NASA CONTRACTOR
REPORT

NASA CR

MARSHALL SYSTEM FOR AEROSPACE SYSTEM SIMULATION (MARSYAS), USER'S MANUAL

By A. Ventre, R. Sevigny, W. McCollum, and T. Balentine
Computer Sciences Corporation
8300 S. Whitesburg Drive
Huntsville, Alabama 35802

July 1, 1973

Prepared for

NASA-GEORGE C. MARSHALL SPACE FLIGHT CENTER
Marshall Space Flight Center, Alabama 35812

180

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.
NASA CR-124288

2. GOVERNMENT ACCESSION NO.

3. RECIPIENT'S CATALOG NO.

4. TITLE AND SUBTITLE
Marshall System for Aerospace System Simulation (MARSYAS), User's Manual

5. REPORT DATE
July 1, 1973

6. PERFORMING ORGANIZATION CODE

7. AUTHOR(S)
A. Ventre, R. Sevigny, W. McCollum, T. Balentine

8. PERFORMING ORGANIZATION REPORT #
Technical Report #1

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Computer Sciences Corporation
8300 S. Whitesburg Drive
Huntsville, AL 35802

10. WORK UNIT NO.

11. CONTRACT OR GRANT NO.
NAS8-21805

13. TYPE OF REPORT & PERIOD COVERED
Contractor Report

12. SPONSORING AGENCY NAME AND ADDRESS
National Aeronautics and Space Administration
Washington, D. C. 20546

14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES
Work performed for Computation Laboratory, Science and Engineering Directorate

16. ABSTRACT: This document describes the capabilities of the Marshall System for Aerospace System Simulation (MARSYAS) and how to use it.

The Marshall System for Aerospace System Simulation (MARSYAS) is a software system that allows easy setup and control of the simulation of the dynamics of large physical systems on a digital computer. It is particularly suited to the engineer who has little experience in simulation and computer programming. The physical systems are modeled in the form of block diagrams or equations. The blocks can have multiple inputs and multiple outputs, and they can be nested to form hierarchies. The block diagrams can contain transfer functions, nonlinear and logical functions, equations, analog computer elements and FORTRAN programs. The input format of the equations can be combinations of nonlinear, time-varying differential equations and algebraic equations in their original format. MARSYAS could also serve as a storage and retrieval system for models as a basis for a "model configuration control" system on a central time-shared computer. The language allows a standard description of models and easy modification of models stored in a library using descriptive names as in engineering drawings. The outputs of the simulation system can be not only time-responses but also other analysis data such as frequency response, power spectrum and stability parameters.

Several integration modes can override the standard mode. Algebraic loops and discontinuities are identified and solved automatically.

The MARSYAS translator is written in FORTRAN running on the UNIVAC 1108 computer under the EXEC 8 operating system.

17. KEY WORDS
Digital Simulation
Computer Language
Software
Computation Techniques

18. DISTRIBUTION STATEMENT

H. Hoelzer
Director, Computation Laboratory

Unclassified-unlimited

19. SECURITY CLASSIF. (of this report)
U

20. SECURITY CLASSIF. (of this page)
U

21. NO. OF PAGES
177

1.a

For sale by National Technical Information Service, Springfield, Virginia 22151

PREFACE

The Marshall System for Aerospace System Simulation (MARSYAS) has been developed under the direction of Dr. H. Trauboth and has been in use at the Marshall Space Flight Center for more than two years. The software system is written in FORTRAN for the UNIVAC 1108/EXEC 8 computer system and is now available for public use under file number MFS 22 672 at the NASA computer program library

COSMIC
112 Barrow Hall
University of Georgia
Athens, GA 30602

Publications explaining the mathematical foundation of MARSYAS can be found in the Reference.

For further information concerning the material in this manual contact:

Dr. Heinz Trauboth
Chief, Systems Analysis Branch
National Aeronautics and Space Administration
Computation Laboratory
George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama 35812
Telephone (205) 453-1397

or

W. L. McCollum
Project Leader, Senior Computer Scientist
Aerospace Systems Center
Computer Sciences Corporation
8300 S. Whitesburg Drive
Huntsville, Alabama 35802
Telephone (205) 453-2232

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

SECTION I

INTRODUCTION TO MARSYAS

I. INTRODUCTION TO MARSYAS

A. General Description of MARSYAS

MARSYAS (Marshall System for Aerospace Simulation) was developed by NASA's Computation Laboratory at Marshall Space Flight Center to furnish engineers with a software system that allows quick and easy simulation of physical systems on a digital computer.

MARSYAS is a simple, flexible language which can be coded by users who are unfamiliar with computer programming. It is designed for the engineer with little experience in simulation who desires to simulate large physical systems. The language can be used to solve a system of differential equations or to simulate control systems including analog computer block diagrams or both simultaneously. Thus, the user has the ability to mix differential equations with diagrams in his model. The block diagrams can contain, among other things, adders, integrators, transfer functions, multiple input/output nonlinear devices, algebraic equations and nonlinear ordinary differential equations. A block diagram is specified by the user-given names of its models and submodels, inputs and outputs, element names, parameters (if any), and their interconnections. Submodels can be nested to any degree required. With MARSYAS, no preset pattern of connecting elements is required. Elements can be connected in pairs, groups or any manner desired by the user. A large library of Standard Elements and Excitation Functions is part of the MARSYAS system. DEVICE and FUNCTION statement operators allow the user to construct unusual element or excitation functions as needed.

I A. GENERAL DESCRIPTION OF MARSYAS

MARSYAS is a flexible language in that, with few exceptions, there is no rigid statement operator structure within a given module. Most statements can be used without regard for the order in which they appear within the modules. Depending upon the computer system in which MARSYAS is installed, the user has the capability of storing models in a Functional Data Base. The Fortran Object Program generated from the MARSYAS source program can be extracted and run separately, if the user's computing facility can accommodate this feature. When using CHANGE operators, the user has multiple simulation capability without the necessity of either rewriting his model or resubmitting his deck.

An elaborate plotting system is part of the MARSYAS language allowing the user nearly unlimited flexibility in specifying his graphical output. Additionally, the Fast Fourier Transform of any output variable can easily be obtained. A tabular listing of a model in the Functional Data Base or of a model currently being run can be obtained using the LIST operator.

Automatic features of MARSYAS include the detection and solution of linear and nonlinear algebraic loops. For problems which contain discontinuities, the MARSYAS system automatically changes integration schemes to integrate through the discontinuity, unless instructed otherwise by the user.

MARSYAS is designed in modular form so that modifications to the system models can be made with a minimum of effort. In order to achieve comprehensive analysis capability and effective computation, modern control theory is used as the mathematical foundation of MARSYAS. The differential equations generated from block diagrams, or coded as equations, are rearranged internally into vector-matrix state equations which are then solved.

The language is designed so that the user transmits to the computer only the information essential to describe the mathematical model and specify the simulation run.

MARSYAS is divided into four successive modules which describe independent functions of the simulation. These modules are as follows:

Description Module

Modification Module (optional)

Simulation Module

Post Processing Module

The user has the ability to control some of the internal processing of the simulation by specifying his numerical integration method, integration step size or even the truncation error. Normally, he need not concern himself with these details since MARSYAS handles these details automatically.

MARSYAS names can be up to 36 characters in length so that the same names as found in engineering documentation can be used. The MARSYAS alphabet consists of the letters A through Z, the numbers 0 through 9, and the backward slash (\ ). There are no reserved words in MARSYAS.

I B. THE MARSYAS PROGRAMMING SYSTEM

B. The MARSYAS Programming System

The MARSYAS programming system consists of two basic components:

1. A Source Program which consists of a set of MARSYAS statements which may contain a set of FORTRAN subprograms.
2. A Processor Program which pre-compiles the MARSYAS language into a set of FORTRAN programs called the Object Program.

The Source Program is fed to the computer on cards. The MARSYAS coding format is "free form" and information may be punched into any card column, using as many cards as necessary to complete a statement. Statement operators are always followed by a colon (:) and ended by a dollar sign ($). A colon used without a statement operator repeats the previous operator. Imbedded blanks in the coding are ignored. If FORTRAN subprograms are used, the coding must adhere to the rules of FORTRAN.

The MARSYAS Processor Program converts the MARSYAS source language into FORTRAN code. This FORTRAN code, also called the Object Program, is then processed and executed by the computer operating system in the same manner as any other FORTRAN source coding.

The MARSYAS processor program is written in FORTRAN V and, at the present time, can be implemented only on a Univac 1108 computer. Since the processor is modular in design, it can be modified so that the full system can be installed on other large computers.

C. MARSYAS Program Control Statements

There are two program control statements necessary to control the execution of the MARSYAS program. The BEGIN statement identifies the MARSYAS system model being executed and the END statement marks the end of the MARSYAS coding. The use of the BEGIN and END statements is illustrated in Figure 1. The names given to the system model must be identical for the BEGIN and END statements. The general format for both is as follows:

```
BEGIN:  name $

    MARSYAS Program

END:  name $
```

The LIST operator is a third program control statement used to create a tabular listing of a MARSYAS model currently being run or previously stored in a Functional Data Base (if any).

The LIST statement will create a list of the following:

a) The model name.
b) Model input and output terminals with alternate names, if any.
c) Element mnemonics, names and alternate names, if any.
d) Parameter names and associated values.
e) Connections between elements and system input-output terminals.
f) A list of submodels with their inputs and outputs.

The general format for the LIST operator is as follows:

```
LIST:  model name $
```

The LIST statement may be placed anywhere after the BEGIN statement and before the SIMULATE statement, even within a Description or Modification Module.

I D. THE STRUCTURE OF THE MARSYAS LANGUAGE

D. The Structure of the MARSYAS Language

Usually, engineers prefer to describe the system being simulated using block diagrams since this form of "graphical" representation is visually comprehensive. The blocks of the diagram can have multiple inputs and multiple outputs and blocks imbedded within blocks. The lowest level block is called an ELEMENT, the highest, a MODEL. A SUBMODEL is a model imbedded within a MODEL.

A Description Module is used to describe the structure of a model given in block diagram or equation form. It is headed by the operator MODEL and terminated by an END$ statement. The ELEMENTS statement contains the name of the element, its mnemonic, and its parameters.

ELEMENTS are devices which may be linear or nonlinear. A linear element can be as simple as a constant multiplier or as complex as a transfer function. Nonlinear elements are representations of either algebraic equations or switching functions or memory devices such as hysterisis. Frequently-used elements are listed in the Table of Standard Elements at the back of the Manual. If an element needed is not found in the Table of Standard Elements, a FORTRAN subroutine called a DEVICE can be constructed to form the needed element. Parameters are constants written in the format shown in the Table of Standard Elements and are either numerical values or names. The numerical value of a named parameter is given by the PARAMETERIZE Statement. The CONNECT Statement connects strings of inputs and outputs of elements, submodels, system inputs, or system outputs, to form the system block diagram. For

B

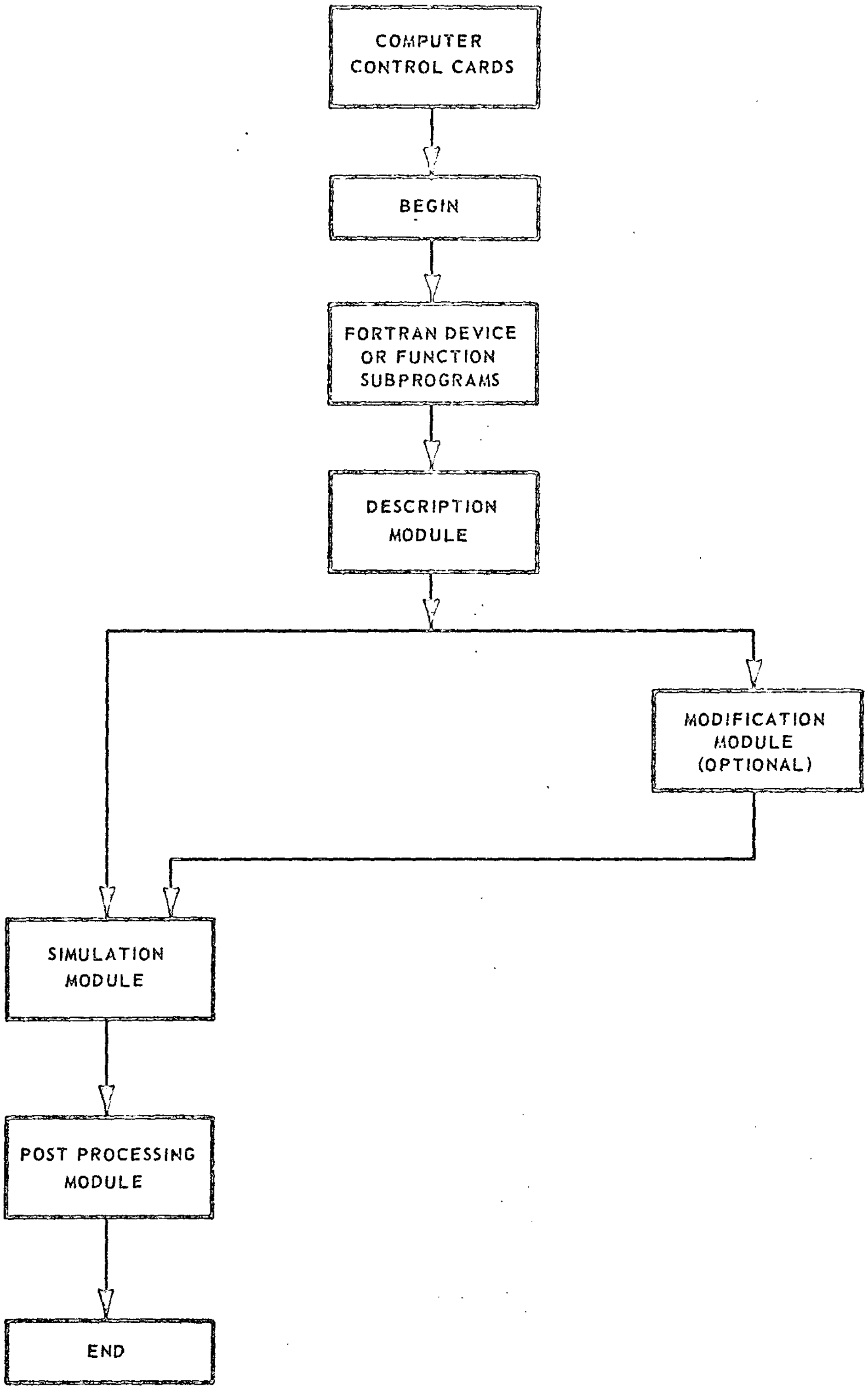


FIGURE 1 SIMULATION OF A MATHEMATICAL MODEL USING MARSYAS

elements or submodels having a single input and output, only the name of the element or submodel appears in the CONNECT Statement. The INPUTS Statement designates names of the inputs of the model; the OUTPUTS Statement designates names of the outputs of the model.

The Modification Module allows inserting, deleting, and disconnecting of elements and submodels previously described in the Description Module through the use of the SUBSTITUTE, DELETE and DISCONNECT Statements.

The Simulation Module completes the specifications of the system being analyzed. The INITIALIZE Statement specifies the initial conditions for the integrators and transfer functions. Excitation functions applied at the system inputs are specified with EXCITE Statements. Frequently used excitation functions are listed in the Table of Standard Excitation Functions at the back of the Manual. If an excitation function needed is not found in this Table, a FORTRAN subprogram called a FUNCTION can be constructed to form the needed excitation. The INTEGRATE Statement specifies the integration method to be used if a method other than the standard method is desired. The STOP IF and TERMINATE IF Statements determine the condition(s) under which the simulation is halted. A CHANGE Statement is used for performing repetitive simulations without the need for dismantling the system model.

The Post Processing Module follows the Simulation Module and specifies the format of data presentation to the MARSYAS processor. Both tabular data and graphs are available for presenting the output data by specifying the PRINT and PLOT statements, respectively. The printing interval is specified in the SAMPLE Statement. A frequency analysis can be obtained at any output terminal through the use of the FOURIER Statement. See Section II for a more detailed description of each of the Modules mentioned in this section.

E. How to Use MARSYAS - Basic Ideas

The complete simulation of a model under the MARSYAS system is composed of several phases or modules as shown in Figure 1. The following brief description of Figure 1 will enable the user to follow the construction of the simple example shown on the following pages. A detailed explanation of the modular structure of MARSYAS can be found in Section II.

The BEGIN statement identifies the user-given MARSYAS program name (Example: BEGIN: TEST RUN1$).

The optional FORTRAN DEVICE and FUNCTION statement box, which appears at the top of Figure 1, is used for placing unusual excite and element descriptions not found in the standard MARSYAS library. In general, the user-coded FORTRAN DEVICE and FUNCTION subprograms may appear anywhere within the MARSYAS program provided they appear before being referenced in the MARSYAS coding. The safest approach, however, is to place such subprograms ahead of the Description Module as shown to avoid difficulty.

The Description Module is used to define the structure of the model. There may be more than one Description Module in a MARSYAS program. It may be modified in a Modification Module. When using the block diagram problem formulation, six distinct statement operators must be specified, in any order with the exception of MODEL and END which must be first and last, respectively, as follows: MODEL name, INPUTS, OUTPUTS, ELEMENTS, CONNECT statements and END. The MODEL name is user-defined and is referenced by the Simulation Module (Example: MODEL: MARSYAS ILLUSTRATION MODEL$). The INPUTS and OUTPUTS statements define all the model inputs and outputs. If the model has one input and one output whose user-assigned names are U1 and Y1, respectively, the statements would read as:

```
INPUTS:  U1$
OUTPUTS:  Y1$
```

The ELEMENTS statement(s) specifies all of the elements contained within the model, such as adders, integrators, transfer functions, etc.

A complete list of MARSYAS elements appears in the Table of Standard Elements, along with their mnemonics. For example, the mnemonic for an integrator is IN and the mnemonic for an adder is AD. If the model contains one adder and two integrators, the ELEMENTS statement would read:

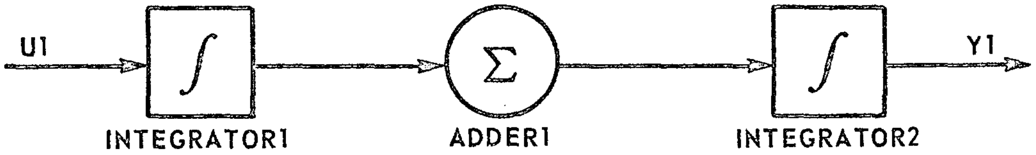
```
ELEMENTS:  AD, ADDER1$  : IN, INTEGRATOR1, INTEGRATOR2$
```

The CONNECT statement(s) connects the elements, inputs and outputs of the model to form a complete circuit. Using the inputs, outputs, and elements referred to above, the CONNECT statement could read:

```
CONNECT:  U1, INTEGRATOR1, ADDER1, INTEGRATOR2, Y1$
```

IE. HOW TO USE MARSYAS - BASIC IDEAS

The CONNECT statement is easy to visualize in the following schematic:



For the simple model just shown, the complete Description module is as follows:

```
MODEL:  MARSYAS ILLUSTRATION MODEL$

INPUTS:  U1$

OUTPUTS:  Y1$

ELEMENTS:  AD, ADDER1$
        :  IN, INTEGRATOR1, INTEGRATOR2$

CONNECT:  U1, INTEGRATOR1, ADDER1, INTEGRATOR2, Y1$

END$
```

The END statement is the last statement to appear in the Description Module. Every Module shown in Figure 1 must terminate with an END statement.

The Simulation Module follows the Description Module. This module begins with the SIMULATE statement. It references the main model name being simulated as:

```
SIMULATE:  MARSYAS ILLUSTRATION MODEL$
```

I E. HOW TO USE MARSYAS - BASIC IDEAS

SIMULATE must be the first and END the last statements in the Simulation Module. The remaining statements described below may be located anywhere within the Module, without regard to order. The Simulation Module contains the following statements: EXCITE, INTEGRATE (optional), TERMINATE IF or STOP IF, INITIALIZE, CHANGE (optional) and END.

The EXCITE statement is used to specify an excitation function on a particular input. Usually, but not always, a Standard MARSYAS excitation function is used. If an excitation function not listed in the Table of Standard Excitation Functions is needed, the user may construct a FORTRAN subprogram to do the job. This subprogram is called a FUNCTION and is placed before the Description Module as previously discussed.

A typical EXCITE statement might read:

```
EXCITE: U1, FSIN (2.0, 3.0, 4.0)$
```

where FSIN is the MARSYAS mnemonic for a sinusoid. The above EXCITE statement says, "Excite input terminal U1 with 2 sin (3t + 4)."

The INTEGRATE statement specifies the mode of integration to be used in solving the model. If the INTEGRATE statement is omitted, the problem will be solved using the Sarafyan variable-step method. If fourth-order Runga-Kutta is desired, the INTEGRATE statement could read:

```
INTEGRATE: RK, TIMESTEP, 0.01$
```

where RK is an abbreviation for Runga-Kutta and TIMESTEP, 0.01 specifies the integration interval. If the word TIMESTEP is omitted, a step size of 0.01 is used automatically.

I E. HOW TO USE MARSYAS - BASIC IDEAS

It is usually necessary to terminate the problem solution when a certain condition has been reached in one of the model parameters or when a certain point in time is reached. If, for example, it is desired to stop the solution after ten time units, one of the following statements could be used:

```
          TERMINATE IF:  TIME .GT. 10.0$
     or
          STOP IF:  TIME .GT. 10.0$
```

The INITIALIZE statement is used to impose initial conditions, other than zero, on integrators and transfer functions. If, for example, an initial condition of 3.2 were required on INTEGRATOR1 in the above example, the statement would read

```
INITIALIZE:  MARSYAS ILLUSTRATION MODEL, INTEGRATOR1 (3.2)$
```

When using the INITIALIZE statement, the model name as well as the element name must be specified. If there are no initial conditions associated with the problem, the INITIALIZE statement is omitted.

When it is necessary to make a succession of runs with different parameter elements, a CHANGE statement may be used. CHANGE statements are discussed in detail in Section II C.

The END statement is the last statement to appear in the Simulation Module. Summarizing the statements discussed above, the Simulation Module for the simple example is as follows:

```
SIMULATE:  MARSYAS ILLUSTRATION MODEL$
EXCITE:  U1, FSIN (2.0, 3.0, 4.0)$
INTEGRATE:  RK, TIMESTEP, 0.01$
TERMINATE IF:  TIME .GT. 10.0$
INITIALIZE:  MARSYAS ILLUSTRATION MODEL, INTEGRATOR1 (3.2)$
END$
```

The POST PROCESSING Module follows the Simulation Module and is used to specify which outputs to record, the time interval and the method of presentation for viewing, that is, tables and/or graphs. In addition, a frequency analysis can be specified at any output using the FOURIER statement. The following statements are used in the POST PROCESSING Module: PRINT, SAMPLE, PLOT, FOURIER, END. An example of the use of these statements is as follows:

```
PRINT:  U1, Y1$
SAMPLE:  STEP, 10$
PLOT:  LINEAR (0.0, 10.0, 1), U1, Y1$
END$
```

I E. HOW TO USE MARSYAS - BASIC IDEAS

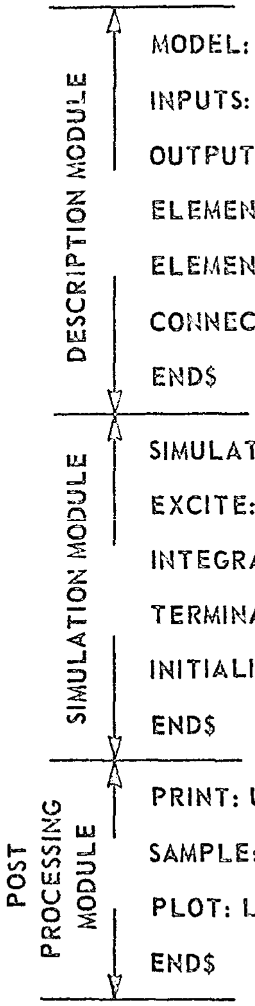
As a consequence of the above statements, the MARSYAS processor will tabulate Time, U1, Y1 at every tenth integration step (STEP, 10) and plot on separate sheets of linear graph paper (SC 4020 plotter) U1 and Y1 versus time in the time interval 0-10. As usual, the END statement is the last to appear in the Module.

The very last statement to appear in the MARSYAS coding is a final END name statement as:

```
END:  TEST RUN1$
```

This is the counterpart of the BEGIN statement which is the first statement to appear in the MARSYAS deck. Its purpose is to signal the MARSYAS processor that there is no more MARSYAS coding.

