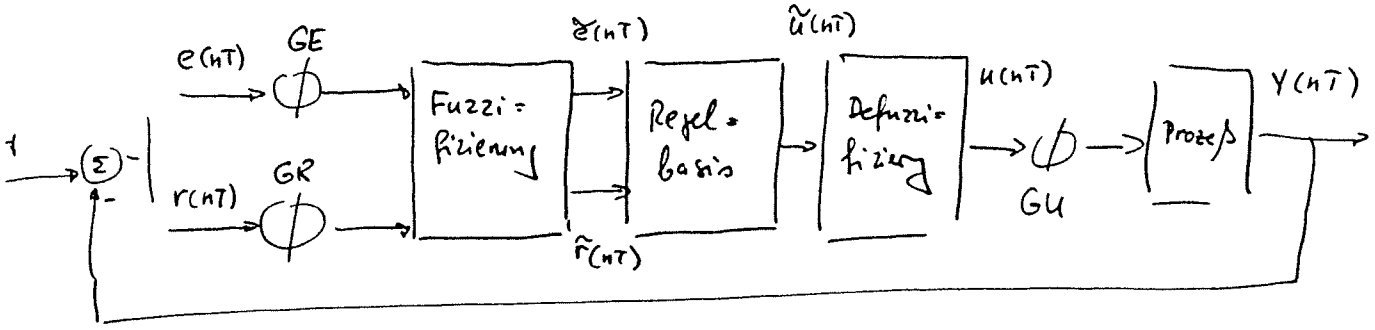


Bild 3.3 Fuzzy Logic Controller FLC

Der FLC besteht im wesentlichen aus drei Komponenten:

- der Fuzzifizierung, d.h. der Einteilung der Ein- und Ausgangsgrößen in unscharfe Mengen
- der Regelbasis mit Inferenzmechanismus, die neben den Verknüpfungsregeln zwischen Ein- und Ausgangsgrößen auch die Entscheidungslogik zur Bestimmung des unscharfen Ausgangswertes beinhalten
- der Defuzzifizierung, d.h. der Umwandlung der unscharfen Mengen für die Ausgangsgröße in einen diese unscharfen Mengen repräsentierenden diskreten Ausgangswert

Aus Bild 3.3 lassen sich somit die folgenden Beziehungen ablesen:

$$e(nT) = y(nT) - \text{Sollwert}$$

$$\tilde{e}(nT) = F[GE * e(nT)]$$

$$r(nT) = [e(nT) - e(nT-T)]/T$$

$$\tilde{r}(nT) = F[GR + r(nT)]$$

$$u(nT) = GU + DF[\tilde{u}(nT)]$$

dabei ist n eine positive natürliche Zahl und T kennzeichnet die Abtastschritte. Die Werte e(nT), r(nT), y(nT) und u(nT) beschreiben den Fehler (error), die Änderung des Fehlers (rate), den Prozeßausgang (**y**) sowie den Fuzzy-Regler-Ausgang (**u**), der gleichzeitig den Prozeßeingang darstellt.
Der Fehler zum Zeitpunkt (nT-T) ist gegeben durch den Term e(nT-T).
GE (gain of error) und GR (gain of rate) bezeichnen die Skalierungsfaktoren für den Fehler bzw. die Änderung des Fehlers und GU (gain of controller output) den Skalierungsfaktor für die Ausgangsgröße des Fuzzy-Reglers.

Mit Hilfe diese Skalierungsfaktoren werden die physikalischen Größen in das Intervall von 0 bis 255 abgebildet, was einer 8 Bit Zahl entspricht-
F[] beschreibt die sog. Fuzzifizierung, d.h. die Einteilung der Eingangsgrößen und der Ausgangsgröße in unscharfe Klassen. Der diskrete Wert des Regleusgangs u(nT) zum Zeitpunkt nT, wird durch die Defuzzifizierung DF[] der unscharfen Mengen der Ausgangsgröße berechnet.

Die Umsetzung dieser drei Komponenten hängt von der Realisierungsform des Fuzzy-Reglers ab. Einerseits können die Schritte Fuzzifizierung und Regelbais/Inferenzmechanismus durch Diskretisierung der linguistischen Variablen in Form einer (i.a. mehrdimensionalen) Relationsmatrix realisiert werden.
Durch Einbeziehung der Defuzzifizierung kann der Regler dann in eine mehrdimensionale Tabelle (eine sog. Look up-Table) überführt werden, aus welcher sich der (scharfe) Stellgrößenwert für eine bestimmte Kombination von (scharfen) Eingangsgrößen unmittelbar oder durch Interpolation von Zwischenwerten entnehmen läßt, ohne daß dazu noch irgendwelche Rechenschritte erforderlich sind. Diese **Off-Line** Realisierungsform ist für spezielle Hardwarestrukturen von besonderem Interesse.

Für die Regelbasis spielt die Wahl der Ein- und Ausgangsgrößen eine besondere Rolle. so daß die Eingangsgrößen dem jeweiligen Anwendungsfall angepaßt werden müssen. Neben der Differenz zwischen Soll- und Istwert (Fehler/error) kann z.B. auch anstelle der Ableitung dieses Fehlers die Ableitung des Istwertes als zweite Eingangsgröße verwendet werden.