Combining all of the above MARSYAS statements as shown in Figure 2 will emphasize the basic structure of the language and will enable the user to follow the solution of the sample problem on the following pages. A detailed description of all of the statement operators discussed in this section will be found in Section II of the Manual.

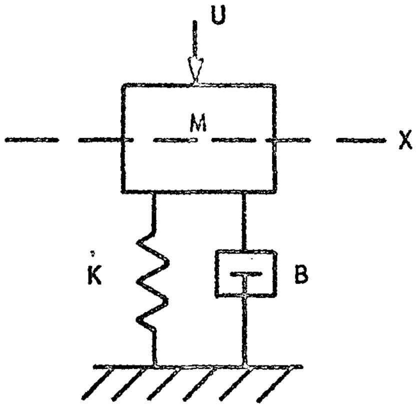


```
BEGIN: TEST RUN1$
MODEL: MARSYAS ILLUSTRATION MODEL$
INPUTS: U1$
OUTPUTS: Y1$
ELEMENTS: AD, ADDER1$
ELEMENTS: IN, INTEGRATOR1, INTEGRATOR2$
CONNECT: U1, INTEGRATOR1, ADDER1, INTEGRATOR2, Y1$
END$
SIMULATE: MARSYAS ILLUSTRATION MODEL$
EXCITE: U1, FSIN (2.0, 3.0, 4.0)$
INTEGRATE: RK, TIMESTEP, 0.01$
TERMINATE IF: TIME .GT. 10.0$
INITIALIZE: MARSYAS ILLUSTRATION MODEL, INTEGRATOR1 (3.2)$
END$
PRINT: U1, Y1$
SAMPLE: STEP, 10$
PLOT: LINEAR (0.0, 10.0, 1), U1, Y1$
END$
END: TEST RUN1$
```

FIGURE 2. ILLUSTRATION OF MARSYAS CODING

F. A Sample Problem

Consider the simple mass-spring-dashpot system shown in the illustration below.



The motion of the mass is described by the following differential equation and initial conditions:

$$M\ddot{x} + B\dot{x} + Kx = U \qquad (1)$$

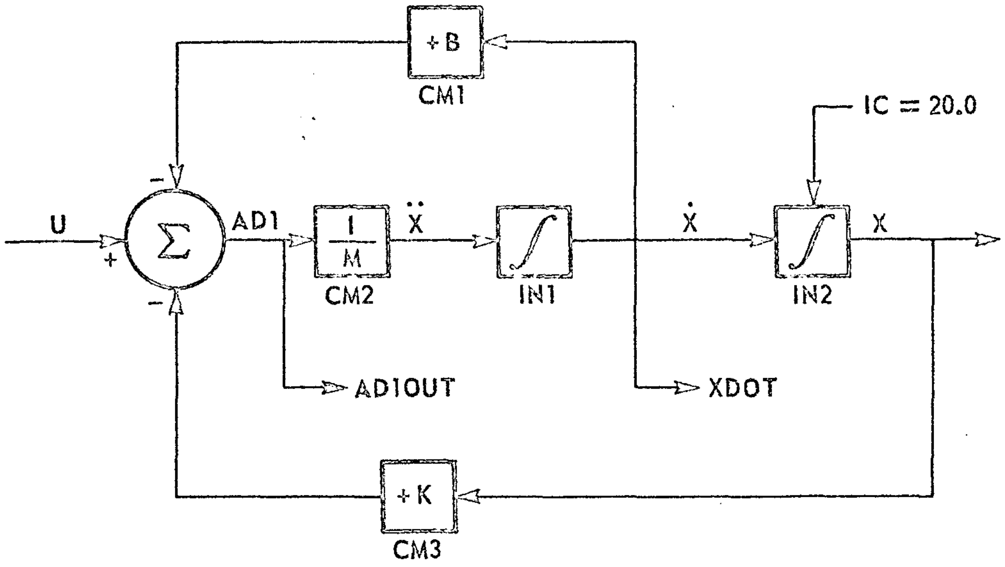
$$x(0) = 20.0 \quad \text{and} \quad \dot{x}(0) = 0 \qquad (2)$$

where $M = 10.0$, $B = 2.5$, $K = 8.6$, and $U = \sin t$

There are two methods of solving this problem using MARSYAS: (i) Equation (1) can be transformed into a block diagram whose elements will consist of two integrators, an adder, and three constant multipliers or, (ii) the equation can be coded directly as shown above. The two methods of coding are illustrated in Figures 3 and 4.

The EQUATION option appears to be the more straight-forward method of solution for this particular problem in that the Description Module requires less coding. However, larger systems would probably be solved using the block diagram approach since more insight into the structure of the problem is thereby gained.

IF. A SAMPLE PROBLEM



```
BEGIN: MARSYAS EXAMPLE$
```

DESCRIPTION MODULE

```
MODEL: MASS SPRING DAMPER$

INPUTS: U$

OUTPUTS:AD1OUT, XDOT, X$

ELEMENTS: IN, IN1, IN2$ : AD, AD1$ : CM, CM1(2.5), CM2(0.10), CM3(8.6)$

CONNECT: AD1, AD1OUT$ : U, AD1, CM2, IN1, IN2, CM3, -AD1$ : IN1, XDOT$
           : IN1, CM1, -AD1$  : IN2, X$

END$
```

SIMULATION MODULE

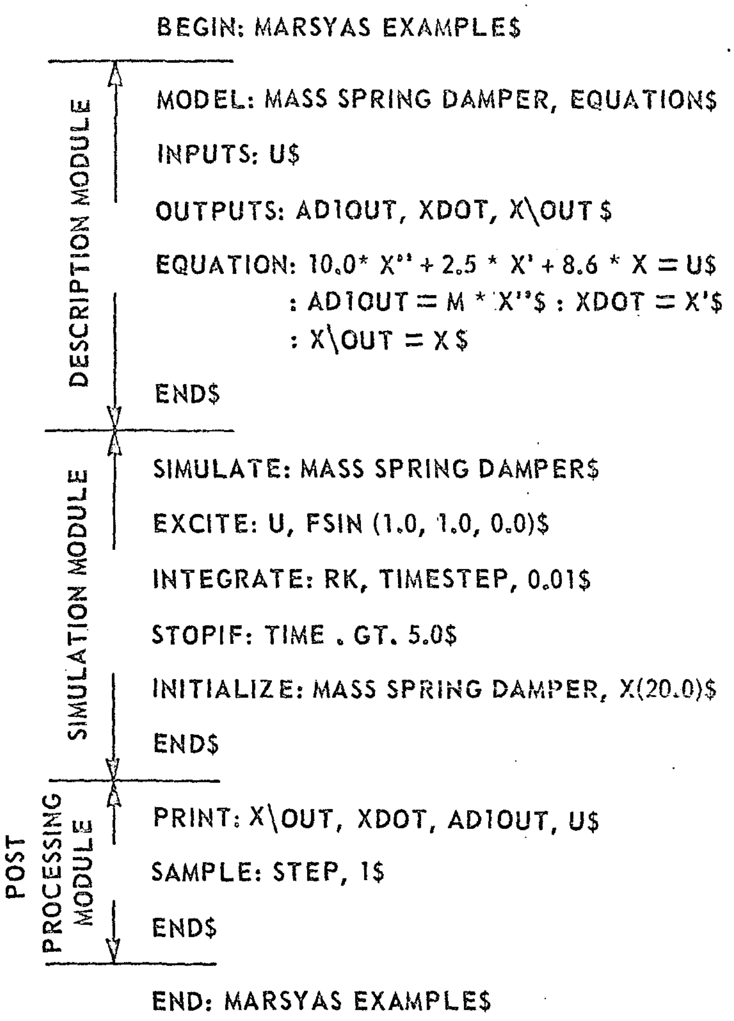
```
SIMULATE: MASS SPRING DAMPER$

EXCITE: U, FSIN (1.0, 1.0, 0.0)$

INTEGRATE: RK, TIMESTEP, 0.01$

STOPIF:  TIME .GT. 5.0$

INITIALIZE: MASS SPRING DAMPER, IN2(20.0)$

END$
```

POST PROCESSING MODULE

```
PRINT: X, XDOT, AD1OUT, U$

SAMPLE: STEP, 1$

END$

END: MARSYAS EXAMPLE$
```

FIGURE 3. BLOCK DIAGRAM APPROACH TO SOLVING A SECOND ORDER PROBLEM USING MARSYAS

IF. A SAMPLE PROBLEM



```
BEGIN: MARSYAS EXAMPLE$

MODEL: MASS SPRING DAMPER, EQUATION$

INPUTS: U$

OUTPUTS: AD1OUT, XDOT, X\OUT $

EQUATION: 10.0* X'' + 2.5 * X' + 8.6 * X = U$
          : AD1OUT = M * X''$ : XDOT = X'$
          : X\OUT = X $

END$

SIMULATE: MASS SPRING DAMPER$

EXCITE: U, FSIN (1.0, 1.0, 0.0)$

INTEGRATE: RK, TIMESTEP, 0.01$

STOPIF: TIME . GT. 5.0$

INITIALIZE: MASS SPRING DAMPER, X(20.0)$

END$

PRINT: X\OUT, XDOT, AD1OUT, U$

SAMPLE: STEP, 1$

END$

END: MARSYAS EXAMPLE$
```

FIGURE 4. EQUATION APPROACH TO SOLVING A SECOND ORDER PROBLEM USING MARSYAS

I F. A SAMPLE PROBLEM

The program control statements BEGIN name and END name must appear regardless of which method of coding is selected. The name MARSYAS EXAMPLE is selected to identify this MARSYAS simulation.

The following explanation of the coding which appears in Figures 3 and 4 should enable the user to become familiar with the MARSYAS language. A complete description of each of the Modules and Statement operators appears in Section II of this Manual.

Description Module

The name MASS SPRING DAMPER identifies the MODEL whose structure is being described in the statements to follow. If SUBMODELS were referenced in this MODEL, then additional Description Modules would be required to describe the structure of each submodel. Notice that in Figure 4 the word EQUATION follows the model name indicating that equations are to be used in the MODEL. The excitation U is an input to the system and is so stated in the INPUTS statement. Since we wish to observe the behavior of the system being excited, the names X, XDOT and AD1OUT have been specified as outputs. Generally, outputs can be at any point in the system so long as they are specified in the OUTPUTS statement.

When the problem is simulated in block diagram form, all of the elements and connections, as well as the inputs and outputs, must be specified in a Description Module as shown in Figure 3. The constants B, M and K are constant multiplier elements with names CM1, CM2 and CM3, respectively. The numerical value of each constant multiplier appears in parenthesis after its name in accordance with the format shown in the Table of Standard Elements. The two integrators in the block diagram are elements with names IN1 and IN2, so that the total number of elements in the block diagram is six. All of the information concerning these elements is contained within the ELEMENTS statement shown in Figure 3. When elements are present, they must be connected together to form the block diagram. This is done using a CONNECT statement. One important rule that must not be violated when making connections is that redundant connections are to be avoided. The same path must not be retraced. Since there is great flexibility in constructing CONNECT statements, the user should always strive to keep it simple for "bookkeeping" purposes. The first CONNECT statement in Figure 3 connects the output of adder AD1 to an output terminal AD1OUT. Note that the dollar sign ($) terminates a string of connections. The next CONNECT statement (denoted by the colon) traces the feed forward path and the lower feedback path, all in one statement. This is done strictly for the sake of convenience and speed of writing. This connection could have been effected using many separate CONNECT statements, if desired. The next CONNECT statement connects the output of integrator IN1 with output terminal XDOT. The next statement traces the upper feedback path and the last statement connects integrator IN2 with output terminal X. CONNECT statements are easy to construct and should present no difficulty for the user. More information on CONNECT statements can be found in Section II of this Manual.

I F. A SAMPLE PROBLEM

When the problem is simulated in EQUATION form, ELEMENTS and CONNECT statements are unnecessary. All that is required is the EQUATION operator as shown in Figure 4. Each equation is terminated by a ($) sign. The colon repeats the EQUATION operator for each equation.

The END$ statement terminates the Description Module.

Simulation Module

The SIMULATE statement name MASS SPRING DAMPER references the model being simulated. The name is the same as the main MODEL name given in the Description Module. The excitation function, sin t, is impressed upon the input terminal U using the EXCITE statement whose format is given in the Table of Standard Excitation Functions. The integration method selected to solve this problem is Runga-Kutta with a step size of 0.01. This information is recorded in the INTEGRATE statement. The STOP IF statement terminates the simulation after five time units. The initial condition X(0) = 20.0 is specified in the INITIALIZE statement. In Figure 3, the initial condition is placed on the integrator whereas in Figure 4, X is initialized directly.

The END$ statement terminates the Simulation Module.

Post Processing Module

Since we wish to examine the system input and outputs, the PRINT statement contains all of the names listed in the INPUTS and OUTPUTS statement which are to be printed out. TIME will automatically be listed in the first column of each page of printout. The printing will occur at each integration STEP as indicated in the SAMPLE statement. If the SAMPLE statement is omitted, printing will occur at each integration step.

The END$ statement terminates the Post Processing Module.

I G. MARSYAS MATH REFERENCE

G. MARSYAS MATH REFERENCE

When using the EQUATION option in the Description Module, mathematical expressions may arise which allude to a quantity called a MATH REFERENCE. The MARSYAS MATH REFERENCE library consists of the Standard Elements, user-defined elements (DEVICES) and Standard Excitation Functions. The form which these MATH REFERENCES take may be quite complex. since they may consist of mathematical expressions, mathematical operations or other MATH REFERENCES.

Suppose an equation contains a forcing function f(x,y,t) which can be represented by a Standard Excitation Function. The general form of the Standard Excitation Functions, as shown in the Table of Standard Excitation Functions at the back of the Manual is as follows:

excitation function mnemonic (parameters) $

For use in an EQUATION as a MATH REFERENCE, this form must be altered slightly to include TIME or an equivalent expression for time as

excitation function mnemonic (parameters, A)$

where A is either the word TIME or a mathematical expression from which time can be computed. Usually, however, the user will simply insert the word TIME in place of A. (See EXAMPLE (2) in EQUATION, Part II A.) The user then associates each of the parameters required by the Standard Excitation Function with the variables in f(x,y,t) as shown in EXAMPLE (1).

I G. MARSYAS MATH REFERENCE

EXAMPLE (1)

Suppose an equation contains a forcing function $5X \sin(2X^2 + 3Yt)$. Using the Table of Standard Excitation Functions at the back of the Manual, the MATH REFERENCE is coded as follows:

```
FSIN(5. * X, 3. * Y, 2.0 * X ** 2, TIME)
```

where d_1 = 5. * X, d_2 = 3. * Y, and d_3 = 2. * X ** 2.

Next, suppose that an EQUATION contains a term which is to be represented by a Standard Element. This situation might arise when using EQUATION in conjunction with a block diagram. In this case, the MATH REFERENCE is represented in much the same way as with the Standard Excitation Function except that we now use Standard Elements in place of Standard Excitation Functions. The general form of the Standard Elements as shown in the Table of Standard Elements at the back of the Manual is as follows:

element mnemonic (parameters)$

For use as a MATH REFERENCE, this form must be altered slightly to include a mathematical expression(s) for each input of the element, as well as an integer indicating to which element output terminal the EQUATION applies, as

element mnemonic (parameters, M_1, M_2, ..., N)

where M_i is a mathematical expression for each element input terminal which may involve derivatives, other mathematical expressions or other

I G. MARSYAS MATH REFERENCE

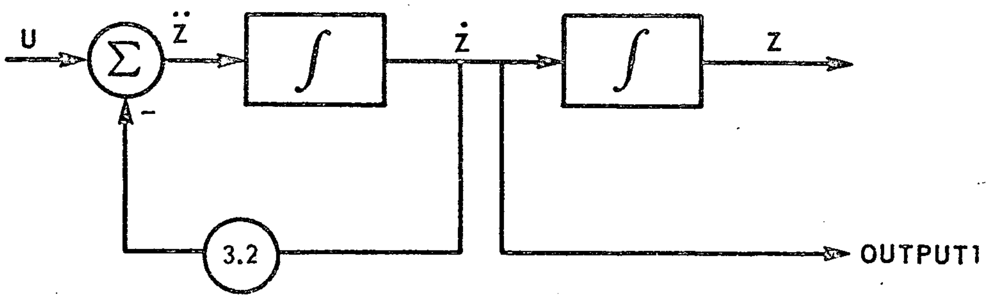
MATH REFERENCES. N is the integer indicating to which output terminal the EQUATION applies.

If the element has only one output, the integer N may be omitted. The user assigns numerical values to the indicated parameters (if any) as required in the usual use of Standard Elements. See EXAMPLE (2).

Note in EXAMPLE (2) that, since there are no parameters associated with an integrator element and only one output, both the parameters and the integer N are omitted from the parenthesis. Additionally, only one mathematical expression is needed since the integrator has only one input. Notice that either method of coding OUTPUT1 is legitimate since both are equivalent mathematically.

EXAMPLE (2)

Suppose a block diagram contains two integrators and one constant multiplier as shown in the sketch below. Using the mnemonic for an integrator, IN, as given in the Table of Standard Elements at the back of the Manual, an equation for OUTPUT1 using a MATH REFERENCE is coded as shown below.



I G. MARSYAS MATH REFERENCE

```
EQUATION:  Z"  =  U - 3.2 * Z' $
        :  OUTPUT1  =  IN(Z") $
```

or

```
EQUATION:  OUTPUT1  =  IN(U - 3.2 * Z') $
```

SECTION II

MODULAR STRUCTURE OF MARSYAS

II. MODULAR STRUCTURE OF MARSYAS

A. Description Module

The function of the Description Module is to provide the MARSYAS processor with a description of the structure of the system being simulated. The system might be a mathematical representation of a complete Space Shuttle, its engines or any one of its subsystems. The mathematical representation could be in the form of a block diagram, a series of block diagrams, a set of differential equations or a mixture of block diagrams and equations.

A Description Module is a MARSYAS representation of a complete or partially complete mathematical system. A MARSYAS program may contain several Description Modules. A model is completely described in a Description Module. A submodel is a model wholly contained within another model. If a model contains one or more submodels, then each submodel's structure must be described in a separate Description Module.

When a model is represented in block diagram form, its structure is specified by its INPUTS, OUTPUTS, ELEMENTS, SUBMODELS and connections. An input is any point in a model where a driving function is applied and an output is any point where the behavior of the model is to be examined. An element is any device whose input(s) and output(s) can be related by algebraic or differential equations or based on logic. Each element type, such as an adder or integrator, has a mnemonic, viz., AD and IN, respectively. A library of Standard Elements is part of the MARSYAS system and is given in the Table of Standard Elements at the back of the Manual. A FORTRAN subprogram, called a DEVICE, can be constructed by the user to simulate any element not found in the Table of Standard Elements. Some elements contain parameters which can be numerical constants or named constants. If the parameter constants are named, their values are specified with PARAMETERIZE statements. The interconnections among the system inputs, elements, submodels and outputs are specified with CONNECT statements. Connect statements can be modified without altering the original coding by using DISCONNECT statements. Alternate names for models, elements, inputs, outputs or parameters can be assigned using the NAME statement. There are no reserved names (words) within the MARSYAS system.

When a model is represented by a collection of differential equations, the statement operators ELEMENTS, PARAMETERIZE, SUBMODEL, CONNECT, DISCONNECT and NAME are not used. The EQUATION operator is used when specifying equations. Equations may contain mathematical expressions which consist of MARSYAS names, mathematical operators, numerical constants and MATH REFERENCES. MATH REFERENCES are discussed in Section I G.

The order in which the above statements appear within a model description is immaterial except that MODEL and END must be the first and last statements, respectively. A detailed description of all of the statement operators used in the Description Module appears in alphabetical order on the following pages. Figure 5 is a list of these operators with a brief description of their function.

II A. DESCRIPTION MODULE

OPERATOR	FUNCTION
CONNECT	Specifies the interconnection of elements, input and output terminals and submodels within the model.
DEVICE	Defines a non-standard element.
DISCONNECT	Cancels connections specified by a previous CONNECT statement.
ELEMENTS	Specifies the mnemonic, names and parameters of elements in a model.
END	Marks the end of a Description Module.
EQUATION	Specifies a model in terms of equations.
INPUTS	Assigns names to the model input terminals.
MODEL	Marks the appearance of a Description Module and names the model.
NAME	Assigns alternate names to units already defined.
OUTPUTS	Assigns names to the model output terminals.
PARAMETERIZE	Assigns a numerical value to a parameter which was given a name instead of a number in an ELEMENTS statement. Also used to change the value of a parameter assigned a numerical value by an ELEMENTS or PARAMETERIZE statement.
SUBMODEL	Assigns a name to a submodel and its associated inputs and outputs.
*	Allows comments for the user.

FIGURE 5. STATEMENT OPERATORS USED IN THE DESCRIPTION MODULE

COMMENT

FORMAT -

```
*    message or blanks    $
```

COMMENTARY -

The comment statement may appear anywhere in any module and may extend over as many source cards as are necessary. The source cards are printed exactly as read without the editing common to all other MARSYAS statements. The asterisk can appear in any card column.

EXAMPLE

```
*    This is a comment card    $
```

CONNECT

FORMAT -

CONNECT: name1, name2, name3, etc. $

COMMENTARY -

The CONNECT statement is used to describe the signal paths among the elements, submodels and input and output terminals of the system. The direction of the signal flow is implied by the order in which the arguments of the connection statement are given. CONNECT statements must always <u>begin</u> as follows:

1. at an element output

or

2. at a system input terminal

CONNECT statements must always <u>end</u> as follows:

1. at an element input

or

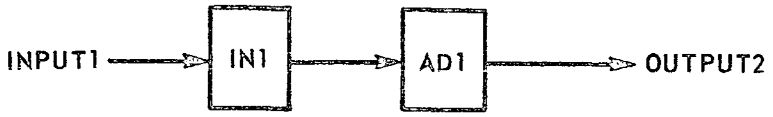
2. at a system output terminal

Since a system input terminal is used to apply a signal, it must be the first to occur in the CONNECT statement of which it is a part. Similarly, since a system output name can only receive a signal, it can only appear as the last name in a string of CONNECT statements. Redundant connection paths are not allowed.

If a signal path goes through a submodel, it is only necessary to specify the connections to and from the submodel. It is not necessary to repeat the connections within the submodel which have previously been described in that submodel's Description Module.

Every signal path which occurs within the model must be described by a CONNECT statement. If a signal path branches, then the new path is started at the point of branching. The CONNECT statement is not used with the EQUATION statement.

EXAMPLE

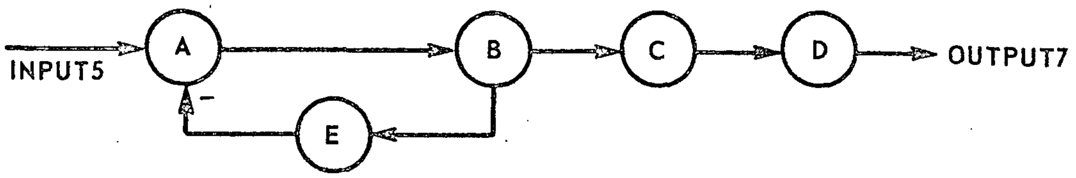


The CONNECT statement for the above simple circuit is as follows:

CONNECT: INPUT1, IN1, AD1, OUTPUT2$

EXAMPLE

Consider the following simple circuit:



The CONNECT statement reads as follows:

```
CONNECT:  INPUT5, A, B, C, D, OUTPUT7$
       :  B, E, -A$
```

There are several permissible variations of this CONNECT statement such as

```
CONNECT:  INPUT5, A, B, E, -A
       :  B, C, D, OUTPUT7$
```

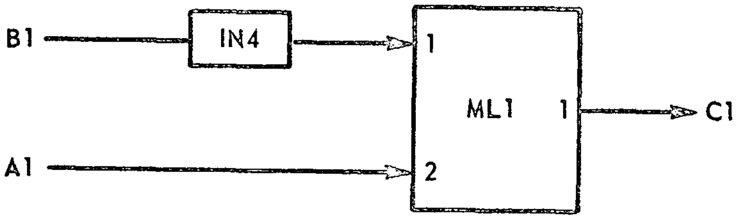
The intermediate dollar sign(s) may be omitted when using the colon to repeat a statement operator as above.

Note that the CONNECT statement allows the use of a negative sign on element inputs only. For example, -INPUT5 would not be permitted in the above CONNECT statements.

When elements with multiple inputs and outputs are connected, the element terminal numbers must be carefully noted when writing the CONNECT statement.

EXAMPLE

Consider the following connections of system INPUT5, A1 and B1 with ELEMENTS, ML1 and IN4 and the connection of the output of element ML1 with the system output, C1.



The CONNECT statement could be coded as follows:

```
                     _ _ _ _ _ _ _
CONNECT:  B1, IN4,  | 1#ML1, ML1#1 | ,   C1$
                    |_ _ _ _ _ _ _ |

          :  A1, 2#ML1$
```

The two terms in the dotted box may be combined into one for ease of writing. Thus 1#ML1#1 is equivalent to writing 1#ML1, ML1#1.

The above connect statement reads as follows: "Connect input B1 to element IN4 and element IN4 to input #1 of element ML1 and connect output #1 of element ML1 to output C1. Connect input A1 to input #2 of element ML1." Note that the second CONNECT statement, CONNECT: A1, 2#ML1$, does not connect past input #2 of ML1, thus avoiding a redundant connection.

Should the user desire, he can give names to the terminals of an element using a NAME statement, then CONNECT using element names instead of terminal numbers (see NAME).

EXAMPLE

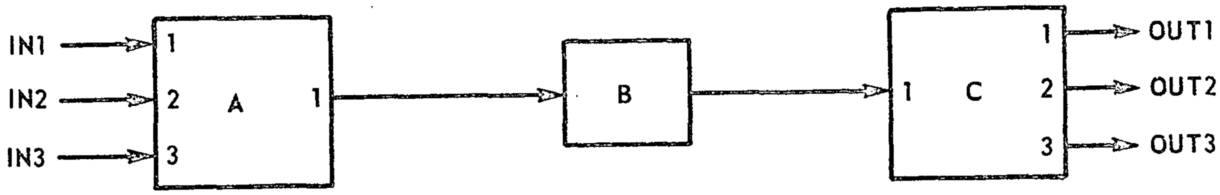
Suppose that we wish to assign the names INPUT1, INPUT2, and OUTPUT to the three terminals of ML1 in the previous example. The coding for the NAME and CONNECT statements is as follows:

```
NAME:  1#ML1, INPUT1$  :  2#ML1, INPUT2$  :  ML1#1, OUTPUT$
CONNECT:  B1, IN4, INPUT1, OUTPUT, C1$
        :  A1, INPUT2$
```

When elements with multiple inputs and a single output or a single input and multiple outputs are connected, additional simplification of the coding is possible as shown in the EXAMPLE below.

EXAMPLE

The three elements with names A, B and C are connected as shown in the sketch below.



The CONNECT statement can be coded as follows:

```
CONNECT:  IN1, 1#A#1, B, 1#C#1, OUT1 $
       :  IN2, 2#A $   : IN3, 3#A $
       :  C#2, OUT2 $   : C#3, OUT3 $
```

The first CONNECT statement can also be written as follows:

```
CONNECT:  IN1, 1#A, B, C#1, OUT1 $
```

This shortened form of the CONNECT statement is possible since A has a single output, B has a single input and output and C has a single input.

DEVICE

FORMAT -

DEVICE: a FORTRAN name, number of element input terminals, number of element output terminals, number of element parameters$

COMMENTARY -

The DEVICE statement is used to create a new Element, coded in FORTRAN, which is treated as though it were a Standard Element. This is done by specifying the mnemonic, as identified by the FORTRAN name, and referenced in a MARSYAS ELEMENTS statement, and the number of inputs, outputs and parameters the new element is to have. A FORTRAN subroutine must immediately follow the DEVICE statement and is used to add to the MARSYAS library a temporary program capable of calculating the output of the new element from its input values. A DIMENSION statement is required in the subroutine even if there are no arrays needed within the subroutine.

The subroutine name referenced in the FORTRAN SUBROUTINE statement is the same FORTRAN name referenced in the DEVICE statement. This is the element mnemonic assigned in the appropriate Description Module ELEMENTS statement. A maximum of four arguments in parenthesis may follow the subroutine name. These arguments are arbitrary real FORTRAN array names for the element inputs, outputs, parameters and time, in that order. If time is not needed in the subprogram, it may be omitted from the argument list.

If time is used, however, its value must not be altered or otherwise manipulated within the subprogram. Since the element inputs, outputs and parameters are treated as arrays within the FORTRAN subprogram, a DIMENSION statement dimensioning the arrays must appear within the subroutine. Real variable array names must be given to the dimensioned arguments. The element <u>name</u> referenced in the MARSYAS ELEMENTS statement is a MARSYAS name and, hence, is not restricted to six characters as are FORTRAN names.

EXAMPLE (1)

The user decides to construct an element with MARSYAS name ALPHAONE for use in Description Module model DELTA. The element, having a FORTRAN mnemonic name ALPHA, has three inputs, two outputs and eight parameters. The arbitrary real array names for the element inputs, outputs, parameters and time are A, B, C, and D, respectively. The coding is as follows:

```
DEVICE:  ALPHA, 3, 2, 8 $
```

FORTRAN coding

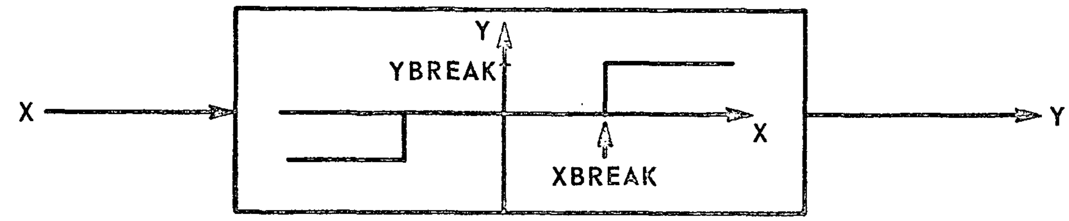
```
SUBROUTINE  ALPHA (A, B, C, D)
DIMENSION   A(3), B(2), C(8)
.        .        .        .
.        .        .        .
RETURN
END
```

MARSYAS coding

```
MODEL:  DELTA
.        .        .        .
.        .        .        .
ELEMENTS:  ALPHA, ALPHAONE (1.1, -3.0, 0.0, 6.2,
                 8.0, 2.1, 3.4, -7.6) $
.        .        .        .
END$
```

EXAMPLE (2)

Shown below is the FORTRAN coding for a relay device which is available in the Standard Elements library of MARSYAS. Following it is the MARSYAS coding referencing this device.



FORTRAN Coding

```
DEVICE:  RELAY, 1, 1, 2$
SUBROUTINE RELAY (X, Y, Z)
DIMENSION X(1), Y(1), Z(2)
YBREAK = Z(1)
XBREAK = Z(2)
IF (ABS X(1) .LT. XBREAK) Y(1) = 0.0
IF (X(1) .GE. XBREAK) Y(1) = YBREAK
IF (X(1) .LE. -XBREAK) Y(1) = -YBREAK
RETURN
END
  .       .       .       .       .
MODEL:  TIM$
INPUTS:  . . . $
OUTPUTS:  . . . $
ELEMENTS:  RELAY, RELAY1 (YBREAK, XBREAK) $
CONNECT:  . . . $
PARAMETERIZE:  RELAY1, (YBREAK, 2.0, XBREAK, 1.0)$
END$
  .       .       .       .       .
```

Note that the FORTRAN name for the DEVICE and for the input, output and parameter array names must be real variable names.

Should the need arise, the DEVICE statement can be constructed in a general manner by substituting the letter N for any or all of the number of element inputs, outputs and parameters. Then the value(s) of N is specified in the ELEMENTS statement as shown in EXAMPLE (3) below, following by the appropriate element parameters.

EXAMPLE (3)

If in EXAMPLE (2) the number of element inputs, outputs and parameters are to be varied, the coding would appear as follows:

```
DEVICE:  ALPHA, N, N, N $
SUBROUTINE   ALPHA (A, B, C, D)
DIMENSION   A(3), B(2), C(8)
.      .      .      .
.      .      .      .
RETURN
END
.      .      .      .
.      .      .      .
MODEL:  PSY$
.      .      .      .
ELEMENTS:  ALPHA, ALPHA1(2, 1, 4, 3.2, 1.1, 6.4, 9.3)$
.      .      .      .
END$
.      .      .      .
.      .      .      .
MODEL:  BETA$
.      .      .      .
ELEMENTS:  ALPHA, ALPHA15(1, 2, 3, 0.0, 2.4, 6.1)$
.      .      .      .
END$
```

Note in EXAMPLE (3) that the first three arguments in the element parameter string identify the values of N to be inserted in the device statement. The first value of N being the number of element input terminals, the second value of N being the number of element output terminals and the third value of N being the number of element parameters. The remaining numbers in parenthesis are the actual values of the element parameters.

It is permissible to place DEVICE statements, and associated subroutines anywhere within the MARSYAS program after the BEGIN statement and before the Simulation Module provided they appear prior to their first reference in an ELEMENTS statement. It is recommended, however, that all DEVICE statements be placed at the very beginning of the MARSYAS deck, after the BEGIN statement. This serves the purpose of isolating the source FORTRAN code from the MARSYAS language statements.

DISCONNECT

FORMAT -

```
DISCONNECT: name1, name2, etc. $
```

COMMENTARY -

The DISCONNECT statement is used to cancel already defined signal paths among the elements, submodels and input and output terminals. The statement is written in exactly the same manner as for the CONNECT statement. The DISCONNECT must appear after a CONNECT statement which established the connection which DISCONNECT is now to break. The DISCONNECT statement is not used with EQUATION.

EXAMPLE

If a CONNECT statement is given as

```
CONNECT: A, B, C, D $
```

a DISCONNECT statement might be

```
DISCONNECT: B, C $
```

The above two statements are now equivalent to the following statements:

```
CONNECT: A, B $
CONNECT: C, D $
```

II A. DESCRIPTION MODULE

ELEMENTS

FORMAT -

```
ELEMENTS:  mnemonic, name (parameters, if required) $
```

COMMENTARY -

MARSYAS provides the user with a larger number of operational devices called ELEMENTS, similar to those available on an analog computer, but far more numerous and versatile. These elements include items such as adders, integrators, multipliers, transfer functions, etc. A Table of Standard MARSYAS ELEMENTS appears at the back of the Manual. The Table lists the elements according to type or class, gives the element block diagram symbol, the number of inputs and outputs, the element mnemonic, the input-output relation and the appropriate parameters, if any. The user is not restricted to the "standard" set. Non-standard elements may be defined by the user through the use of the DEVICE statement. These new elements are treated in exactly the same manner as the original set furnished by the system. The ELEMENTS statement is not used when the EQUATION statement is used.

EXAMPLE

```
ELEMENTS:  AD, ADDER1, ADDER2, ADDER3 $
ELEMENTS:  CM, CM1(1.0), CM2(-3.6) $
```

ELEMENTS (Cont'd.)

The constant multiplier (CM) elements above specify the parameter constants 1.0 and -3.6. If the user desires, dummy variables may be substituted for these values and later specified in PARAMETERIZE statements.

An example of using parameter constants in an ELEMENTS statement is as follows:

```
ELEMENTS:  CM, CM1(A), CM2(B) $
    .        .      .       . .
    .        .      .       . .
    .        .      .       . .

PARAMETERIZE:  CM1 (A,   1.0) $
             :  CM2 (B,  -3.6) $
```

See CONNECT for additional comments on elements.

END

FORMAT -

```
END$
```

COMMENTARY -

The END statement is used to close a DESCRIPTION MODULE. The END statement must be the last statement in a DESCRIPTION MODULE.

EXAMPLE

```
END$
```

EQUATION

FORMAT -

EQUATION: Differential or algebraic equation $

COMMENTARY -

The EQUATION statement operator is designed to provide the user with great flexibility in specifying his model. The model can be described in terms of a system of differential and algebraic equations with almost no restrictions on their structure. The one main restriction is that a "solution scheme" of the set of equations must exist. This is discussed in detail in Section IV C. of the Manual.

The ordinary differential equations which MARSYAS solves may be linear or nonlinear and of any degree. Since the independent variable is always time in the MARSYAS system, all differentiation is assumed to be with respect to TIME. Direct differentiation of the variable name TIME or of system INPUT or OUTPUT names is not permitted. The output variables may be differentiated by assigning to them alternate names. An apostrophe is used to indicate differentiation with respect to time. Thus d^2x/dt^2 would be represented as X".

There are no restrictions on the form of the mathematical expressions contained within the equations, except that the expression itself must not be differentiated. Expressions may employ any of the standard arithmetic operations of addition (+), subtraction (or minus sign) (-), multiplication (*), division (/) and exponentiation (**). Balanced parentheses may be used where needed. See EXAMPLE (1).

Mathematical expressions may consist of MARSYAS names, mathematical operators, numerical constants and MATH REFERENCES. The use of the MATH REFERENCE adds an additional dimension to the use of the EQUATION option in that the user is permitted to take shortcuts when constructing equations, especially when the equations are used in conjunction with block diagrams. See MATH REFERENCE, Section I G.

The block diagram and equation modes may be mixed when the system main MODEL, described in block diagram form, contains SUBMODELS, some or all of which contain equations. The structure of each submodel is, of course, described as a separate MODEL.

When a MODEL contains an EQUATION statement, the statement operators ELEMENTS, PARAMETERIZE, SUBMODEL, CONNECT, DISCONNECT and NAME are not necessary and, therefore, not used. The MODEL statement must contain the word EQUATION to alert the MARSYAS processor that equations are used in the MODEL (see MODEL).

Many times, the EQUATION statement is used, in conjunction with the INPUTS and OUTPUTS statements, to build a model of a system. When used in this manner, the EQUATION operator presents a mathematical relation which determines the value of the system outputs from the value of the inputs and TIME. These relations may involve an arbitrary number of intermediate variables.

An example of using MARSYAS in this manner is illustrated in Figure 4, Section I F. Note that each equation in Figure 4 is preceded by the EQUATION statement (or its colon equivalent) and followed by the dollar sign ($) terminator.

The number of INPUTS and OUTPUTS names for a MODEL is arbitrary. There need not be any INPUTS, but there must be at least one output terminal when using EQUATION. Thus, the INPUTS statement may be omitted but the OUTPUTS statement must appear.

Initial conditions on the equations are coded in the Simulation Module as noted in the INITIALIZE statement in Section II C. of this Manual.

EXAMPLE (1)

Code the following homogeneous differential equation using the EQUATION operator.

$$\frac{d^2x}{dt^2} + \frac{dx}{dt} + cx = 0.0$$

```
EQUATION:  X'' + X' + C*X  =  0.0$
```

EXAMPLE (2)

Code the following non-homogeneous differential equation using the EQUATION operator and the MATH REFERENCE.

$$\frac{d^2x}{dt^2} + \frac{dx}{dt} + cx = 5 \sin 2 t$$

```
EQUATION:  X" + X' + C*X  =  FSIN (5.0, 2.0, 0.0, TIME) $
```

An example of using the EQUATION operator with a MARSYAS transfer function element converted to a MATH REFERENCE is shown in EXAMPLE (3) of INITIALIZE in Section II C.

II A. DESCRIPTION MODULE

INPUTS

FORMAT -

```
          INPUTS:  name1, name2, etc. $
```

COMMENTARY -

The INPUTS statement is used to give identifying names to the points in the model (input terminals) at which driving functions are to be introduced. The INPUTS statement is optional if driving functions are not needed in the model.

EXAMPLE

```
          INPUTS:  IN1, IN2, IN3$
     or
          INPUTS:  IN1$
                :  IN2, IN3$
```

II A. DESCRIPTION MODULE

MODEL

FORMAT -

```
        MODEL:  name$  or MODEL : name, EQUATION $
```

COMMENTARY -

The MODEL statement is used to give an identifying name to the model or submodel being described. Within a Description Module, all model names must be unique. MODEL must be the first statement of a Description Module and END the last. The name referenced is the user-specified model name.

EXAMPLE

```
        MODEL:  SATURN5 SIMULATOR$
        MODEL:  X-LOOP GYRO$
        MODEL:  ALPHA$
```

If the EQUATION option is chosen, then the MODEL statement includes the word EQUATION as shown in the example below.

EXAMPLE

```
        MODEL:  DELTA, EQUATION$
```

NAME

FORMAT -

NAME: original name, new name $

COMMENTARY -

The NAME statement is used to assign an additional MARSYAS name to an element, parameter, system input terminal, or system output terminal which is defined elsewhere through an ELEMENTS, PARAMETERIZE, INPUTS, or OUTPUTS statement, respectively. In addition, it may be used to assign a new name to a particular terminal of an element or its associated parameters. The NAME statement is not used in connection with the EQUATION statement.

EXAMPLE

```
NAME:  BILL, WILLIAM$
```

This NAME statement assigns the new name WILLIAM to BILL. The name BILL is not erased from the MARSYAS program and the names WILLIAM and BILL may be freely interchanged.

The NAME statement can also be used to assign names to the input(s) and output(s) of elements. See CONNECT for an example.

It is illegal to attempt to change a name previously defined by a NAME statement with a new NAME statement.

OUTPUTS

FORMAT -

```
OUTPUTS:  name1, name2, etc. $
```

COMMENTARY -

The OUTPUTS statement is used to give identifying names to the points in the model (output terminals) at which the system is to be examined. There must be at least one output terminal in the main model.

EXAMPLE

```
          OUTPUTS:  OUT1, OUT2, OUT3$

or

          OUTPUTS:  OUT1$
                 :  OUT2, OUT3$
```

PARAMETERIZE

FORMAT -

```
PARAMETERIZE:  element name (parameter number or name, numerical
                                 value)$
```

or

```
PARAMETERIZE:  parameter name (numerical value)$
```

COMMENTARY -

The PARAMETERIZE statement is used to give numerical values to specific MARSYAS parameters or to change parameter values previously assigned. Parameters are identified through the names given in ELEMENTS and NAME statements. The PARAMETERIZE statement is not used with EQUATION.

EXAMPLE

Suppose the following is referenced in an ELEMENTS statement

```
ELEMENTS:  TF, TF1(2, 2.0, 1.0, PHY, 1.8, 8.5, BETA)$
```

Then the PARAMETERIZE statement might read

```
PARAMETERIZE:  TF1 (PHY, 6.0)$  :  TF1 (BETA, -1.0E-2)$
```

or

```
PARAMETERIZE:  PHY (6.0)$  : BETA (-1.0E-2)$
```

The PARAMETERIZE statement can be used to alter the value of a parameter previously specified in an ELEMENTS statement. The user may wish to use this method to effect a parameter change rather than recode the original ELEMENTS statement, since it requires less work. If an element contains several parameters, the PARAMETERIZE statement effecting a change in parameters can be shortened by specifying the position of the parameter within the parentheses, along with its revised value. See CHANGE in Section II C.

EXAMPLE

For the ELEMENTS statement shown below, change the value of the fifth parameter from 1.8 to 15.2.

```
ELEMENTS:  TF, TF6(2, 1.3, 2.1, 7.4, 1.8, 3.4, 9.7) $

PARAMETERIZE:  TF6(5, 15.2)$
```

The ELEMENTS then appears to the system as if it had been originally coded as

```
ELEMENTS:  TF, TF6(2, 1.3, 2.1, 7.4, 15.2, 3.4, 9.7) $
```

SUBMODEL

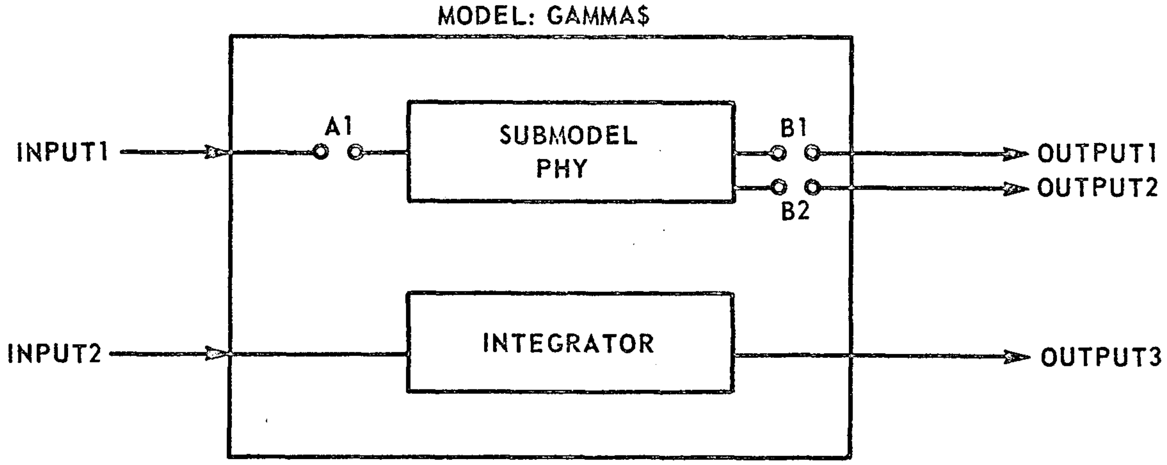
FORMAT -

SUBMODEL: name ; INPUTS: a, b, c, d, etc; OUTPUTS: h, i, j, k, etc$

COMMENTARY -

A submodel is a model wholly contained within another model. The SUBMODEL statement is needed to give identifying names to the submodel and its input and output terminals. These terminal names will be used to CONNECT the submodel to the main model. Submodels can be imbedded within submodels. The structure of each submodel must be described in a Description Module separate from the one in which it is imbedded.

EXAMPLE



II.A. DESCRIPTION MODULE

SUBMODEL (Cont'd.)

The model GAMMA contains one submodel and an integrator. It could be coded as follows:

Description Module for GAMMA

```
           .        .        .        .        .
           MODEL:      GAMMA$
           INPUTS:     INPUT1, INPUT2$
           OUTPUTS:    OUTPUT1, OUTPUT2, OUTPUT3$
           ELEMENTS:   INTEGRATOR$
           SUBMODEL:   PHY; INPUTS:A1;  OUTPUTS: B1, B2$
           CONNECT:    INPUT1, A1$
                  :    B1, OUTPUT1$
                  :    B2, OUTPUT2$  : INPUT2, INTEGRATOR, OUTPUT3$
           END$
```

Description Module for PHY

```
           .        .        .        .        .
           MODEL:      PHY$
           INPUTS:     A1$
           OUTPUTS:    B1, B2$
           ELEMENTS:   ___$ }
                            } Describes the configurations of PHY
           CONNECT:    ___$ }
           END$
           .        .        .        .        .        .
```

NOTE: In describing the submodel, the input and output terminal names of the submodel need not be the same as those defined in the main model. However, there must be a one-to-one correspondence between the number and order of the terminals specified in both models' INPUTS and OUTPUTS statements.

MARSYAS names used within a particular MODEL must be unique. However, names which are used in a submodel can be repeated in the main MODEL.

II B. MODIFICATION MODULE

B. Modification Module

The Modification Module is used to alter an already existing model by adding or deleting elements, input or output terminals, and submodels and/or by changing the interconnections among these items. A MARSYAS program may contain several Modification Modules.

Figure 6 is a list of statements which can appear in the Modification Module with brief comments as to their function. A detailed explanation of a few of those statements, not previously discussed, is given in the following pages. Explanations of the remaining statements appear in the Description Module portion of this Manual. The order in which the statements appear is immaterial except that MODIFY must be first and END last. DISCONNECT, DELETE and SUBSTITUTE can be used to alter the Description Module statements which defined the original connection (CONNECT) which is now to be broken or the original element (ELEMENTS) which is now to be deleted or substituted. DEVICE may appear at any point in the program provided it is used before the first appearance of the new element type in an ELEMENTS statement. The INPUTS and OUTPUTS statements are used to define additional input and output terminals, respectively. The SUBMODEL, ELEMENTS and DEVICE statements are used to add additional submodels, elements and non-standard elements, respectively. All additional terminals, if any, must be connected within the Modification Module. The PARAMETERIZE and NAME statements have the same functions as described within the Description Module. When the EQUATION option is used in the Description Module, there can be no Modification Module since no ELEMENTS, element names, CONNECTs or parameters will exist there.

If the system contains several Description and Modification Modules, the only requirement for ordering the Modules within a MARSYAS program is that the Modification Module must follow the Description Module it modifies. For example, a MARSYAS program might contain a group of three Description Modules followed by a group of three Modification Modules.

II B. MODIFICATION MODULE

OPERATOR	FUNCTION
CONNECT	Same functions as in Description Module.
DELETE	Deletes an already specified element or input or output terminal, and removes all connections to or from that element or terminal.
DEVICE	Defines a non-standard element.
DISCONNECT	Same functions as in Description Module.
ELEMENTS	Adds additional elements to the model.
END	Marks the end of a Modification Module.
INPUTS	Adds additional input terminals to the model.
MODIFY	Marks the appearance of a modification module and can give a new name to the modified model.
NAME	Same functions as in Description Module.
OUTPUTS	Adds additional output terminals to the model.
PARAMETERIZE	Assigns or changes numerical values of parameters of elements present in either a Description or Modification model.
SUBMODEL	Adds additional submodels to the model.
SUBSTITUTE	Substitutes one element for a previously defined element. The number of inputs and outputs must be the same for both elements.
*	Same functions as in Description Module.