Die Komponenten eines Fuzzy-Reglers mit zwei Eingangs- und einer Ausgangsgröße hönnen somit wie folgt zusammengefaßt werden:

- den Skalierungsfaktoren GE, GR und GU für die Ein- bzw. Ausgangsgrößen

- ein Fuzzifizierungsalgorithmus für die skalierten Ein- und Ausgangsgrößen

- die Regelbasis (Fuzzy Control Rules)

- der Fuzzy-Logik zur Berechnung der Inferenz

- der Defuzzifikationsmethode, um aus der unscharfen Menge der Ausgangsgröße einen diskreten Ausgangswert zu berechnen.

Ein großer Vorteil des FLC ist begründet in dem Tatbestand, daß innerhalb dieses Reglers auch Nichtlinearitäten berücksichtigt werden könne, was mit herkömmlichen Reglern nur mit entsprechendem Aufwand möglich ist. Die Implementierungsmöglichkeiten für Nichtlinearitäten sind beim Fuzzy-Regler die folgenden:

- beim verwendeten Fuzzifizierungs-Algorithmus

- dem Fuzzy-Regelwerk

- der Art der Fuzzy-Logik, die für die Berechnung der Inferenz verwendet wird (decision making logic)

- der Defuzzifizierungsmethode, die zur Bestimmung der diskreten Ausgangsgröße herangezogen wird.

Aus der Darstellung wird ersichtlich, daß beim Fuzzy-Regler im Vergleich zu einem konventionellen PID-Regler weitaus mehr Parameter für die Variation zur Verfügung stehen, was zum einen den Vorteil einer guten Anpassungsmöglichkeit des Reglers an spezielle Anfor-derungen beinhaltet, zum anderen jedoch die Gefahr mit sich bringt, daß bei der Reglerauslegung nur eine suboptimale Einstellung erreicht wird. Aus dieser kurzen Darstellung wird deutlich, daß auf dem Gebiet der Fuzzy-Logik nach wie vor ein Bedarf an theoretischer Arbeit vorhanden ist um z.B. Kriterien zur Regelerauslegung bzw. Stabilitätsbetrachtungen entsprechend den konventionellen Reglern durchführen zu können.

Eine weitere Realisierungsvariante für FLC besteht darin, die Stellgröße für die aktuelle Kombination von Eingangsgrößen jeweils **On-Line** zu berechnen. Dazu geht man wie folgt vor:

- Ermittlung des Erfüllungsgrades für die einzelnen Prämissen (WENN-Teile) der Regel
- Verknüpfung der einzelnen Erfüllungsgrade über den UND- bzw. ODER-Operator (z.B. MIN- bzw. MAX-Operator)
- Regeln mit einem Erfüllungsgrad $E > 0$ sind aktiv.
- Ermittlung der zugehörigen Stellgrößen-Fuzzy-Mengen für alle aktiven Regeln
- Ermittlung der resultierenden Stellgrößen-Fuzzy-Mengen durch Überlagerung aller Stellgrößen-Fuzzy-Sets,
- Ermittlung der scharfen Stellgrößen durch Defuzzifizierung.

Diese allgemeine Vorgehensweise kann verdeutlicht werden an folgendem Beispiel:

Für FLC mit zwei Eingangsgrößen $e_1 = 0{,}25$ und $e_2 = -0{,}3$ soll die Stellgröße bestimmt werden. Es sei angemommen, das nur zwei Regeln aktiv sind und zwar:

R_1: WENN e_1 = PK UND e_2 = NU DANN u = PK

R_2: WENN e_1 = N UND e_2 = NK DANN U = N

mit PK = positiv klein
NU = null
NK = negativ klein

Anhand des nachfolgenden Bildes soll die Vorgehensweise schrittweise erläutert werden:

ARGESIM Reports

No.	Title	Authors / Editors	ISBN
# 1	Congress EUROSIM'95 - Late Paper Volume	F. Breitenecker, I. Husinsky	3-901608-01-X
# 2	Congress EUROSIM'95 - Session Software Products and Tools	F. Breitenecker, I. Husinsky	3-901608-02-8
# 3	EUROSIM'95 - Poster Book	F. Breitenecker, I. Husinsky	3-901608-03-6
# 4	Seminar Modellbildung und Simulation - Simulation in der Didaktik	F. Breitenecker, I. Husinsky, M. Salzmann	3-901608-04-4
# 5	Seminar Modellbildung und Simulation - COMETT - Course "Fuzzy Systems and Control"	D. Murray-Smith, D.P.F. Möller, F. Breitenecker	3-901608-05-2
# 6	Seminar Modellbildung und Simulation - COMETT - Course "Object-Oriented Discrete Simulation"	N. Kraus, F. Breitenecker	3-901608-06-0
# 7	EUROSIM Comparison 1 - Solutions and Results	F. Breitenecker, I. Husinsky	3-901608-07-9
# 8	EUROSIM Comparison 2 - Solutions and Results	F. Breitenecker, I. Husinsky	3-901608-08-7