FIGURE 6. STATEMENT OPERATORS USED IN THE MODIFICATION MODULE

II B. MODIFICATION MODULE

DELETE

FORMAT -

```
DELETE:  name1, name2, name3, etc.$
```

COMMENTARY -

The DELETE statement removes an element, input terminal or output terminal from the model being modified. All connections associated with the deleted items are broken. The user must reconnect (using CONNECT statements), the elements and terminals which are affected by the DELETE statement.

EXAMPLE

If the unmodified model ELEMENTS statement is

```
ELEMENTS:  AD, AD1, AD2, AD3$
```

and a Modification Module DELETE statement is

```
DELETE:  AD2$
```

then the effect is that the original model would look as if it had been coded as

```
ELEMENTS:  AD, AD1, AD3$
```

The user must reconnect the deleted connections associated with the deleted element, AD2.

II B. MODIFICATION MODULE

END

FORMAT -

```
END$
```

COMMENTARY -

The END statement is used to close a Modification Module. The END statement must be the last one in a Modification Module.

EXAMPLE

```
END$
```

II B. MODIFICATION MODULE

MODIFY

FORMAT -

```
MODIFY:  model name, new model name (optional)$
```

COMMENTARY -

The MODIFY statement is used to identify the model which is to be modified and optionally to give the modified model a new name. If a new name is not given, the modified model has the same name as the original model. MODIFY must be the first statement of the Modification Module.

EXAMPLE

```
MODIFY:  ALPHA$
MODIFY:  ALPHA, BETA$
```

The first statement identifies the model which is to be modified. The second statement may be used in place of the first if the user desires to retain the original model, ALPHA, and create a new model BETA which will be a modified version of ALPHA.

II B. MODIFICATION MODULE

SUBSTITUTE

FORMAT -

```
SUBSTITUTE:  original element name, replacement element name$
```

COMMENTARY -

The SUBSTITUTE statement is used to effect the complete replacement of one element by another element. The elements do not have to be of the same type as long as each element has the same number of input and output terminals. For example, a power function could be substituted for an integrator since both have one input and one output. All connections to and from the original element are replaced by the connections to and from the substitute element.

EXAMPLE

If the original model contained the following statements,

```
ELEMENTS:     CM, CM1$
        :     AD, AD1, AD2$
CONNECT :     AD1, CM1, AD2$
```

and the Modification Module contained the statements,

```
SUBSTITUTE:   CM1, IN1$
ELEMENTS:     IN, IN1$
```

SUBSTITUTE (Cont'd.)

then the effect is that the original model would look as if it had been coded as

```
ELEMENTS:      AD, AD1, AD2$  : IN, IN1$
CONNECT:       AD1, IN1, AD2$
```

Note: It is up to the user to determine whether a particular substitution is physically meaningful.

C. Simulation Module

The Simulation Module completes the description of the system whose structure is defined in a Description Module(s). A MARSYAS program can contain only one Simulation Module. In this Module, the user specifies the INPUT excitation function(s), the numerical integration scheme to be used in the problem solution and any required initial conditions. The conditions for stopping the simulation are also specified in the Simulation Module.

A sequential series of simulations can be run without resubmitting the MARSYAS deck by using CHANGE statements. These permit the user to alter one or more of the following statements without altering the original coding: EXCITE, INITIALIZE, STOP IF, TERMINATE IF and PARAMETERS.

Excitation functions are impressed upon the model input terminals using EXCITE statements. A Table of Standard Excitation functions is part of the MARSYAS system and is shown at the back of this Manual. Additional excitation functions may be defined by the user through FUNCTION statements.

Initial conditions (other than zero) on integrators or transfer functions are specified with the INITIALIZE statement.

There are currently five methods of numerical integration available in the MARSYAS system, Euler's 1st-order, Butcher's 5th-order, Sarafyan 5th-order variable-step, 4th-order Runga-Kutta and Adams-Bashforth predictor-corrector. These integration methods are discussed in Section IV B. The MARSYAS system automatically selects the Sarafyan variable-step method unless instructed otherwise by an INTEGRATE statement. If the user chooses the Adams-Bashforth method, the MARSYAS system automatically selects a relative error of 0.002 unless otherwise specified. In addition, Runga-Kutta and Butcher's methods will be run using a step size of 0.01 unless otherwise specified.

Conditions for temporarily or permanently halting the simulation are given by the STOP IF and TERMINATE IF statements, respectively. STOP IF is used to halt each simulation when using CHANGE statements.

Constant gain elements (Constant Multipliers) can be changed into time-varying multipliers through the use of the VARY GAIN statement.

If the user has described his system in block-diagram form in the Description Module, linear and/or nonlinear "loops" may be present. The absence of an integrator or transfer function in a closed path produces a loop. A loop is nonlinear if it contains at least one nonlinear element (such as a power function). Otherwise it is linear. In the MARSYAS system, loops are automatically solved using either a Newton-Raphson or a successive approximation technique and the elements within these loops are automatically listed for the user by the MARSYAS processor. The ESTIMATE statement can be used to assign initial values to the outputs of nonlinear elements within a nonlinear loop(s) should the user desire to do so, otherwise a value of zero is assumed.

A summary of the statements which appear in the Simulation Module is shown in Figure 7 along with brief comments as to their function. Details of these statement operators are given in the following pages.

II C. SIMULATION MODULE

The order in which the statements appear in the MARSYAS deck is immaterial except that SIMULATE must be the first statement of the Simulation Module and END the last. FUNCTION does not have to appear within the Simulation Module, but must appear before it is referenced in an EXCITE statement. It is recommended that all FUNCTION statements be placed at the very beginning of the MARSYAS deck, after any DEVICE statements or just following the BEGIN statement.

II C. SIMULATION MODULE

OPERATOR	FUNCTION
CHANGE	Creates additional simulation runs with changes in the model.
END	Marks the end of the Simulation Module.
ESTIMATE	Specifies initial values for the outputs of nonlinear elements within a nonlinear loop.
EXCITE	Specifies the excitation functions to be applied to the input terminals.
FUNCTION	Defines a non-standard excitation function.
INITIALIZE	Specifies initial conditions for integrators or transfer function elements.
INTEGRATE	Specifies the integration method to be used.
PARAMETERS	Specifies new parameters for a Description Module element.
SIMULATE	Marks the appearance of the Simulation Module.
STOP IF TERMINATE IF	Specifies the condition or conditions for stopping a simulation either temporarily (STOP) or permanently (TERMINATE).
VARY GAIN	Defines time varying coefficients.
*	Provides a comment for the user.

FIGURE 7. STATEMENT OPERATORS USED IN THE SIMULATION MODULE

II C. SIMULATION MODULE

CHANGE

FORMAT -

CHANGE: name of change; Statement operator : Statement format $

Note: For each change statement, a separate simulation is performed.

COMMENTARY -

The CHANGE statement is used when repetitive simulations are desired. For example, the user may wish to observe the behavior of his model when certain parameters, excitation(s), initial conditions or stopping conditions have been changed. A series of simulations can be set up in one run, thus saving valuable user and computer time. When using CHANGE, the user specifies an identifying name for the new simulation, the statement operator being changed and the statement format. The statement operators used with the CHANGE statement are PARAMETERS, EXCITE, INITIALIZE, STOP IF and TERMINATE IF.

When changing Description Module ELEMENT parameters, the user must specify the name of the MODEL in which the element occurs, the element name, the position number of the parameter in the parameter string and changed parameter values.

EXAMPLE

The following ELEMENTS statement appears in the Description Module model PSY

```
ELEMENTS: TF, TF6(2, 1.3, 2.1, 7.4, 1.8, 3.4, 9.7)$
```

and the user desires to change the fifth value in the parameter string to 15.2. The CHANGE statement would be coded as follows:

```
CHANGE: NEW TF6; PARAMETERS: PSY, TF6(5, 15.2) $
```

This causes the original ELEMENTS statement to be processed as if it had originally been coded as

```
ELEMENTS: TF, TF6(2, 1.3, 2.1, 7.4, 15.2, 3.4, 9.7)$
```

If one of the parameters in an ELEMENTS statement is a MARSYAS name rather than a real number, the same technique as described above can be used to change its value. Consider the same ELEMENTS statement as shown in the above example except that the fifth element which we want to change has the name BILL as shown below:

```
ELEMENTS: TF, TF6(2, 1.3, 2.1, 7.4, BILL, 1.8, 3.4, 9.7)$
```

Changing the value of BILL to 15.2 can be accomplished in the following manner:

```
CHANGE:  NEW TF6; PARAMETERS: PSY, TF6(5, 15.2)$
```

Since the name BILL is unique within the MODEL where it is used, the CHANGE statement can be shortened somewhat to take advantage of this fact by referencing the parameter name rather than the element name. The previous CHANGE statement could then be written as follows:

```
CHANGE:  NEW TF6; PARAMETERS:  PSY, BILL(15.2)$
```

If several parameters of one element are to be altered using one CHANGE statement, the position numbers and new parameter values are listed in pairs, in any sequence, as shown below.

EXAMPLE

The following time varying coefficient element appears in the Description Module model TIM

```
ELEMENTS:  TV, TV1(3, 0.0, 0.0, 4.2, 1.6, 5.4, -3.2)$
```

and the user desires to change the last four parameters to the following values: 5.1, 1.93, 6.2, 1.1. The MARSYAS coding is as follows:

```
CHANGE:  RUN3; PARAMETERS: TIM, TV1(4, 5.1, 5, 1.93, 6, 5.4, 7, 1.1)$
```

CHANGE (Cont'd.)

An alternate method of coding the previous statement is as follows:

```
CHANGE:  RUN3; PARAMETERS: TIM, TV1(4,5.1)(5,1.93)(6,5.4)(7,1.1)$
```

If an element contains only one parameter, the position number of the parameter is one. The coding for changing the value of a constant multiplier is shown in the example below.

EXAMPLE

If the constant multiplier element with name CM1 and parameter value 3.0 exists in the Description Module model ALPHA and the user desires to change the multiplier parameter value to 6.4, then the Simulation Module CHANGE statement with new simulation name NEW CM2 would read

```
CHANGE:  NEW CM2; PARAMETERS: ALPHA, CM2 (1, 6.4)$
```

If the user desires to change the Simulation Module EXCITE, INITIALIZE, STOP IF and TERMINATE IF statements, the form of the statement(s) is exactly as described elsewhere in this section of the Manual except that each statement is preceded by the word CHANGE and the title of the change.

EXAMPLE

CHANGE: name of change; EXCITE : input terminal name, new excitation function mnemonic$

CHANGE: name of change; INITIALIZE : model name, element name (new initial conditions)$

CHANGE: name of change; STOP IF : new logical expression$

CHANGE: name of change; TERMINATE IF : new logical expression$

Except for STOP IF and TERMINATE IF, CHANGE statement alterations are permanent unless changed again in subsequent simulations. STOP IF statement changes are temporary and are valid only for the simulation specified by the given CHANGE statement.

If there are "n" CHANGE statements in the Simulation Module, there will be "n+1" simulations executed by the MARSYAS processor. The additional simulation being that of the system as originally configured, as if no CHANGE statements were present. If the user desires to omit the original simulation and have the MARSYAS processor proceed to the simulation as specified by the first CHANGE statement, he must insert the following statement anywhere in the Simulation Module.

CHANGE: (DELETE) $

The simulations will then proceed, one by one, until all of the CHANGE statements have been executed <u>or</u> until a TERMINATE IF statement appears.

When a TERMINATE IF statement is executed, the MARSYAS program is halted and no further CHANGE statements are executed. This is not the case with the STOP IF statement. For example, after changing one or more element parameters, initial conditions or excitation functions, the user may decide to examine the system outputs after a short period of simulation time, then proceed to the next CHANGE statement. The STOP IF statement is used for this purpose.

EXAMPLE

Let us assume that the user decides to change one of the system EXCITE functions to a ramp and also to change the simulation time. The CHANGE statement could be coded as follows:

```
CHANGE: RUN  ONE ; EXCITE : INPUT6, FRAMP (-2.0); STOP IF : TIME .GT. 10.0$
                                                ↑
                                                | Note - Semicolon
```

When there is more than one change specified in a particular CHANGE statement, each is separated by a semicolon as shown in the example above.

END

FORMAT -

```
          END$
```

COMMENTARY -

The END statement is used to close the Simulation Module. The END statement must be the last onc in the Simulation Module.

EXAMPLE

```
          END$
```

ESTIMATE

FORMAT -

ESTIMATE: model name, element name (output(s) estimate)$

COMMENTARY -

The ESTIMATE statement is used to assign initial values to the outputs of non-linear elements which are part of the non-linear loop. When using the ESTIMATE statement, the user specifies the initial estimate of the output of a non-linear element which is part of a non-linear loop. The use of ESTIMATE is similar to that of INITIALIZE which specifies the initial conditions on integrators or transfer functions. The use of ESTIMATE is optional and is intended for use as an aid, if needed, in solving imbedded non-linear loops.

EXAMPLE

If a non-linear loop in model ALPHA contains a power function element whose name is PF1, and whose user specified estimate at time zero is 1.0, then the ESTIMATE statement would be

ESTIMATE: ALPHA, PF1 (1.0)$

Should a non-linear element contain two outputs, such as an output relay (with name R01), then the ESTIMATE statement would be

```
ESTIMATE:  ALPHA, R01 (1.5, 3.2)$
```

II C. SIMULATION MODULE

EXCITE

FORMAT -

```
EXCITE:  system input name, excitation function mnemonic (parameters)$
```

COMMENTARY -

The EXCITE statement is used to specify an excitation function at each input terminal. The user must specify the excitation mnemonic and the parameters associated with the given excitation. A Table of Standard MARSYAS EXCITE statements appears at the back of this Manual. The Table lists the excitation function mnemonics and parameters, the type of excitation, the mathematical description of the excitation function, and a graph of the function versus time. The user is not restricted to the "standard" set. Non-standard excitations may be defined by the user through the use of the FUNCTION statement (see FUNCTION). When coding, non-standard excitations are treated in exactly the same manner as the standard MARSYAS set.

EXAMPLE

```
EXCITE:  INPUT1, FSIN (5.0, 2.0, 0.0)$
```

This statement causes the MARSYAS processor to excite terminal INPUT1 with the function 5 sin 2t.

EXCITE (Cont'd.)

The same excitation function can be used to excite several different inputs as shown in the example below.

EXAMPLE

The system contains inputs IN1, IN2, IN3 which are to be excited with a step function of magnitude 3.2. The coding required to effect this is as follows:

```
     EXCITE:  IN1, IN2, IN3, FSTEP(3.2)$
or
     EXCITE:  IN1, FSTEP(3.2)$  : IN2, FSTEP(3.2)$
           :  IN3, FSTEP(3.2)$
```

FUNCTION

FORMAT -

FUNCTION: a FORTRAN name, number of parameters$

COMMENTARY -

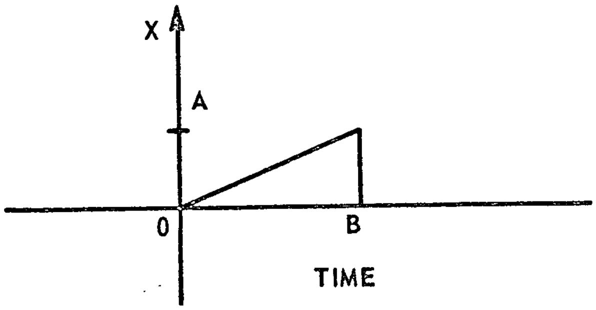
The FUNCTION statement is used by the programmer to create a new excitation function, coded in FORTRAN, which is treated as though it were a MARSYAS Standard Excitation Function. This is done by specifying the new mnemonic as identified by the FORTRAN name, and the number of mathematical parameters required to compute the excitation function. A FORTRAN FUNCTION subprogram must immediately follow the MARSYAS FUNCTION statement as formatted above. The user-created FORTRAN FUNCTION subprogram then computes the output of the new excitation function from the parameter values specified in the EXCITE statement. (See EXCITE.)

The subprogram name referenced in the FORTRAN FUNCTION statement is the same FORTRAN name referenced in the MARSYAS FUNCTION statement. This is the excitation function mnemonic assigned in the Simulation Module EXCITE statement. The arguments in parenthesis which follow the FORTRAN FUNCTION name are arbitrary real FORTRAN names for the parameters required to compute the excitation function in the subprogram. The last argument in the parameter list must be a symbol for time. The value of time must not be altered or otherwise manipulated within the subprogram. The excitation name referenced in the MARSYAS EXCITE statement is a MARSYAS name and, hence, is not restricted to six characters as are FORTRAN names.

It is permissible to place the FUNCTION statement(s) and associated subroutine(s) anywhere within the MARSYAS program after the BEGIN statement and before the Post Processing Module provided they appear prior to being referenced in EXCITE statements. It is recommended, however, that all FUNCTION statements be placed at the beginning of the deck, after or shortly following, the BEGIN statement to isolate the source FORTRAN code from the MARSYAS language statements.

EXAMPLE

Shown below is the FORTRAN coding for the truncated ramp excitation function shown. Following it is the MARSYAS coding referencing this FUNCTION.



FUNCTION (Cont'd.)

```
.       .       .       .       .       .

FUNCTION:  RAMP, 3 $
```

FORTRAN Coding

```
FUNCTION   RAMP (A, B, T)

IF (( T .GE. 0.0) .AND. (T .LT. B))  X = A/B * T

IF (T .GE. B)  X = 0.0

RAMP = X

RETURN

END
```

```
.       .       .       .       .       .
.       .       .       .       .       .

SIMULATE:  EPOCH$
.       .       .       .       .       .

EXCITE:  INPUT6, RAMP (2.0, 2.0)$
.       .       .       .       .       .

END$
```

II C. SIMULATION MODULE

INITIALIZE

FORMAT -

INITIALIZE: model name, element name (initial conditions)$

COMMENTARY -

The INITIALIZE statement is used to assign initial conditions to the integrators and transfer functions defined in the MARSYAS program. When using the block-diagram approach, the user must specify the name of the model in which the integrator or transfer function appears along with its element name and initial conditions. If specifying the initial conditions of an n^{th}-order transfer function, n-initial conditions must be specified in the ascending order of the derivatives as shown in EXAMPLE (1). If the INITIALIZE statement is omitted, all initial conditions are defaulted to zero, automatically.

When using the EQUATION mode of solution as defined in the Description Module, the initial conditions are specified by stating the model name followed by each derivative and its initial condition. If one of the derivatives is omitted, then its initial condition is automatically defaulted to zero. See EXAMPLE (2).

When using a model which has a MATH REFERENCE transfer function, the initial condition on the transfer function is specified as shown in EXAMPLE (3).

EXAMPLE (1)

If model ALPHA contains an integrator with name IN1, initial condition of 3.6, and second order transfer function with name TF6, where TF6 is

$$\frac{y(s)}{u(s)} = \frac{s + 1}{s^2 + 2s + 1} \quad \text{with } y(0) = 1.2 \text{ and } y'(0) = 4.4$$

then the initial conditions on each are specified as follows:

```
INITIALIZE:  ALPHA, IN1 (3.6)$
          :  ALPHA, TF6 (1.2, 4.4)$
```

EXAMPLE (2)

Suppose in model BETA, which uses EQUATION, the variable X as well as the derivatives X'''', X''', X'', and X' appear. Then the non-zero initial conditions would be expressed as follows:

```
INITIALIZE:  BETA, X'''(1.6), X'' (2.2)$
```

The initial condition on X' would automatically be set to zero since it has been omitted from the INITIALIZE statement.

EXAMPLE (3)

Suppose we wish to code the following expression in EQUATION form in a Description Module model whose name is PHY.

$$\frac{Y(s)}{X(s)} = \frac{1}{s + 1}$$

Using the MARSYAS Transfer Function element as a MATH REFERENCE, the above could be written in EQUATION form as follows:

```
MODEL:  PHY, EQUATION$

.       .       .       .

Y  =  TF (1, 0, 1, 1, 1, X)$

.       .       .       .

END$
```

Then, in the Simulation Module, the INITIALIZE statement would read

```
INITIALIZE:  PHY, Y (1.6)$
```

C2

INTEGRATE

FORMAT -

INTEGRATE: mnemonic, code word, real number $

COMMENTARY -

The INTEGRATE statement (optional) is used to specify the integration algorithm to be used in the problem solution. Five methods of integration are presently available in the MARSYAS system, Euler's (1st-order), Butcher's (5th-order), Sarafyan (5th-order) variable-step, Runga-Kutta (4th-order) and Adams-Bashforth predictor-corrector. Other numerical integration methods soon will be added to the MARSYAS language allowing the user greater flexibility in solving his problem. Only one INTEGRATE statement can appear in a MARSYAS deck. When choosing the method of integration desired, the user specifies its mnemonic, the code word TIMESTEP, and integration step size. The mnemonics for each of the integration methods currently available are as follows: Euler's (EU), Butcher's (BU), Sarafyan (SA), Runga-Kutta (RK), Adams-Bashforth (AB). The timestep may be specified only for Euler, Butcher and Runga-Kutta methods. If the timestep is omitted, the simulations will be performed with an integration step size of 0.01.

If the user omits the INTEGRATE statement, the simulation will be run using the Sarafyan method. If the user chooses the Adams-Bashforth method, the MARSYAS system automatically selects a relative error of 0.002.

An absolute error or other relative error may be selected if the code word ABSERR or RELERR, respectively, is used in an INTEGRATE statement, followed by the user specified error.

If the simulation problem contains a Sample and Hold element(s), Butcher's method or the Runga-Kutta method of integration should be used, with the TIMESTEP smaller than the smallest Sample and Hold interval.

EXAMPLES

a) INTEGRATE: RK, TIMESTEP, 0.01$

b) INTEGRATE: AB, RELERR, 0.05$

c) INTEGRATE: AB, ABSERR, 0.001$

d) INTEGRATE: BU, TIMESTEP, 0.01$

e) INTEGRATE: EU, TIMESTEP, 0.01$

Euler's method is implemented automatically whenever the MARSYAS processor detects the presence of a discontinuity. Thus, if a discontinuity occurs at time, t_1, the integration method will change to the Euler method at time $t_1-\epsilon$ and back to the method of integration used before the discontinuity was encountered, at time $t_1+\epsilon$. The reason for the Euler-interrupt is that the lower order method gives better results at discontinuities than the higher order methods. See Section IV B.

To disable the Euler-interrupt at discontinuities so that the chosen integration scheme continues without interruption, simply insert the word XEULER after the timestep or error value in the INTEGRATE statement as follows:

```
INTEGRATE:  RK, TIMESTEP, 0.001, XEULER$
```

II C. SIMULATION MODULE

PARAMETERS

FORMAT -

PARAMETERS: model name, element name (parameter number, numerical value)$

or

PARAMETERS: model name, parameter name (numerical value)$

COMMENTARY -

The PARAMETERS statement is used to assign numerical values to Description Module element parameters in much the same way as the Description Module PARAMETERIZE statement (Section II A.). The only difference between the two operators is that the model name must be specified in the PARAMETERS statement, whereas it is not required in the PARAMETERIZE statement.

The user might choose to use this operator when altering parameters in several different Description Modules thus eliminating the need for changing his original coding.

EXAMPLE

```
PARAMETERS:  PSY, CM1 (1, 2.2)$
```

See PARAMETERIZE and CHANGE for additional comments and examples.

II C. SIMULATION MODULE

SIMULATE

FORMAT -

SIMULATE: System model name$

COMMENTARY -

The SIMULATE statement is used to identify the system being simulated. The system model name is the main model name specified in the Description Module. Since there is only one system model, any other models appearing in the Description Module describe the structure of submodels appearing in the main system model.

SIMULATE must be the first and END the last statement of this module.

EXAMPLE

SIMULATE: SPACE SHUTTLE ENGINE$

STOP IF

FORMAT -

```
STOP IF:  logical expression $
```

COMMENTARY -

The STOP IF statement specifies the condition(s) for temporarily halting a simulation. When more than one simulation is to be executed (via CHANGE statements), a halt caused by STOP IF allows the execution of the next simulation to proceed. This is contrasted with a halt caused by TERMINATE IF which permanently stops the simulation and prevents the execution of any additional simulations. There is no difference between STOP IF and TERMINATE IF if no CHANGE statements are present. The construction of the logical expression part of the STOP IF statement is described under TERMINATE IF. There may be as many STOP IF statements as the user may require to control the simulation.

EXAMPLE

```
STOP IF:  TIME .GT. 20.0$
STOP IF:  (OUTPUT6 .LE. OUTPUT3)$
STOP IF:  (INPUT1 .AND. INPUT2) .GE. 60.0$
```

TERMINATE IF

FORMAT -

```
TERMINATE IF:  logical expression $
```

COMMENTARY -

The TERMINATE IF statement specifies the condition(s) for permanently halting a simulation. The halting conditions are expressed using standard FORTRAN logical expressions. The logical operators AND, OR, NOT and relational operators EQ, NE, GT, GE, LE and LT are used to form logical expressions involving TIME, and/or model INPUT or OUTPUT names. A halt caused by TERMINATE IF halts the simulation being executed and no additional simulations will be processed even though CHANGE statements may be present. (See STOP IF.) There may be as many TERMINATE IF statements as the user may require to control the simulations.

EXAMPLE

```
TERMINATE IF:  TIME .GT. 5.0$
TERMINATE IF:  (OUTPUT1 .OR. OUTPUT2) .GT. 3.2$
TERMINATE IF:  (INPUT1 .AND. INPUT2) .EQ. 60.0$
```

Testing on equality as shown above should be avoided because the equality may never be exactly satisfied.

VARY GAIN

FORMAT -

$$\text{VARY GAIN:}\quad \text{model name,}\ \begin{matrix}\text{constant multiplier}\\ \text{element } \underline{\text{name}}\end{matrix}\ \begin{pmatrix}\text{algebraic expression}\\ \text{or numerical list}\end{pmatrix}\$$$

COMMENTARY -

The VARY GAIN statement is used to change Constant Multiplier elements into Time Varying multipliers. This feature permits the user to solve different equations with time varying coefficients or solve models which contain time varying multipliers. Recall that Constant Multiplier (CM) elements are assigned names in a DESCRIPTION Module using an ELEMENTS statement. These CM element names along with the model name of the DESCRIPTION Module in which they appear are used in the VARY GAIN statement above.

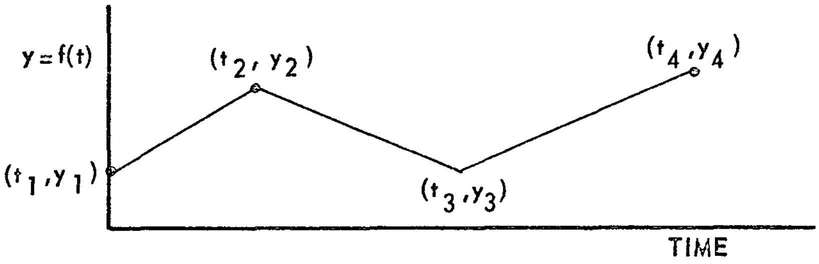
EXAMPLE

Suppose in model BETA, with Constant Multiplier element names MULT1(36.3), MULT2(0.05) and MULT3(-3.4), the user desires to change MULT2 to $3t^2 + 5.2 \cos t$. Then the VARY GAIN statement would be expressed as follows:

```
VARY GAIN:  BETA, MULT2('3.0 * TIME ** 2 + 5.2 * COS(TIME)')$
```

Note the quote marks ' ' which must enclose the entire expression.

Should the time-varying nature of the function be expressed as a linear piecewise continuous curve, then the coordinates of the curve at the discontinuities are sufficient to define the function as shown below



which is represented by the numerical list

$(n, t_1, y_1, t_2, y_2, \ldots, t_n, y_n)$

where the first number in the list, n, is an integer specifying the number points defining the function (≤100 points), and the numbers following n are the coordinates of the points. Note that the numerical list is not enclosed by quote marks.

EXAMPLE

In the previous example, let MULT2 be a piecewise continuous function similar to the plot above. Then the VARY GAIN statement would read

```
VARY GAIN:  BETA, MULT2 (4, 0.0, 1.0, 1.2, 2.2, 2.8, 1.0, 4.3, 2.4)$
```

II D. POST PROCESSING MODULE

D. Post Processing Module

The function of the Post Processing Module, which always follows the Simulation Module, is to specify the output format for the MARSYAS processor. In addition, the Post Processing Module contains a Fast Fourier Transform (FFT) processor which is explained later in this section. A MARSYAS program can contain only one Post Processing Module.

The following is a brief description of the terminology which the user will encounter in reading this section of the Manual.

The PRINT statement is used to specify which output variables are to be printed while the SAMPLE statement indicates the print interval. If all or part of the output is to be plotted, a PLOT statement is required. Should the frequency response at any output terminal be desired, the FOURIER statement is used.

A summary list of the statement operators used in this section appears in Figure 8 along with brief comments as to their function. The order in which the statements appear within the Post Processing Module is immaterial except that END must be the last statement.

A more detailed description of these operators as well as examples as to their use appears in alphabetical order on the following pages.

II.D. POST PROCESSING MODULE

OPERATOR	FUNCTION
END	Marks the end of the Post Processing Module.
FOURIER	Calculates the frequency response of the system inputs or outputs specified.
PLOT	Defines the variables to be plotted and the way they are to be plotted.
PRINT	Specifies the variables to be printed.
SAMPLE	Defines the sampling rate at which the variables are printed.
*	Provides comment for the user.

FIGURE 8. STATEMENT OPERATORS USED IN THE POST PROCESSING MODULE

END

FORMAT -

```
        END$
```

COMMENTARY -

This should be the last statement in the post-processing module.

II D. POST-PROCESSING MODULE

FOURIER

FORMAT -

FOURIER: name 1 (period length), name 2 (period length), etc. $

COMMENTARY -

The FOURIER statement provides the user with a frequency response of the system inputs or outputs specified. The frequency response consists of a Fourier Transform magnitude and phase spectrum and power spectral density (PSD). The waveform period length must be specified by the user and is indicated by a real number in parenthesis after the input or output terminal name. A maximum of 2048 samples are available for use in obtaining the frequency response of each terminal. Since Runga-Kutta is the only integration scheme permitted when using FOURIER, the user must coordinate his integration step size with the period length to take advantage of the maximum number of sample points which the MARSYAS processor is capable of handling. In addition to tabulation data output, the user is provided with linear plots of PSD, magnitude and phase angle versus frequency.

EXAMPLE

The user has decided to obtain the frequency spectrum of his system at output LAMBDA. After having examined the time response at LAMBDA, he decides that a period length of 16.0 seconds is required to define his waveform. In order to utilize all 2048 permissible sample points, the user computes his Runga-Kutta step size to be

$$16.0 \ \frac{\text{seconds}}{\text{period}} \div 2048 \ \frac{\text{points}}{\text{period}} = 0.0078125 \text{ seconds/sample point.}$$

This is the integration TIMESTEP which must be used as part of the INTEGRATE statement in the Simulation Module. The FOURIER statement is coded as follows:

```
FOURIER: LAMBDA (16.0)$
```

It is not necessary for the user to use all 2048 sample points in order to obtain the frequency response. However, the use of the maximum number of points (2048) will always give more accurate results than when using fewer points.

II D. POST-PROCESSING MODULE

PLOT

FORMAT -

PLOT: $\begin{array}{l}\text{grid-type}\\\text{mnemonic}\end{array}$ (TIME1,TIME2,CODE), N(X1,Y1, X2,Y2, X3,Y3)$

COMMENTARY -

The PLOT statement is used to obtain automatically scaled plots of the INPUTS and/or OUTPUTS of the system being simulated. The grid type is specified by the mnemonics LINEAR, SEMILG, LOGLOG and LOGSEM. Figure 9 summarizes the functions of the grid type mnemonics. TIME1 and TIME2 are the plot start and stop reference times. CODE is an integer, 1 or 2, indicating the nature of the independent variable(s) being plotted. If CODE is 1, the independent variable being plotted is TIME; and, if CODE is 2, it is not TIME. Code equal to one may be implied, if desired, by closing the parenthesis after TIME2.

The integer N is a positive or negative number which designates the number of frames of graph paper to be used when plotting the data. N can be greater than one only when TIME is the independent variable being plotted. If N is one, both N and the parenthesis may be omitted.

A <u>frame</u> is a unit of output paper approximately 7.5 inches square. Two frames would be 7.5 inches x 15 inches; three frames would be 7.5 inches x 22.5 inches; etc. If the user decides to spread out the TIME axis, he can do so by specifying as many frames, N, as needed to achieve the result.

GRID TYPE MNEMONIC	ABSCISSA	ORDINATE
LINEAR	linear	linear
SEMILG	linear	logarithmic
LOGLOG	logarithmic	logarithmic
LOGSEM	logarithmic	linear

FIGURE 9. AXIS DESIGNATION FOR VARIOUS PLOT MNEMONICS

Normally, there appears one set of grid lines per frame of graph paper. There may be times, however, when the user decides to place more than one grid, one above the other, on a frame of graph paper. A maximum of three grids can be placed on one frame of graph paper. A negative sign preceding the integer N denotes that multiple grids are desired. The number of coordinate names in parenthesis following N denotes the number of grids needed. Since a maximum of three grids are permitted on a frame of graph paper, up to three sets of coordinate names are permitted inside the parenthesis. A maximum of three plots can be placed on one grid if multiple plots are desired. A positive sign or lack of sign for the integer N denotes that multiple plots are required with the number of sets of coordinate names in parentheses indicating the number of plots needed.

XI and YI are the user-given MARSYAS names of the system INPUTS or OUTPUTS which are to be plotted. XI is the independent variable (the abscissa), TIME or other system OUTPUT or INPUT name, and YI the dependent variable (the ordinate). A maximum of three sets of variable names may appear within the parenthesis of a given PLOT statement if parentheses are used. See EXAMPLE (1). As many PLOT statements as required may be used within the Simulation Module so long as the total number of graphs processed does not exceed one hundred.

If a simple plot of one variable against another on one frame of graph paper is desired, the integer N and the parenthesis may be omitted from the plot statement. Then, up to one hundred coordinate names may be plotted using a single PLOT statement, as

```
PLOT:  grid type (TIME1,TIME2,CODE), X1,Y1, X2,Y2, ..., X100,Y100$
       mnemonic
```

(See EXAMPLE (2).)

Additionally, if the independent variable being plotted is time (CODE = 1), the abscissa name, TIME, need not be specified,thus further simplifying the PLOT statement. The PLOT statement would then become

```
PLOT:  grid type  (TIME1,TIME2),   Y1, Y2, Y3, ..., Y100$
       mnemonic
```

(See EXAMPLE (3).)

EXAMPLE (1)

The user decides to cross plot three OUTPUT variables with names V1, V2, V3 on a single sheet (one frame) of linear graph paper using three separate grids. The reference time interval will be from 0 to 36 seconds. The variables are to be plotted in the following manner: V2 vs. V1, V3 vs. V2 and V1 vs. V3.

The PLOT statement is coded as follows:

```
PLOT : LINEAR (0.0,36.0,2), -1 (V1,V2, V2,V3, V3,V1)$
```

Note that N=-1 indicates that one frame of graph paper is to be used. The negative sign indicates multiple grids are needed. The three coordinates inside the parenthesis indicate that three grids are required. The execution of the above statement in an actual MARSYAS program is shown in Figure 10.

EXAMPLE (2)

The user decides to cross plot OUTPUTS V2 and V1 with V1 the abscissa on a single frame of linear graph paper in the time interval 0 to 36 seconds. Since N=1, both N and the parenthesis may be omitted from the PLOT statement as shown below.

```
PLOT : LINEAR (0.0,36.0,2), V1, V2$
```

The execution of the above statement in an actual MARSYAS program is shown in Figure 11.

EXAMPLE (3)

A plot of system OUTPUT V3 against TIME is desired, on one frame of graph paper with a linear grid. The time interval is 0 to 36 seconds. Since N=1 and TIME is the independent variable, the abbreviated form of the PLOT statement is used. Figure 12 is an actual MARSYAS execution of the following statement:

```
PLOT : LINEAR(0.0, 36.0, 1), V3 $
```

PLOT (CON'T)

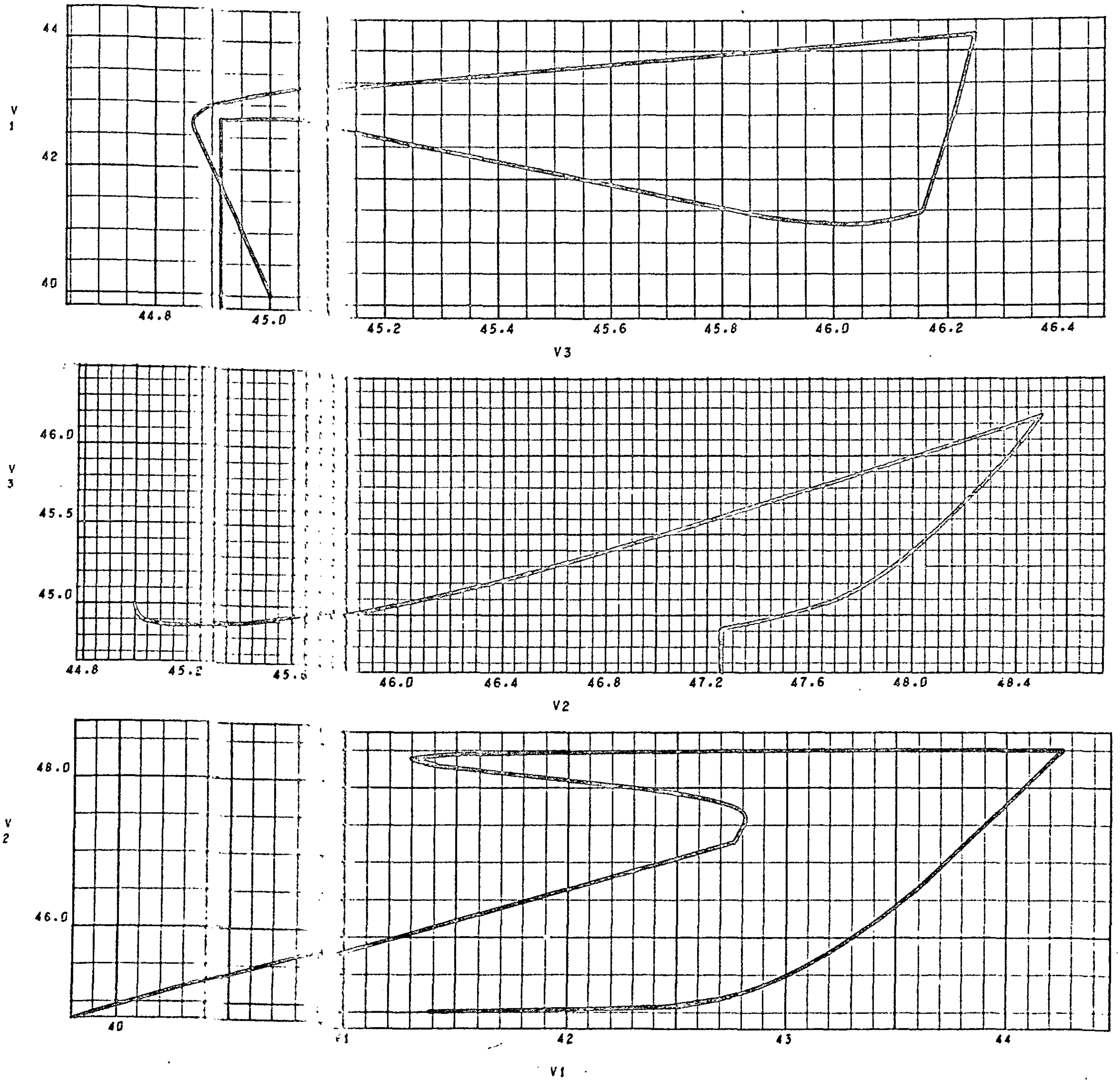


FIGURE 10 EXAMPLE OF MULTIPLE GRIDS ON ONE FRAME
PLOT: LINEAR (0.0,36.0,2), -1 (V1,V2,
V2,V3, V3, V1) $

PLOT (CON'T)

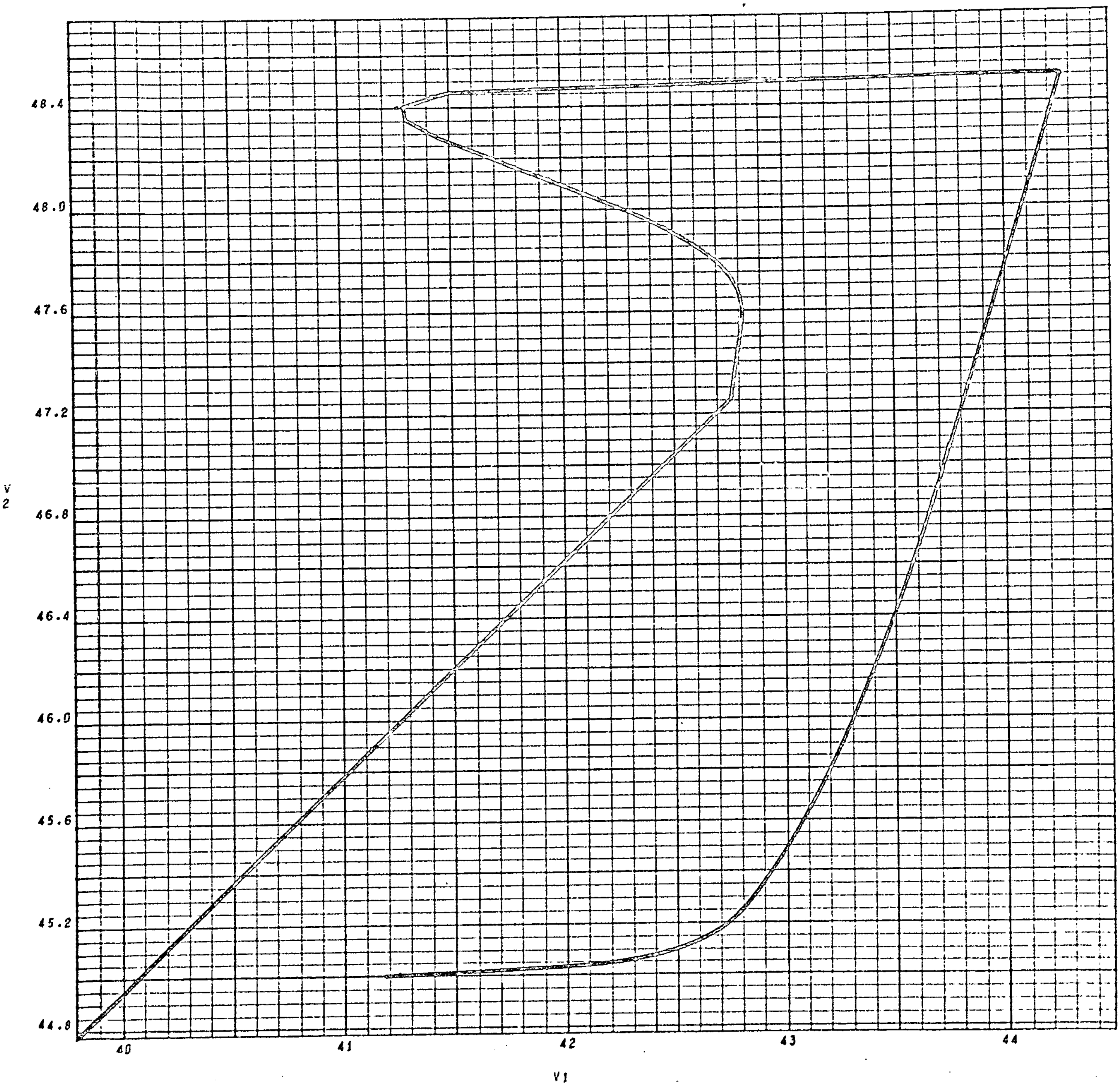


FIGURE 11 PLOT OF ONE VARIABLE AGAINST ANOTHER
PLOT: LINEAR (0.0,36.0,2), V1, V2 $

PLOT (CON'T)

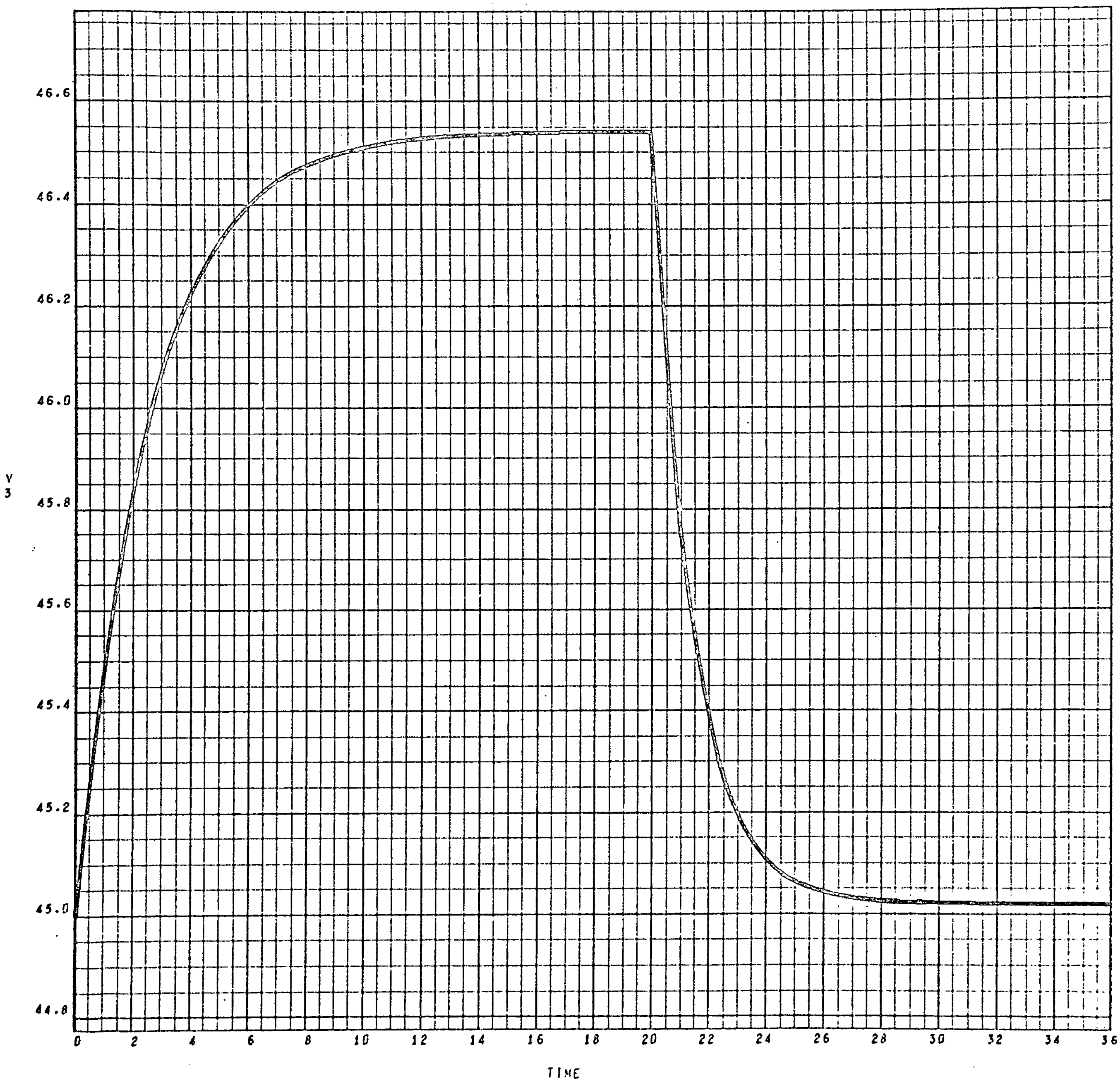


FIGURE 12 PLOT OF VARIABLE AGAINST TIME
PLOT: LINEAR(0.0,36.0,1),V3 $

II D. POST-PROCESSING MODULE

PRINT

FORMAT -

```
        (1)  PRINT:  name 1, name 2, etc. $
   or
        (2)  PRINT:  (output header title) $
```

COMMENTARY -

The PRINT statement specifies the names of the system input and/or output data to be printed. FORMAT (1) above yields the standard MARSYAS output print listing of six columns of data per page. The first printed column is always the independent variable TIME. FORMAT (2) allows the user to specify a title heading which will appear at the top of the output printed page.

The appropriate SAMPLE statement must be used in conjunction with PRINT to indicate the print step desired (see SAMPLE). There is a limit of one hundred input and output names that may be included in a given PRINT statement. The MARSYAS processor will automatically print the indicated variables in groups of six until the output list is completed.

EXAMPLE

(1) Assume the user's system has three inputs, INPUT1, INPUT2, and INPUT3, and two outputs, OUTPUT1 and OUTPUT2, which he wishes to be printed out, along with the header title FIRST TEST CASE. He decides to examine his input-output data at every fifth integration step. The PRINT and associated SAMPLE statements would be coded as follows:

```
PRINT:  (FIRST TEST CASE)$
PRINT:  INPUT1, INPUT2, INPUT3,
        OUTPUT1, OUTPUT2$
SAMPLE:  STEP, 5$
```

When using Runga-Kutta, if instead of examining the output at every fifth integration step the user decides to examine the data at every 0.10 seconds, the PRINT statement would remain unchanged and the SAMPLE statement becomes

```
SAMPLE:  TIME, 0.10$
```

SAMPLE

FORMAT -

```
        (1)  SAMPLE:  STEP, integer number$
    or
        (2)  SAMPLE:  TIME, real number$
```

COMMENTARY -

The SAMPLE statement is used in conjunction with the PRINT statement to specify the output printing interval. The print interval may be expressed in terms of the independent variable TIME or the integration STEP size. If TIME is selected, the step size is taken in approximately equal increments of simulation time as specified by the real number in Format (2) above.

EXAMPLE

```
         SAMPLE:  STEP, 5$

    or

         SAMPLE:  TIME, .05$
```

In the first example, the MARSYAS processor will list the output(s) at every fifth integration step, whereas the second will list the output(s) in time increments of 0.05 units.

SECTION III

SOFTWARE STRUCTURE OF MARSYAS

III. SOFTWARE STRUCTURE OF MARSYAS

A. Introduction

The prime objective of the MARSYAS software is to transform a MARSYAS program, which describes a model and specifies the simulation, into a FORTRAN program that contains the arrays and subroutines for the numerical solution of the various matrix equations. The MARSYAS software is thus a precompiler which compiles the MARSYAS language statements into a set of FORTRAN programs called the Object Program which is then executed by the computer.

B. Overview of MARSYAS Software System

MARSYAS was originally designed for use on a time-sharing machine such as the Univac 1108 computer. The software permits several users to access MARSYAS simultaneously from remote stations. An overview of the MARSYAS software system is shown in Figure 13. The user's block diagram and/or equations are coded in a MARSYAS program consisting of Description, Modification, Simulation and Post Processing Module statements which are punched on cards and fed to the computer. All Program Modules with the exception of the Post Processing Module feed into the Simulation Program Module which ultimately generates the FORTRAN Object Program. This, in conjunction with the Library of Standard Elements and Excitation Functions, is compiled with the FORTRAN Object Program and executed.

III B. OVERVIEW OF MARSYAS SOFTWARE SYSTEM

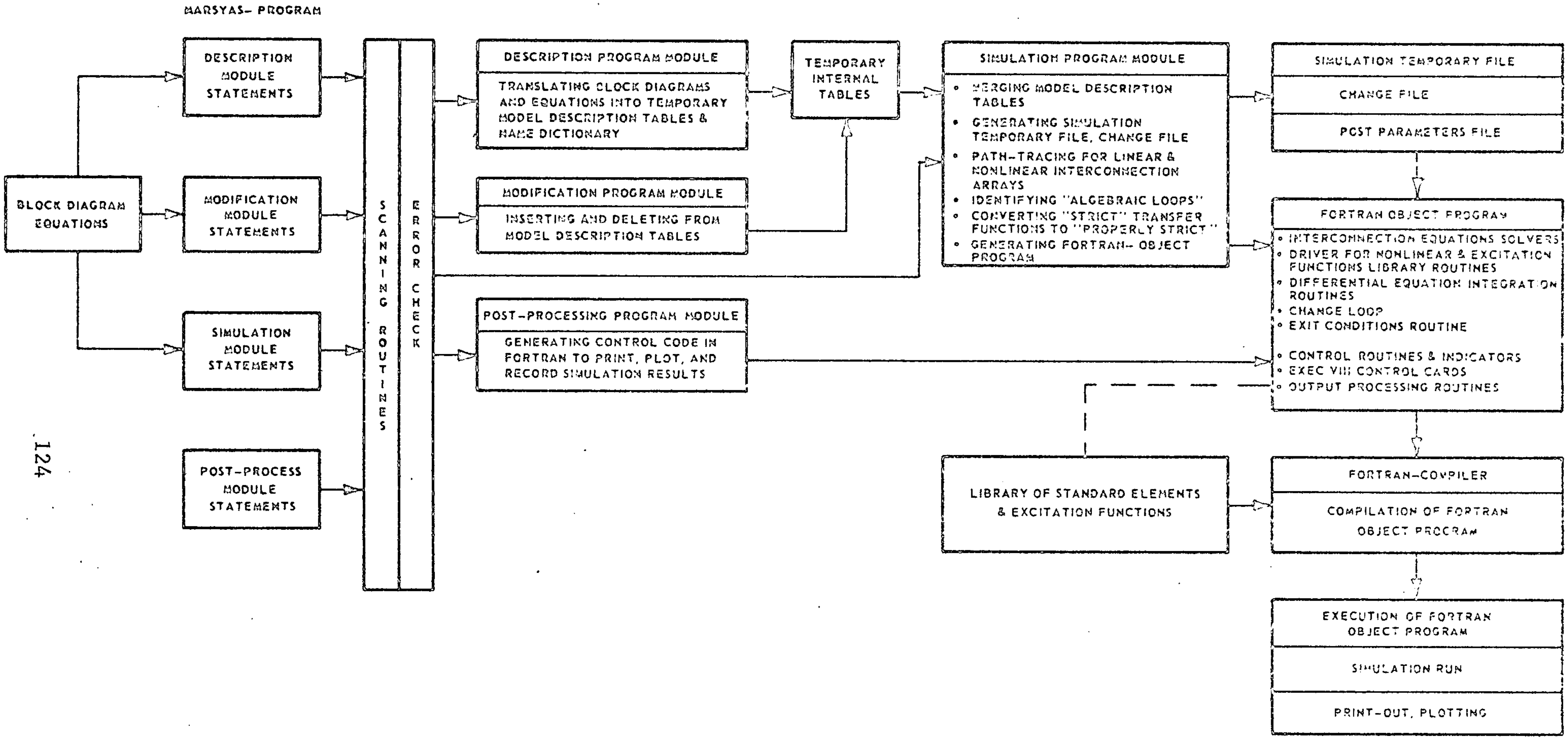

MARSYAS- PROGRAM
BLOCK DIAGRAM EQUATIONS
DESCRIPTION MODULE STATEMENTS
MODIFICATION MODULE STATEMENTS
SIMULATION MODULE STATEMENTS
POST-PROCESS MODULE STATEMENTS
SCANNING ROUTINES
ERROR CHECK
DESCRIPTION PROGRAM MODULE
TRANSLATING BLOCK DIAGRAMS AND EQUATIONS INTO TEMPORARY MODEL DESCRIPTION TABLES & NAME DICTIONARY
MODIFICATION PROGRAM MODULE
INSERTING AND DELETING FROM MODEL DESCRIPTION TABLES
POST-PROCESSING PROGRAM MODULE
GENERATING CONTROL CODE IN FORTRAN TO PRINT, PLOT, AND RECORD SIMULATION RESULTS
TEMPORARY INTERNAL TABLES
SIMULATION PROGRAM MODULE
• MERGING MODEL DESCRIPTION TABLES
• GENERATING SIMULATION TEMPORARY FILE, CHANGE FILE
• PATH-TRACING FOR LINEAR & NONLINEAR INTERCONNECTION ARRAYS
• IDENTIFYING "ALGEBRAIC LOOPS"
• CONVERTING "STRICT" TRANSFER FUNCTIONS TO "PROPERLY STRICT"
• GENERATING FORTRAN- OBJECT PROGRAM
SIMULATION TEMPORARY FILE
CHANGE FILE
POST PARAMETERS FILE
FORTRAN OBJECT PROGRAM
• INTERCONNECTION EQUATIONS SOLVERS
• DRIVER FOR NONLINEAR & EXCITATION FUNCTIONS LIBRARY ROUTINES
• DIFFERENTIAL EQUATION INTEGRATION ROUTINES
• CHANGE LOOP
• EXIT CONDITIONS ROUTINE
• CONTROL ROUTINES & INDICATORS
• EXEC VIII CONTROL CARDS
• OUTPUT PROCESSING ROUTINES
LIBRARY OF STANDARD ELEMENTS & EXCITATION FUNCTIONS
FORTRAN-COMPILER
COMPILATION OF FORTRAN OBJECT PROGRAM
EXECUTION OF FORTRAN OBJECT PROGRAM
SIMULATION RUN
PRINT-OUT, PLOTTING

FIGURE 13. OVERVIEW OF MARSYAS SOFTWARE SYSTEM

III C. LIMITATIONS OF MARSYAS

C. Limitations of MARSYAS

Within the complete MARSYAS program, there is one main MODEL and up to 99 SUBMODELS permitted. Each SUBMODEL, of course, has all the characteristics of a MODEL. Models and Submodels are each limited to a maximum of 50 inputs, 50 outputs, 300 elements with restrictions as noted below and 500 connections.

For the main model and all its submodels, the following restrictions apply:

There may be a maximum of:

350 constant multipliers

150 transfer functions, including integrators

If N_i is the order of each transfer function, M is the total number of transfer functions and $L_i = N_i + 1$, then

$$\sum_{i=1}^{M} L_i \leq 400$$

400 nonlinear elements

4000 parameters

600 inputs to nonlinear elements

400 outputs from nonlinear elements

III C. LIMITATIONS OF MARSYAS

For the Simulation Module, the following restrictions apply:

There may be a maximum of:

125 EXCITE statements

600 STOP IF and TERMINATE IF statement words

30 simulation CHANGE cycles

400 parameter changes per cycle

A "word" is the number of characters divided by six.

The following restrictions apply to the Simulation Module:

There may be a maximum of:

100 plot terminals

100 graphs

100 print terminals

350 Hollerith words for PRINT title statement

SECTION IV

MATHEMATICAL STRUCTURE OF MARSYAS

IV. MATHEMATICAL STRUCTURE OF MARSYAS

A. The State Space Approach

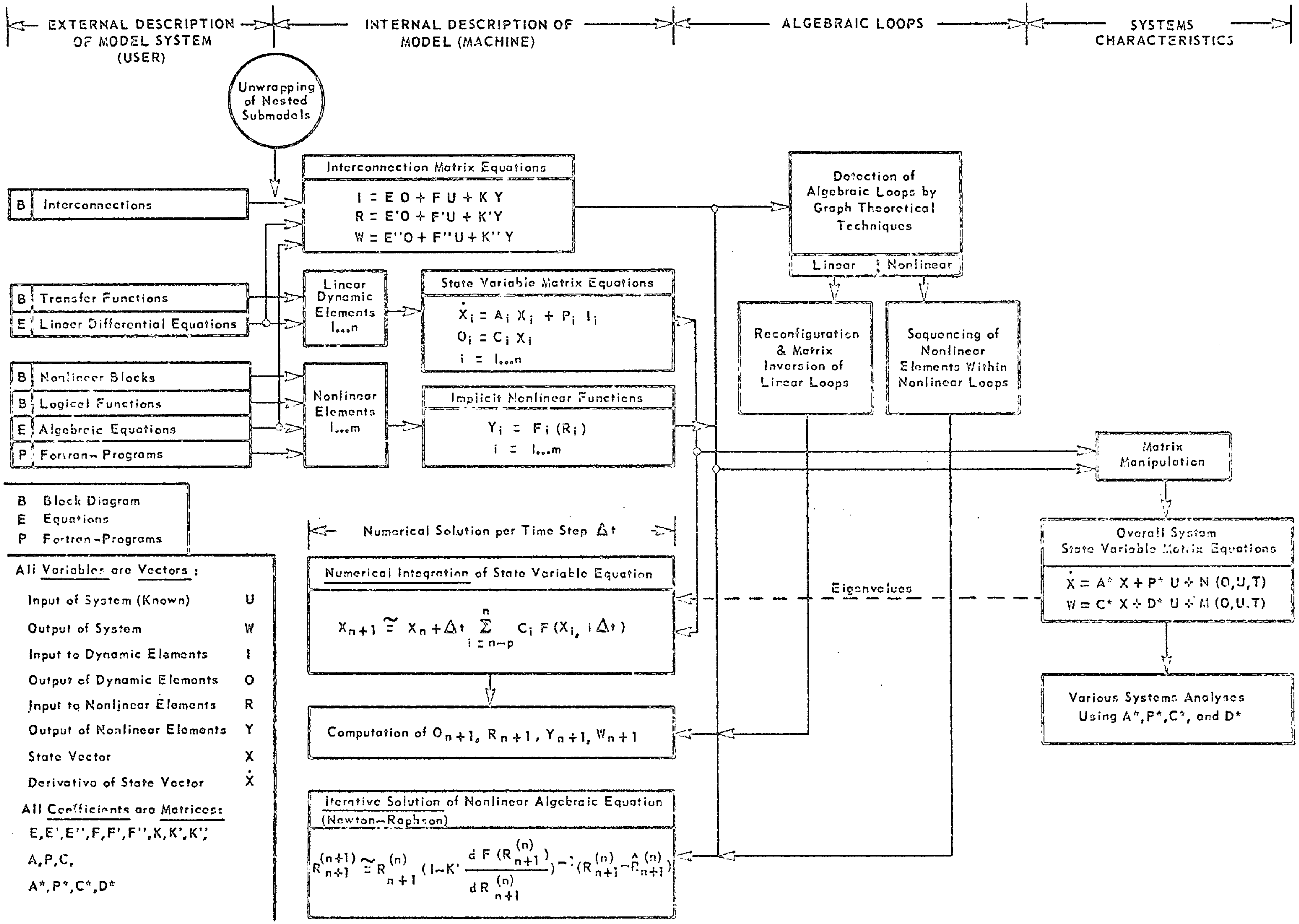
MARSYAS has a basic mathematical structure which incorporates the latest and most up-to-date mathematics associated with modern control theory. The state space approach is used throughout, and directed graph theory is used to detect and unwind algebraic loops automatically. These methods have considerable advantages over older, classical methods.

There are no known methods that will solve every set of nonlinear algebraic equations. Therefore, while many nonlinear loops can be solved by MARSYAS, some will arise which will not converge. However, most all cases of linear loops can be handled. An overview of the mathematical process which is the foundation of MARSYAS appears in Figure 14.

B. Numerical Integration Techniques

The FORTRAN Object Program generated by the MARSYAS processor uses precoded subroutines to control the numerical integration and detect discontinuities which may exist in the model. Since serious errors may arise when integrating through discontinuities, the MARSYAS system automatically switches integration methods when a discontinuity is detected. It integrates through the discontinuity using Euler's method. Euler's method, being of first order, will yield more accurate results at discontinuities than higher order methods. Once past the discontinuity, the original integration scheme is resumed.

IVA. THE STATE SPACE APPROACH



EXTERNAL DESCRIPTION OF MODEL SYSTEM (USER)
INTERNAL DESCRIPTION OF MODEL (MACHINE)
ALGEBRAIC LOOPS
SYSTEMS CHARACTERISTICS
Unwrapping of Nested Submodels
B Interconnections
Interconnection Matrix Equations
$I = EO + FU + KY$
$R = E'O + F'U + K'Y$
$W = E''O + F''U + K''Y$
B Transfer Functions
E Linear Differential Equations
Linear Dynamic Elements 1...n
State Variable Matrix Equations
$\dot{X}_i = A_i X_i + P_i I_i$
$O_i = C_i X_i$
$i = 1 \ldots n$
B Nonlinear Blocks
B Logical Functions
E Algebraic Equations
P Fortran-Programs
Nonlinear Elements 1...m
Implicit Nonlinear Functions
$Y_i = F_i(R_i)$
$i = 1 \ldots m$
Detection of Algebraic Loops by Graph Theoretical Techniques
Linear
Nonlinear
Reconfiguration & Matrix Inversion of Linear Loops
Sequencing of Nonlinear Elements Within Nonlinear Loops
Matrix Manipulation
Overall System State Variable Matrix Equations
$\dot{X} = A^* X + P^* U + N(0,U,T)$
$W = C^* X + D^* U + M(0,U,T)$
Various Systems Analyses Using A*, P*, C*, and D*
Eigenvalues
Numerical Solution per Time Step Δt
Numerical Integration of State Variable Equation
$X_{n+1} \cong X_n + \Delta t \sum_{i=n-p}^{n} c_i F(X_i, i\Delta t)$
Computation of $O_{n+1}, R_{n+1}, Y_{n+1}, W_{n+1}$
Iterative Solution of Nonlinear Algebraic Equation (Newton-Raphson)
$R_{n+1}^{(n+1)} \cong R_{n+1}^{(n)} (I - K' \frac{dF(R_{n+1}^{(n)})}{dR_{n+1}^{(n)}})^{-1} (R_{n+1}^{(n)} - \hat{R}_{n+1}^{(n)})$
B Block Diagram
E Equations
P Fortran-Programs
All Variables are Vectors:
Input of System (Known) U
Output of System W
Input to Dynamic Elements I
Output of Dynamic Elements O
Input to Nonlinear Elements R
Output of Nonlinear Elements Y
State Vector X
Derivative of State Vector $\dot{X}$
All Coefficients are Matrices:
E, E', E'', F, F', F'', K, K', K'',
A, P, C,
A*, P*, C*, D*

FIGURE 14. OVERVIEW OF MATHEMATICAL PROCESS

IV B. NUMERICAL INTEGRATION TECHNIQUES

1. Runga-Kutta Method

The Runga-Kutta-Gill numerical integration scheme used by MARSYAS is the fourth-order single step method which approximates the solution of the state equation $\dot{X}(t) = F(X(t), t)$ at $t = (n+1)h$ by

$$X_{n+1} = X_n + \frac{h}{6} (K_1 + 2\ (1 - \sqrt{.5}\ )\ K_2 + 2\ (1 + \sqrt{.5})\ K_3 + K_4\ )$$

where

$$K_1 = F(X_n,\ t_n)$$

$$K_2 = F(X_n + K_1 h/2,\ \ t_n + h/2)$$

$$K_3 = F(X_n - (\tfrac{1}{2} - \sqrt{.5}\ )\ K_1 h + (1 - \sqrt{.5}\ )\ K_2 h,\ \ t_n + h/2)$$

$$K_4 = F(X_n - \sqrt{.5}\ \ K_2 h + (1 + \sqrt{.5})\ K_3 h,\ \ t_n + h)$$

and h is the step size specified by the user.

2. Adams-Bashforth Method

MARSYAS actually uses the Adams-Bashforth predictor with the Adams-Moulton corrector in solving the state equations. These multi-step equations approximate the solution of the state equation as follows:

Adams-Bashforth predictor:

$$x_{n+1} = x_n + \frac{h}{24} (55\mathring{x}_n - 59\mathring{x}_{n-1} + 37\mathring{x}_{n-2} - 9\mathring{x}_{n-3})$$

Adams-Moulton corrector:

$$x_{n+1} = x_n + \frac{h}{24} (9\dot{x}_{n+1} + 19\dot{x}_n - 5\dot{x}_{n-1} + \dot{x}_{n-2})$$

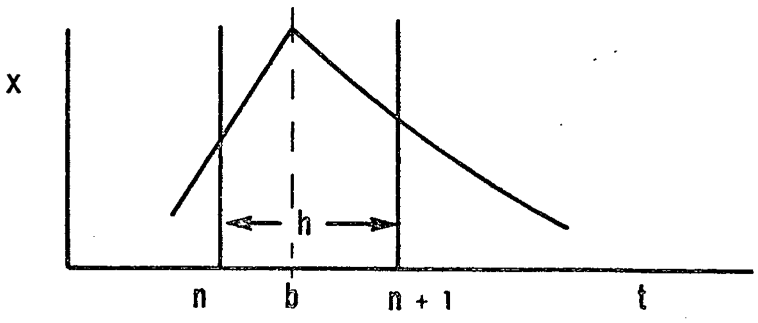
These predictor-corrector equations have a definite advantage with respect to solution time over the Runga-Kutta method in that at most two derivatives must be calculated at each time step whereas the Runga-Kutta method requires four derivatives. The automatic error estimation procedure inherent in the predictor-corrector scheme allows the predictor to select the optimum step length that satisfies the built-in error criterion, thus further speeding up the solution time.

3. Euler's Method

Euler's method is implemented automatically whenever the MARSYAS processor detects the presence of a discontinuity. Euler's is essentially a first-order single step method of the form

$$X(n+1) = X(n) + \dot{X}(n)\ h$$

The integration step size h is measured from the end of the last full integration step before the discontinuity and extends past the discontinuity as shown in the sketch below.



Thus, if a discontinuity occurs at time t=b, then the integration method will change to the Euler method at $t=t_n$ and back again at $t=t_{n+1}$ to the method of integration used before the discontinuity was encountered.

If the user has several discontinuities in his system, it is recommended that Butcher's method or the Runga-Kutta method of integration be specified (using the INTEGRATE statement) since the solution time will often be less than when using Adams-Bashforth.

Should the need arise, the user can disable the Euler-interrupt so that the selected integration scheme can proceed uninterrupted. Additionally Euler's method can be used for the entire simulation if desired. See INTEGRATE statement in Section II C.

4. <u>Butcher's Method</u>

The Butcher's numerical integration method use in MARSYAS is a fifth-order, six-stage Runga-Kutta scheme which approximates the solution of the state equations as follows:

$$x_{n+1} = x_n + \frac{h}{90.0}(7K_1 + 32K_2 + 12K_4 + 32K_5 + 7K_6)$$

where

$$K_1 = F(x_n, t_n)$$

$$K_2 = F(x_n + K_1 h/4,\ t_n + h/4)$$

$$K_3 = F(x_n + k_1 h/8 + K_2 h/8,\ t + h/4)$$

$$K_4 = F(x_n - K_2 h/2 + K_3 h,\ t + h/2)$$

$$K_5 = F(x_n + 3/16K_1h + 9/16K_4h,\ t + 3/4h)$$

$$K_6 = F(x_n - 3/7K_1h + 2/7K_2h + 12/7K_3h - 12/7K_4h + 8/7K_5h,\ t + h)$$

and h is the step size specified by the user.

5. Sarafyan Method

The Sarafyan variable step method is essentially an embedding of Runga-Kutta formulas to achieve step size control. The Sarafyan method used in MARSYAS is essentially a fifth-order, six-stage Runga-Kutta formula (Butcher's) with an imbedded second-order Runga-Kutta formula used to determine the appropriate step size. The fifth-order, six-stage formula is the same as that used in the Butcher method above and the second-order formula is given by

$$x_{n+1} = x_n + h(-K_1 + 2K_2)$$

where

$$K_1 = F(x_n, t_n)$$

$$K_2 = F(x_n + K_1h/4,\ t_n + h/4)$$

and h is the step size.

The higher and lower order Runga-Kutta schemes are compared so that an estimate of the accuracy can be made. Then a judgment is made on the current value of h being used. If h is too large, it is halved. If too small, it is doubled. If just right, it is left as it is.

Computation accuracy using the Sarafyan method is high and computation speed fast. Depending upon problem being simulated, computation speed can be increased significantly over fixed-step methods.

IV B. NUMERICAL INTEGRATION TECHNIQUES

Other numerical integration methods will soon be added to the MARSYAS language allowing the user greater flexibility in solving his problems.

IV C. SOLUTION SCHEMES FOR DIFFERENTIAL EQUATIONS

C. Solution Schemes for Differential Equations

When using the EQUATION option in solving a set of ordinary differential equations, the user must check to see that a solution exists. In other words, it must be possible to construct a "solution scheme" for the system of equations. A solution scheme is an assignment of variables to the equations such that the following three conditions hold:

1. There must be one independent equation for each unknown variable.
2. The highest order of each variable derivative in the set of equations can be "assigned" to one particular equation of the set. No two variable derivatives can be assigned to the same equation.
3. The order of the variable derivative assigned to a particular equation must be the highest order for that variable as it appears in the system of equations.

EXAMPLE (1)

Determine if a solution scheme exists for the following set of differential equations:

1) $a\ddot{X} + b\dot{Y}X + \ddot{Z} = 0$

2) $\ddot{Y} + \dot{Z} = 2XY$

3) $\ddot{Z} + \dot{X} = 0$

In the above set of equations in three unknowns, X, Y and Z, the three highest order derivatives are $\ddot{X}$, $\ddot{Y}$, and $\ddot{Z}$. They can be assigned to equations 1), 2) and 3), respectively. Hence a solution scheme exists for the system of equations.

IV C. SOLUTION SCHEMES FOR DIFFERENTIAL EQUATIONS

EXAMPLE (2)

Determine if a solution scheme exists for the following set of differential equations:

1) $\ddot{X} + \dot{Y} + XY = \mathrm{SIN}\ (\dot{X} - \dot{Y})$

2) $X\dddot{Y} - XY = 0$

These are two equations in two unknowns, X and Y, whose highest order derivatives are $\ddot{X}$ and $\dddot{Y}$. The derivative $\ddot{X}$ can be assigned to equation 1) and $\dddot{Y}$ can be assigned to equation 2). Hence, a solution scheme exists for this set of equations.

EXAMPLE (3)

Determine if a solution scheme exists for the following set of differential equations:

1) $\ddot{X} + \dot{Y} + XY = \mathrm{TAN}\ (3X^2 + 2Y)$

2) $\dot{X}Y - X = 0$

The above two equations in two unknowns X and Y have highest order derivatives $\ddot{X}$ and $\dot{Y}$. Since we cannot assign two variable derivatives to the same equation, no solution scheme exists for this set of equations.

SECTION V

OPERATION OF THE MARSYAS SYSTEM

V. OPERATION OF THE MARSYAS SYSTEM

A. Deck Setup for MARSYAS Operation on the Univac 1108 Computer Under EXEC VIII

The following computer control cards are required to run a MARSYAS program on the Univac 1108.

```
@RUN,
@ASG,X  MARSYAS*MARSYAS
@ASG,T  1,F
@DATA,I   1
MARSYAS DECK
@END
@ADD   MARSYAS*MARSYAS.
@FIN
```

B. MARSYAS Diagnostics

There are two distinct types of error diagnostics in the MARSYAS system, Statement Error Messages and Module Error Messages. The first type deals with errors encountered in coding a given statement. When an error of this type is detected, the processing of the MARSYAS statement is terminated with an error message. That portion of the statement in error is indicated with a series of backward slashes (\ ) under the printed out statement. An error diagnostic is then printed out to aid the user in correcting the problem. Since the MARSYAS processing terminates after processing a group of errors, the user should recheck his program to be sure no more errors are present in the non-processed portion of his program.

V B. MARSYAS DIAGNOSTICS

The second type of error diagnostic will appear when the user violates the structural pattern of MARSYAS. Examples of this are if the user should inadvertently place the Post Processing Module before the Simulation Module, or if he forgets to place an END$ statement at the end of a Module and so on. If a system is incompletely defined by leaving out elements, connections, inputs, outputs, excitations, and so on, the errors will be grouped at the end of the appropriate Module under the heading "MODULE ERROR SUMMARY."

When Module Error Summary diagnostics appear in conjunction with coding errors, they can usually be ignored until all coding errors have been corrected. This is due to the fact that the Module Error Summary diagnostics were generated as a result of the incomplete processing of the statement containing the coding error.

A complete list of the MARSYAS diagnostics appears on the following pages.

V B. MARSYAS DIAGNOSTICS

Statement Error Messages

NUMBER OF LEFT AND RIGHT PARENTHESES DOES NOT MATCH.
Any parentheses that are opened must be closed.

IMPROPER PUNCTUATION. A COMMA IS EXPECTED.

IMPROPER PUNCTUATION. A $ SIGN IS EXPECTED.
The statement has not been completed properly.

IMPROPER PUNCTUATION. A COMMA OR $ SIGN IS EXPECTED.

IMPROPER PUNCTUATION. A LEFT PARENTHESIS IS EXPECTED.

IMPROPER PUNCTUATION USED IN TERMINATING SEQUENCE.

EXPRESSION CONTAINS TOO MANY RIGHT PARENTHESES.

IMPROPER PUNCTUATION. A SEMICOLON IS EXPECTED.

WORD EXCEEDS 36-CHARACTER LENGTH LIMIT.

WORD BEGINS WITH IMPROPER FIRST CHARACTER.
A MARSYAS word must begin with one of the characters ABCDEFGHIJKLMNOPQRSTUVWXYZ0123 456789 . Words which are not numbers must begin with an alphabet letter.

WORD CONTAINS AN INVALID CHARACTER.

WORD CONTAINS IMPROPER NUMBER OF CHARACTERS.

OPERATOR WORD IS NOT A MARSYAS OPERATOR.

OPERATOR SHOULD BE INPUTS (OUTPUTS).

OPERATOR WORD MAY NOT BE USED MORE THAN ONCE IN A GIVEN MODULE.

EXPECTED OPERATOR NOT CORRECT.

FUNCTIONAL DATA BASE PASS WORD INCORRECT OR MISSING.

EQUATIONS MUST HAVE TWO SIDES.

INVALID OPERATOR IN PRESENT CONTEXT.

ONLY VARIABLES (NOT TIME) MAY BE DIFFERENTIATED.

MNEMONIC WORD IS NOT A MARSYAS MNEMONIC.

INTEGRATION MNEMONIC (CONSTRAINT) IS NOT A MARSYAS MNEMONIC (CONSTRAINT).

MNEMONIC IS NOT A MARSYAS ARITHMETIC RELATION.
The MARSYAS arithmetic relational operators are EQ, NE, GT, LT, GE, and LE.

V B. MARSYAS DIAGNOSTICS

Statement Error Messages

WORD IS NOT A MARSYAS LOGICAL OPERATOR.
The logical operators are AND, OR, and NOT.

IMPROPER GRID TYPE MNEMONIC.

IMPROPER SOLUTION MNEMONIC

IMPROPER SAMPLE MNEMONIC.

IMPROPER NUMBER OF PARAMETERS GIVEN FOR ELEMENT.

IMPROPER PUNCTUATION. A COLON IS EXPECTED.

IMPROPER NAME TYPE USED AS ARGUMENT.
The statement requires a different type of name in the argument position.

ARGUMENT SHOULD NOT BE DEFINED IN NAMING STATEMENT.
The "old name" in a NAMING statement argument list and the "new element" in a SUBSTITUTE statement should not be names defined in a NAMING statement.

THE ARGUMENT HAS BEEN USED PREVIOUSLY IN THIS OPERATOR CONTEXT.
Certain operations may be performed on a given argument only once.

TIMESTEP CONSTRAINT SHOULD BE SPECIFIED.
If the Runga-Kutta integration procedure is used, an integration step size must be specified.

ORDER OF A TRANSFER FUNCTION MAY NOT BE VARIED.

MORE THAN ONE WRITE PARAMETER HAS BEEN SPECIFIED IN THE MODULE.
A POST-PROCESSING module may write output at only one sample interval.

CANNOT CHANGE GAINS WHEN THERE ARE LINEAR LOOPS.

IMPROPER NUMBER OF INITIAL CONDITIONS.

IMPROPER ELEMENT TYPE IN SUBSTITUTE STATEMENT.

EXPECTED ARGUMENT IS INCORRECT TYPE.

IMPROPER NUMBER OF ARGUMENTS.

NAME HAS BEEN DEFINED MORE THAN ONCE.
A MARSYAS name should be defined in only one statement.

NAME IS USED INCONSISTENTLY.
Characteristics which are implied by previous uses of the name are inconsistent with the present use of the name.

V B. MARSYAS DIAGNOSTICS

Statement Error Messages

PARAMETER HAS BEEN GIVEN TWO VALUES.

NAME IS NOT IN MODEL DICTIONARY.

MODEL NAME ALREADY EXISTS IN FUNCTIONAL DATA BASE.

MODEL DOES NOT EXIST IN SYSTEM.

SOURCE CODE STATEMENTS ARE MISSING.
The program is incomplete.

SOURCE PROGRAM CONTAINS TOO MANY STATEMENTS.
A MARSYAS deck may contain 99,999 statements.

THE SIMULATED SYSTEM IS TOO LARGE.
The MARSYAS system allows for 2^{18}-1 distinct sequence numbers (objects).

MODEL IS NOT INCLUDED IN THE SIMULATED SYSTEM.

FIRST STATEMENT OF PROGRAM SHOULD BE A MARSYAS BEGIN.

PROGRAM NAME IS INCORRECT.

MODEL INCORRECTLY RECORDED IN DESCRIPTION TEMPORARY FILE.

FUNCTION WORD IS NOT A MARSYAS FUNCTION.

CONTINUATION NUMBER REQUESTED NOT ON CONTINUATION FILE.

END STATEMENT SENTINEL MUST MATCH BEGIN DECKNAME.

A TERMINAL OF AN INCLUDED MODEL HAS NOT BEEN CONNECTED.

FUNCTION (MATH REFERENCE) HAS INCORRECT NUMBER OF ARGUMENTS.

USER DEFINED FUNCTION NOT ALLOWED AS A MATH REFERENCE.

NUMERIC FIELD CONTAINS AN IMPROPER CHARACTER.
Besides the digits, a numeric field may contain a sign, "E" or ".", and only these characters.

THE VALUE OF THE NUMBER IS TOO LARGE OR TOO SMALL.

IMPROPER USE OF NUMERIC "E" FORMAT.

THE NUMBER USED IS NOT PERMISSIBLE.
Certain numbers must be integers (non-negative, positive, non-negative integers, etc.). Element subscripts and parameter numbers should be positive integers less than 2^{12}.

V B. MARSYAS DIAGNOSTICS

Statement Error Messages

LENGTH OF PRINTED LINE EXCEEDS THE LINE LIMIT.
An output line may be at most 132 characters long.

NUMBER IMPROPERLY FORMATTED.

A TRANSFER FUNCTION PARAMETER B(N) IS ZERO.

MODULE BEGINS WITH IMPROPER OPERATOR.

THE OPERATOR IS NOT PERMISSIBLE IN THE PRESENT MODULE.

AN END STATEMENT MUST PRECEDE A NEW MODULE.
A module in a MARSYAS program is completed with an END statement. A new module may begin after the END statement.

THE SEQUENCE OF MODULES INVOLVING THE PRESENT MODULE IS INVALID.
The order of modules for a given model must be chronological, e.g., a model must be described before it is modified or simulated.

THE MODULE CONTAINS AN IMPROPER COMBINATION OF IF STATEMENTS.

SYSTEM HAS NOT BEEN PREVIOUSLY SIMULATED.

SYSTEM HAS BEEN PREVIOUSLY SIMULATED OR CONTINUED.

THE CONSTRAINT ILLEGAL IN PRESENT MODE.

V B. MARSYAS DIAGNOSTICS

Module Error Messages

IMPLIED INTEGRATION AND DEFAULT INTEGRATION METHODS DO NOT AGREE.

MINIMUM TIME STEP LESS THAN STEP SIZE.

NO EXCITATION FUNCTIONS HAVE BEEN SPECIFIED FOR THE FOLLOWING INPUT TERMINALS:

EXCEEDING SYSTEM CONSTRAINTS CREATED AN ABORT SITUATION.

NO MODEL OUTPUT TERMINALS HAVE BEEN SPECIFIED FOR THE MODEL.

NO ELEMENTS HAVE BEEN SPECIFIED FOR THE MODEL.

THE MODULE CONTAINS NO PRINT OR PLOT STATEMENTS.

A TRANSFER FUNCTION ORDER HAS BEEN CHANGED ILLEGALLY.

THE FOLLOWING MARSYAS NAMES HAVE NOT BEEN PROPERLY DEFINED IN THE ABOVE MODULE:

THE FOLLOWING CONNECT STATEMENTS CONTAIN IMPROPER OR AMBIGUOUS CONNECTIONS:

THE FOLLOWING TERMINALS HAVE BEEN CONNECTED AN IMPROPER NUMBER OF TIMES:

THE INITIAL CONDITIONS FOR THE FOLLOWING TRANSFER FUNCTIONS ARE INVALID:

SECTION VI

EXAMPLES OF THE USE OF MARSYAS

VI. EXAMPLES OF THE USE OF MARSYAS

A. Mechanical Extension Device - Example A

An analysis of the motion for a mechanical extension device has resulted in the mathematical model presented in Figure 15. The motion of the device is represented by the quantities θ_1, θ_2, θ_3 and θ_4, which are the required outputs. The system input, δ, is subjected to a ramp input with slope 25.136 units.

The system differential equation, as shown in Figure 15, is of the matrix-vector form

$$A(\theta)\ddot{\theta} + B(\theta)(\dot{\theta})^2 + C\theta + DU = 0 \tag{1}$$

This system of equations is solved for $\ddot{\theta}$ as

$$\ddot{\theta} = A^{-1}(\theta)\left[-B(\theta)(\dot{\theta})^2 - C\theta - DU\right] \tag{2}$$

There are two methods of solving Equation (2) using MARSYAS: the block diagram method and the EQUATION method. The block diagram solution of Equation (2), shown in Figure 16, makes use of specially constructed DEVICEs to perform the required matrix multiplications. The EQUATION approach solves a collection of scalar differential equations in conjunction with one DEVICE to perform the multiplication by A^{-1}. The computer printout of the MARSYAS programs using each method is shown

VI A. MECHANICAL EXTENSION DEVICE – EXAMPLE A

$$\begin{bmatrix} A \end{bmatrix} \begin{pmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \\ \ddot{\theta}_4 \end{pmatrix} + \begin{bmatrix} B \end{bmatrix} \begin{pmatrix} (\dot{\theta}_1)^2 \\ (\dot{\theta}_2)^2 \\ (\dot{\theta}_3)^2 \\ (\dot{\theta}_4)^2 \end{pmatrix} + \begin{bmatrix} C \end{bmatrix} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{pmatrix} + \begin{bmatrix} D \end{bmatrix} \begin{pmatrix} \theta_0 \\ \delta \end{pmatrix} = 0$$

WHERE

$$A = \begin{bmatrix} (a-i_1) & b\cos(\theta_2-\theta_1) & c\cos(\theta_3-\theta_1) & d\cos(\theta_4-\theta_1) \\ b\cos(\theta_2-\theta_1) & (b-i_2) & c\cos(\theta_3-\theta_2) & d\cos(\theta_4-\theta_2) \\ c\cos(\theta_3-\theta_1) & c\cos(\theta_3-\theta_2) & (c-i_3) & d\cos(\theta_4-\theta_3) \\ d\cos(\theta_4-\theta_1) & d\cos(\theta_4-\theta_2) & d\cos(\theta_4-\theta_3) & (d-i_4) \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & -b\sin(\theta_2-\theta_1) & -c\sin(\theta_3-\theta_1) & -d\sin(\theta_4-\theta_1) \\ b\sin(\theta_2-\theta_1) & 0 & -c\sin(\theta_3-\theta_2) & -d\sin(\theta_4-\theta_2) \\ c\sin(\theta_3-\theta_1) & c\sin(\theta_3-\theta_2) & 0 & -d\sin(\theta_4-\theta_3) \\ d\sin(\theta_4-\theta_1) & d\sin(\theta_4-\theta_2) & d\sin(\theta_4-\theta_3) & 0 \end{bmatrix}$$

$$C = -\frac{1}{L_2} \begin{bmatrix} 2k_1 & -k_1 & 0 & 0 \\ -k_2 & 2k_2 & -k_2 & 0 \\ 0 & -k_3 & 2k_3 & -k_3 \\ 0 & 0 & -k_4 & k_4 \end{bmatrix}$$

$$D = -\frac{1}{L_2} \begin{bmatrix} -k_1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -k_4 \end{bmatrix}$$

FIGURE 15 MATHEMATICAL MODEL OF MECHANICAL EXTENSION DEVICE

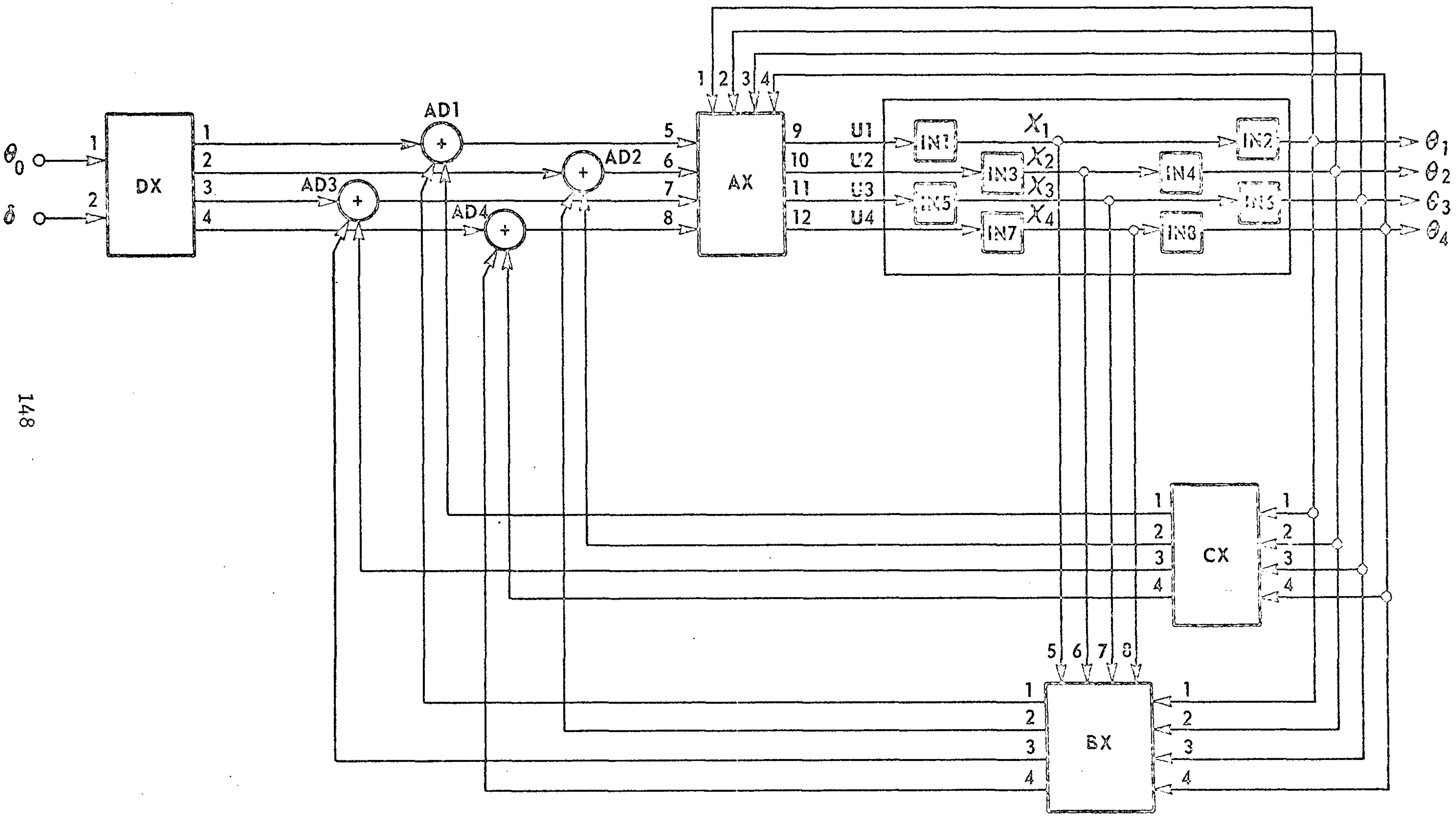
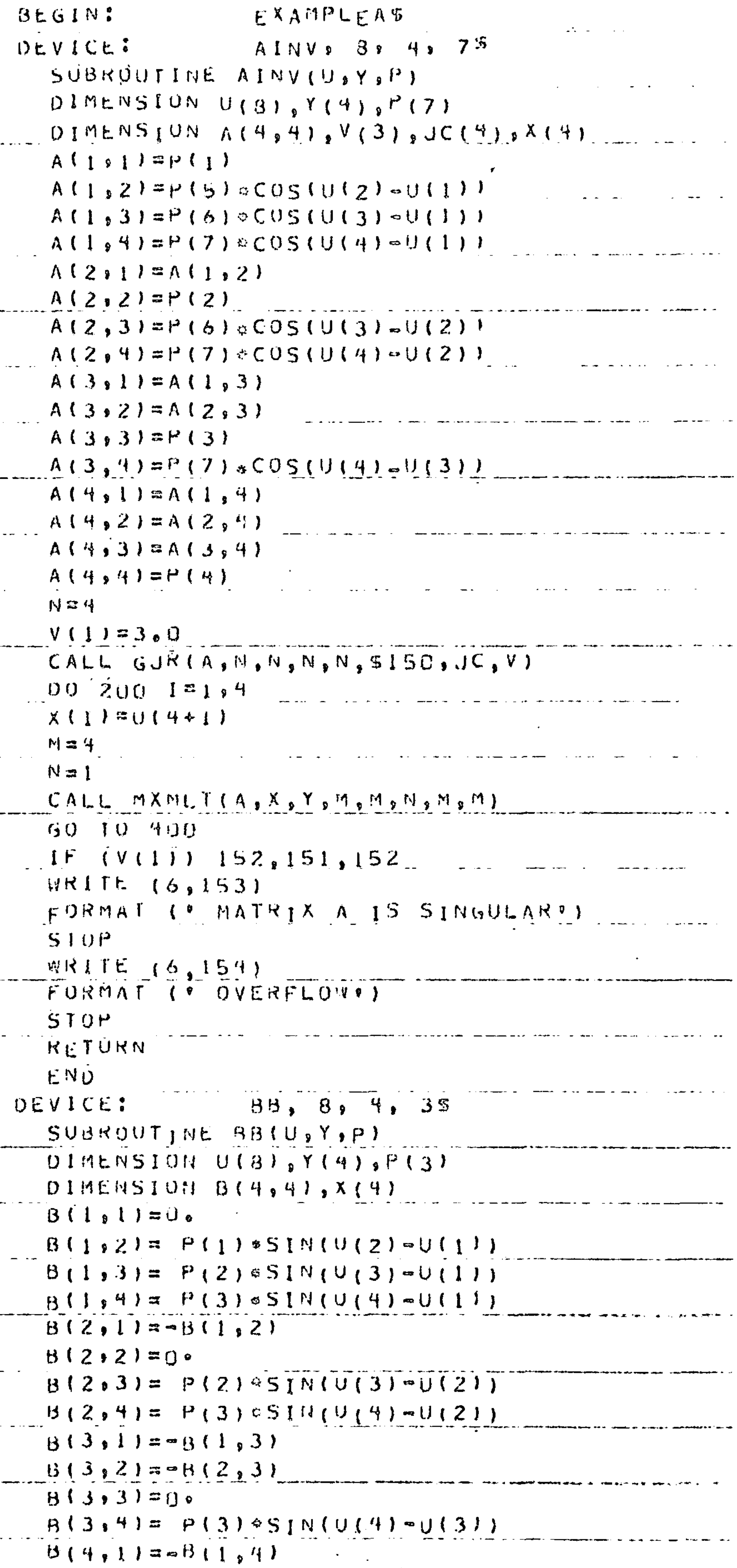


FIGURE 16. BLOCK DIAGRAM OF MECHANICAL EXTENSION DEVICE

on the following pages. Note that the translation time for each module is automatically printed out with the coding.

VI A. MECHANICAL EXTENSION DEVICE - EXAMPLE A
(Block Diagram Option)



```
      BEGIN:          EXAMPLEA$
      DEVICE:         AINV, 8, 4, 7$
         SUBROUTINE AINV(U,Y,P)
         DIMENSION U(8),Y(4),P(7)
         DIMENSION A(4,4),V(3),JC(4),X(4)
         A(1,1)=P(1)
         A(1,2)=P(5)*COS(U(2)-U(1))
         A(1,3)=P(6)*COS(U(3)-U(1))
         A(1,4)=P(7)*COS(U(4)-U(1))
         A(2,1)=A(1,2)
         A(2,2)=P(2)
         A(2,3)=P(6)*COS(U(3)-U(2))
         A(2,4)=P(7)*COS(U(4)-U(2))
         A(3,1)=A(1,3)
         A(3,2)=A(2,3)
         A(3,3)=P(3)
         A(3,4)=P(7)*COS(U(4)-U(3))
         A(4,1)=A(1,4)
         A(4,2)=A(2,4)
         A(4,3)=A(3,4)
         A(4,4)=P(4)
         N=4
         V(1)=3.0
         CALL GJR(A,N,N,N,N,$150,JC,V)
         DO 200 I=1,4
  200    X(I)=U(4+I)
         M=4
         N=1
         CALL MXMLT(A,X,Y,M,M,N,M,M)
         GO TO 400
  150    IF (V(1)) 152,151,152
  151    WRITE (6,153)
  153    FORMAT (' MATRIX A IS SINGULAR')
         STOP
  152    WRITE (6,154)
  154    FORMAT (' OVERFLOW')
         STOP
  400    RETURN
         END
      DEVICE:         BB, 8, 4, 3$
         SUBROUTINE BB(U,Y,P)
         DIMENSION U(8),Y(4),P(3)
         DIMENSION B(4,4),X(4)
         B(1,1)=0.
         B(1,2)= P(1)*SIN(U(2)-U(1))
         B(1,3)= P(2)*SIN(U(3)-U(1))
         B(1,4)= P(3)*SIN(U(4)-U(1))
         B(2,1)=-B(1,2)
         B(2,2)=0.
         B(2,3)= P(2)*SIN(U(3)-U(2))
         B(2,4)= P(3)*SIN(U(4)-U(2))
         B(3,1)=-B(1,3)
         B(3,2)=-B(2,3)
         B(3,3)=0.
         B(3,4)= P(3)*SIN(U(4)-U(3))
         B(4,1)=-B(1,4)
```

VI A. MECHANICAL EXTENSION DEVICE - EXAMPLE A
(Block Diagram Option)

```
          B(4,2)=-B(2,4)
          B(4,3)=-B(3,4)
          B(4,4)=0.
          DO 100 I=1,4
  100     X(I)=U(I+4)**2
          M=4
          N=1
          CALL MXMLT(B,X,Y,M,M,N,M,M)
          RETURN
          END
       DEVICE:          MC, 4, 4, 4$
          SUBROUTINE MC(U,Y,P)
          DIMENSION U(4),Y(4),P(4)
          REAL K
          DIMENSION K(4,4)
          K(1,1)=2*P(1)
          K(1,2)=-P(1)
          K(1,3)=0.
          K(1,4)=0.
          K(2,1)=-P(2)
          K(2,2)=2*P(2)
          K(2,3)=-P(2)
          K(2,4)=0.
          K(3,1)=0.
          K(3,2)=-P(3)
          K(3,3)=2*P(3)
          K(3,4)=-P(3)
          K(4,1)=0.
          K(4,2)=0.
          K(4,3)=-P(4)
          K(4,4)=P(4)
          M=4
          N=1
          CALL MXMLT(K,U,Y,M,M,N,M,M)
          RETURN
          END
       DEVICE:          DM, 2, 4, 2$
          SUBROUTINE DM(U,Y,P)
          DIMENSION U(2),Y(4),P(2)
          DIMENSION D(4,2)
          D(1,1)=-P(1)
          D(1,2)=0.
          D(2,1)=0.
          D(2,2)=0.
          D(3,1)=0.
          D(3,2)=0.
          D(4,1)=0.
          D(4,2)=-P(2)
          M=4
          N=1
          L=2
          CALL MXMLT(D,U,Y,M,L,N,M,L)
          RETURN
          END
       MODEL:           EXTENSIONDEVICE$
       INPUTS:          THETA0, DELTA$
```

E

VI A. MECHANICAL EXTENSION DEVICE - EXAMPLE A
(Block Diagram Option)

```
OUTPUTS:       THETA1, THETA2, THETA3, THETA4$
ELEMENTS:      AINV,    AX(6.423385, 6.361275, 6.299165, 3.242055, 6.366275,
                        6.304165, 6.242055)$
        :      BB,      BX(6.366275, 6.304165, 6.242055)$
        :      MC,      CX(10., 10., 10., 10.)$
        :      DM,      DX(10., 10.)$
        :      AD,      AD1, AD2, AD3, AD4$
        :      IN,      IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8$
CONNECT:       THETA0, 1#DX#1, AD1, 5#AX#1, IN1, IN2, THETA1$
       :       DELTA, 2#DX#2, AD2, 6#AX#2, IN3, IN4, THETA2$
       :       DX#3, AD3, 7#AX#3, IN5, IN6, THETA3$
       :       DX#4, AD4, 8#AX#4, IN7, IN8, THETA4$
       :       IN2, 1#CX#1, AD1$
       :       IN4, 2#CX#2, AD2$
       :       IN6, 3#CX#3, AD3$
       :       IN8, 4#CX#4, AD4$
       :       IN2, 1#AX$
       :       IN4, 2#AX$
       :       IN6, 3#AX$
       :       IN8, 4#AX$
       :       IN2, 1#BX#1, AD1$
       :       IN4, 2#BX#2, AD2$
       :       IN6, 3#BX#3, AD3$
       :       IN8, 4#BX#4, AD4$
       :       IN1, 5#BX$
       :       IN3, 6#BX$
       :       IN5, 7#BX$
       :       IN7, 8#BX$
END$

            MODEL                        1.513400 SEC.
SIMULATE:      EXTENSIONDEVICE$

EXCITE:        DELTA, FRAMP(25.136), THETA0, ZERO$
STOPIF:        TIME.GT..25$
END$
ALL INITIAL CONDITIONS HAVE BEEN DEFAULTED TO ZERO.

            SIMULATE                     1.393600 SEC.
PRINT:         DELTA, THETA1, THETA2, THETA3, THETA4$
SAMPLE:        STEP, 1$
END$

            POST PROC                    3.434200 SEC.
END:           EXAMPLEA$

            TOTAL                        6.695600 SEC.
```

VI A. MECHANICAL EXTENSION DEVICE - EXAMPLE A
(Equation Option)

```
    BEGIN:         EXAMPLEA$
    DEVICE:        AINV, 8, 4, 7$
      SUBROUTINE AINV(U,Y,P)
      DIMENSION U(8),Y(4),P(7)
      DIMENSION A(4,4),V(3),JC(4),X(4)
      A(1,1)=P(1)
      A(1,2)=P(5)*COS(U(2)-U(1))
      A(1,3)=P(6)*COS(U(3)-U(1))
      A(1,4)=P(7)*COS(U(4)-U(1))
      A(2,1)=A(1,2)
      A(2,2)=P(2)
      A(2,3)=P(6)*COS(U(3)-U(2))
      A(2,4)=P(7)*COS(U(4)-U(2))
      A(3,1)=A(1,3)
      A(3,2)=A(2,3)
      A(3,3)=P(3)
      A(3,4)=P(7)*COS(U(4)-U(3))
      A(4,1)=A(1,4)
      A(4,2)=A(2,4)
      A(4,3)=A(3,4)
      A(4,4)=P(4)
      N=4
      V(1)=3.0
      CALL GJR(A,N,N,N,N,$150,JC,V)
      DO 200 I=1,4
200   X(I)=U(4+I)
      M=4
      N=1
      CALL MXMLT(A,X,Y,M,M,N,M,M)
      GO TO 400
150   IF (V(1)) 152,151,152
151   WRITE (6,153)
153   FORMAT (' MATRIX A IS SINGULAR')
      STOP
152   WRITE (6,154)
154   FORMAT (' OVERFLOW')
      STOP
400   RETURN
      END
    MODEL:         EXTENSIONDEVICE, EQUATIONS
    INPUTS:        TO, DEL$
    OUTPUTS:       TO1, TO2, TO3, TO4$
    EQUATION:      P1 = B * SI(1. , 0. , T2 - T1 ) * T2 ' ** 2 + C * SI(1. ,
                          0. , T3 - T1 ) * T3 ' ** 2 + D * SI(1. , 0. , T4 -
                          T1 ) * T4 ' ** 2 + 2 * K * T1 - K * T2 - K * TO $
            :      P2 = - B * SI(1. , 0. , T2 - T1 ) * T1 ' ** 2 + C * SI(1. ,
                          0. , T3 - T2 ) * T3 ' ** 2 + D * SI(1. , 0. , T4 -
                          T2 ) * T4 ' ** 2 - K * T1 + 2 * K * T2 - K * T3 $
            :      P3 = - C * SI(1. , 0. , T3 - T1 ) * T1 ' ** 2 - C * SI(1. ,
                          0. , T3 - T2 ) * T2 ' ** 2 + D * SI(1. , 0. , T4 -
                          T3 ) * T4 ' ** 2 - K * T2 + 2 * K * T3 - K * T4 $
            :      P4 = - D * SI(1. , 0. , T4 - T1 ) * T1 ' ** 2 - D * SI(1. ,
                          0. , T4 - T2 ) * T2 ' ** 2 - D * SI(1. , 0. , T4 -
                          T3 ) * T3 ' ** 2 - K * T3 + K * T4 - K * DEL $
            :      B = 6.366275 $
            :      C = 6.304165 $
```

VI A. MECHANICAL EXTENSION DEVICE - EXAMPLE A
(Equation Option)

```
          :     D = 6.242055 $
          :     K = 10. $
          :     T01 = T1 $
          :     T02 = T2 $
          :     T03 = T3 $
          :     T04 = T4 $
          :     T1 '' = AINV(6.423385 , 6.361275 , 6.299165 , 3.242055 ,
                        6.366275 , 6.304165 , 6.242055 , T1 , T2 , T3 ,
                        T4 , P1 , P2 , P3 , P4 , 1 ) $
          :     T2 '' = AINV(6.423385 , 6.361275 , 6.299165 , 3.242055 ,
                        6.366275 , 6.304165 , 6.242055 , T1 , T2 , T3 ,
                        T4 , P1 , P2 , P3 , P4 , 2 ) $
          :     T3 '' = AINV(6.423385 , 6.361275 , 6.299165 , 3.242055 ,
                        6.366275 , 6.304165 , 6.242055 , T1 , T2 , T3 ,
                        T4 , P1 , P2 , P3 , P4 , 3 ) $
          :     T4 '' = AINV(6.423385 , 6.361275 , 6.299165 , 3.242055 ,
                        6.366275 , 6.304165 , 6.242055 , T1 , T2 , T3 ,
                        T4 , P1 , P2 , P3 , P4 , 4 ) $
    END$
    SIMULATE:      EXTENSIONDEVICE$

    EXCITE:        DEL, FRAMP(25.136), TO, ZERO$
    STOPIF:        TIME.GT..25$
    END$
ALL INITIAL CONDITIONS HAVE BEEN DEFAULTED TO ZERO.
               SIMULATE                4.306400 SEC.
    PRINT:         DEL, T01, T02, T03, T04$
    SAMPLE:        STEP, 1$
    END$

               POST PROC               4.358400 SEC.
    END:           EXAMPLEA$

               TOTAL                  11.951800 SEC.
```

VI B. DO NOTHING SYSTEM WITH SUBMODELS - EXAMPLE B

B. Do Nothing System with Submodels - Example B

The system block diagrammed in Figure 17 contains most of the elements contained within the MARSYAS library. One purpose of presenting this fictitious model is to illustrate the use of submodels and how they are interconnected within the main model. Most of the features of MARSYAS described on the preceding pages (with the exception of EQUATION) have been incorporated in this example. It is intended to be used as a learning aid for the beginning MARSYAS user.

VI B. DONOTHING SYSTEM WITH SUBMODELS–EXAMPLE B

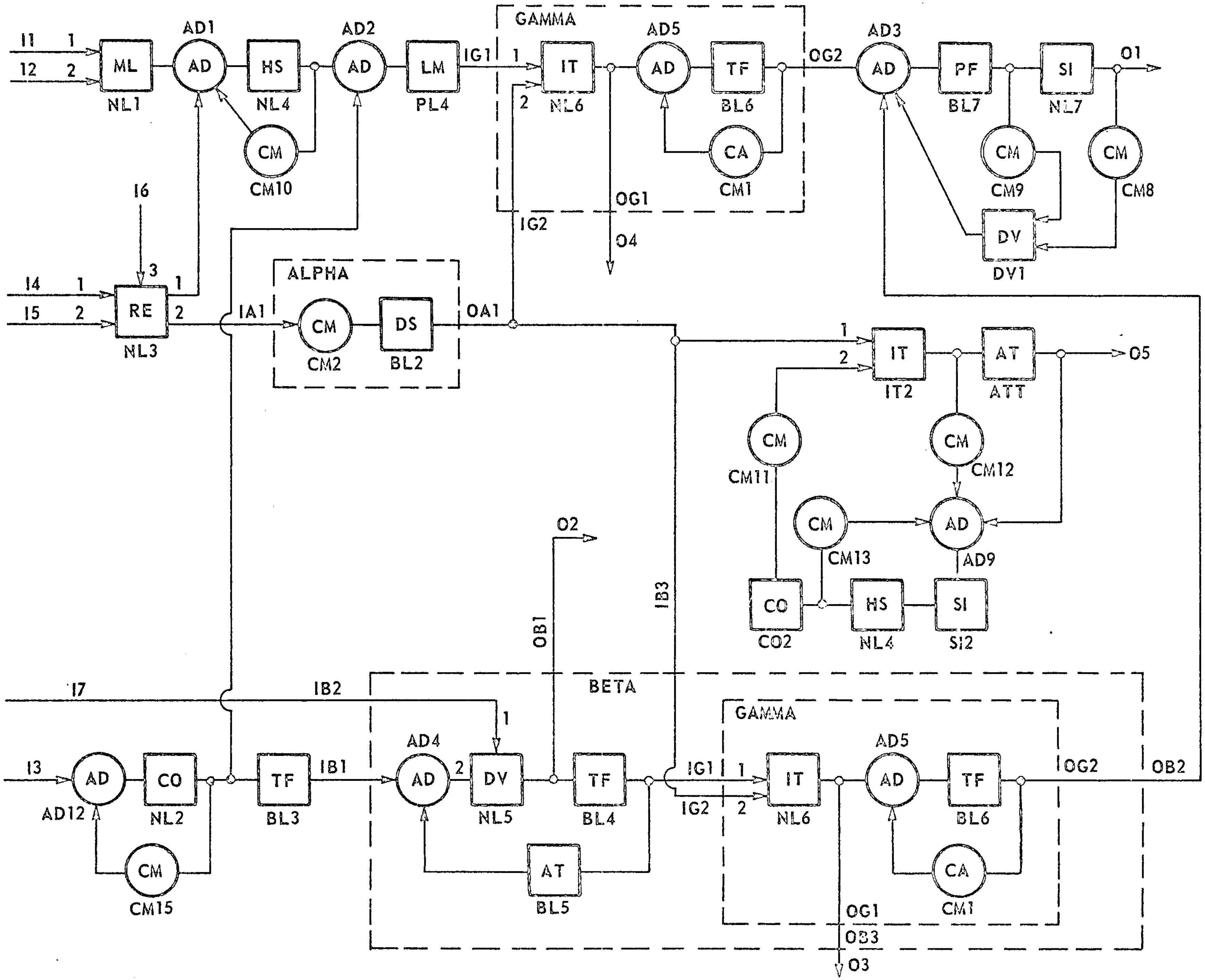


FIGURE 17. BLOCK DIAGRAM DONOTHING SYSTEM

VI B. MODEL OF DO NOTHING SYSTEM

```
      BEGIN:  EXAMPLE B$
*     A USER DEFINED ELEMENT$
      DEVICE:  AT, 1, 1, 1$
      SUBROUTINE AT(AI,AO)
      DIMENSION AI(1), AO(1)
      AO(1) = ATAN (AI(1))
      RETURN
      END

*     A USER DEFINED FUNCTION$
      FUNCTION, DFSIN, 5$
      FUNCTION  DFSIN (D1, D2, D3, D4, TIMEX)
C        THIS FUNCTION IS ALSO PART OF THE MARSYAS LIBRARY
      IF (TIMEX .LT. D4) GO TO 1
      DFSIN = D1 * SIN (D2 * (TIMEX - D4) + D3)
      RETURN
    1 DFSIN = 0.0
      RETURN
      END

      MODEL:  DO NOTHING MODEL$
      INPUTS:    I1,I2,I3,I4,I5,I6,I7$
      OUTPUTS:   O1,O2,O3,O4,O5$
      ELEMENTS:  AD,   AD1,AD2,AD3,AD9,AD12$
              :  SI,   NL7(1.,2.), SI2(2.0,3.0)$
              :  PF,   BL7(1.0)$
              :  ML,   NL1$
              :  RE,   NL3$
              :  TF,   BL3(2,1.,2.,3.,4.,5.,6.)$
              :  HS,   NL4(5,-2.,-2.,+2.,-2.,+6.,+2.,0.,+2., +2.,-2.)$
              :  HS,   HL4(5,-1.1,-2.0,+3.,-2.0,+7.,+3,0.0,0.0, 0.0,0.0)$
              :  CO,   NL2(1.0,2.0)$  : CO2(2.,3.)$
              :  LM,   PL4(2.,-1.,3.)$
              :  DV,   DV1$
              :  IT,   IT2$
              :  CM,   CM8(3.),CM9(4.),CM10(8.),CM11(5.),CM12(7.),CM13(6.),
                       CM15(9.)$
              :  AT,   ATT(1.3)$
      SUBMODEL:  BETA; INPUTS:  IB1,IB2,IB3; OUTPUTS:  OB1,OB2,OB3$
      SUBMODEL:  GAMMA; INPUTS:  IG1,IG2; OUTPUTS:  OG1,OG2$
      SUBMODEL:  ALPHA; INPUTS:  IA1; OUTPUTS:  OA2$
      CONNECT:  I1, I#NL1, AD1, NL4, AD2, PL4, IG1, OG2, AD3, BL7, NL7, O1$
             :  I2, 2#NL1$  : NL4, CM10, AD1$  : NL7, CM8, 1#DV1, AD3$
             :  BL7, CM9, 2#DV1$
```

VI B. MODEL OF DO NOTHING SYSTEM

```
CONNECT:  I4, 1#NL3#1, AD1$  : I6, 3#NL3$
       :  I5, 2#NL3#2, IA1, OA1, IG2$  : OA1, IB3$
       :  OA1, 1#IT1, ATT, O5$
       :  ATT, AD9$  : IT2, CM12, AD9$
       :  AD9, SI2, NL4, CM13, AD9$
       :  NL4, CO2, CM11, 2#IT2$  : OG1, O4$
CONNECT:  I7, IB2$  : I3, AD12, NL2, CM15, AD12$
       :  NL2, AD2$  : OB1, O2$
       :  NL2, BL3, IB1, OB2, AD3$
       :  OB3, O3$
END$

MODEL:  GAMMA$
INPUTS:    IG2,IG1$
OUTPUTS:   OG2,OG1$
ELEMENTS:  IT, NL6$  : AD, AD5$  : TF, BL6(4, 0.0, 1.0, 0.0, 0.0,
                                   1.0, 1.0, 3.0, 2.0, 1.0)$
        :  CA, CM1(2, +1)$
CONNECT:  IG1, 1#NL6, AD5, BL6, OG2$  : BL6, CM1, AD5$
       :  NL6, OG1$  : IG2, 2#NL6$
END$

MODEL:  BETA$
INPUTS:  IB1, IB2, IB3$
OUTPUTS:  OB3, OB2, OB1$
ELEMENTS:  AD, AD4$  : DV, NL5$, TF, BL4(4, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0,
                                 0.0, 3.0, 0.0, 2.0)$
        :  AT, BL5$
SUBMODEL:  GAMMA; INPUTS:  IG1, IG2; OUTPUTS:  OG1, OG2$
CONNECT:  IB1, AD4, 2#NL5, BL4, IG1, OG2, OB2$  : BL4, BL5, AD4$
       :  IB2, 1#NL5$  : NL5, OB1$  : IB3, IG2$  : OG1, OB3$
END$

MODEL:  ALPHA$
INPUTS:  IA1$
OUTPUTS:  OA1$
ELEMENTS:  CM, CM2$  : DS, BL2(2.,-3.)$
CONNECT:  IA1, CM2, BL2, OA1$
END$

SIMULATE:  DO NOTHING MODEL$
EXCITE:  I7, DSTEP (1.0, 2.0)$
      :  I1, ZERO$  : I2, FSTEP(1.0)$  : I3, FSIN(0.8,60.0,0.5)$
      :  I4, DSTEP(2.0,5.0)$  : I5, FRAMP(2.0)$
      :  I6, DAMP(1.0,2.0,3.0)$  : I7, FPULSE(1.0,2.1)$
INTEGRATE:  RK, TIMESTEP, 0.01$
```

VI B. MODEL OF DO NOTHING SYSTEM

```
CHANGE:  RUN2; EXCITE:  I7, DFSIN(1.0,1.0,0.0,1.0)$
CHANGE:  RUN3; PARAMETERS, GAMMA,  BL6(3, 2.6)$

TERMINATE IF:  TIME .GE. 20.0$
END$

PLOT:  LOGLOG(0.0,20.0,1), 01, 02, 03, 04, 05$
PRINT:  I1, I2, I3, I4, I5, I6, I7$
SAMPLE:  STEP, 10$
FOURIER:  04(8.0)$  : 06(8.7)$
END$
END:  EXAMPLE B$
```

VI C. VEHICLE STABILIZATION SYSTEM - EXAMPLE C

C. Vehicle Stabilization System - Example C

The Vehicle Stabilization System shown in Figure 18 is an example of a system containing nested submodels and differential equations. This is an example of mixing the block diagram and Equation methods to solve a problem.

VIC. VEHICLE STABILIZATION SYSTEM–EXAMPLE C

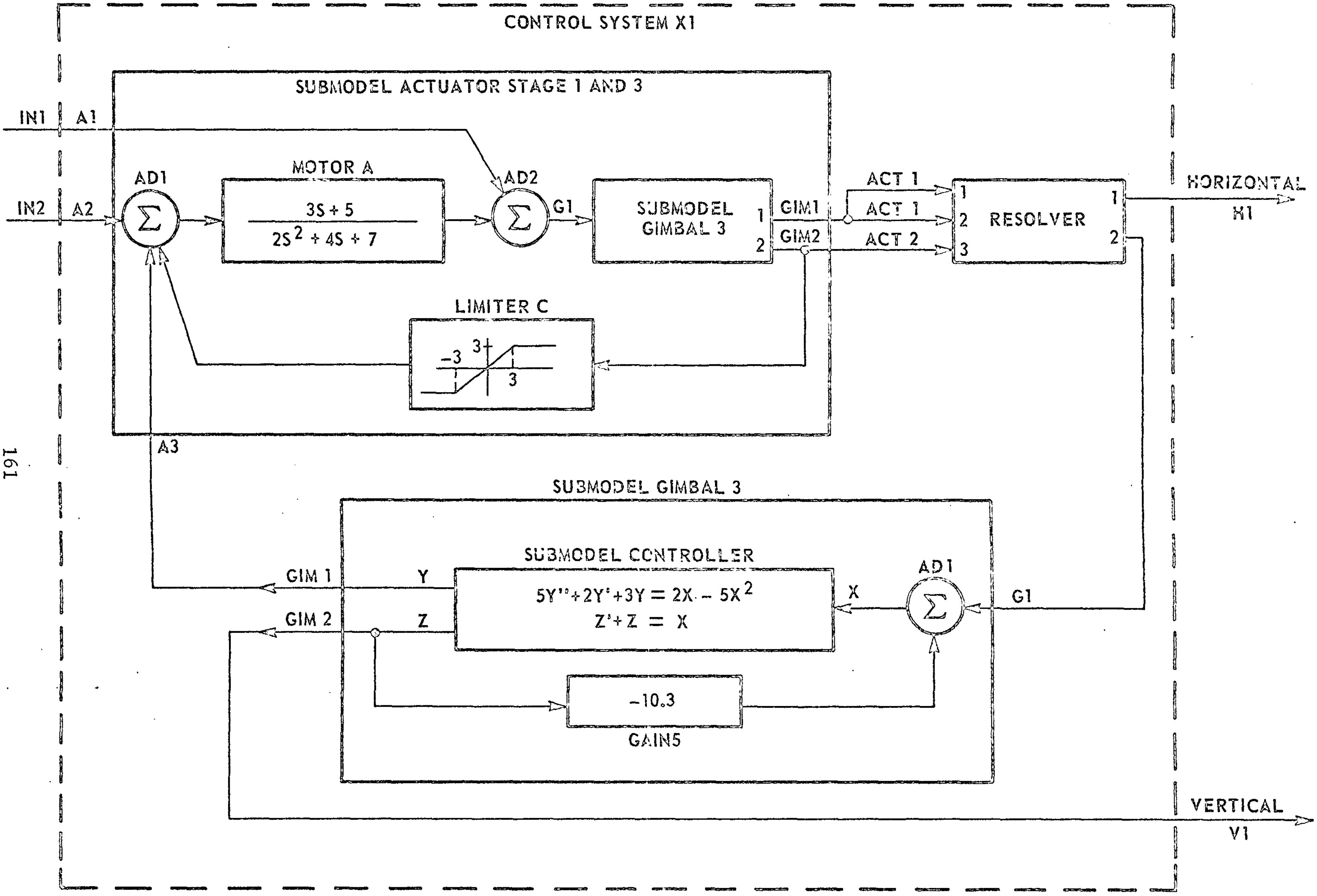


FIGURE 18 VEHICLE STABILIZATION SYSTEM BLOCK DIAGRAM

VI C. VEHICLE STABILIZATION SYSTEM - EXAMPLE C

```
    BEGIN:  VEHICLE STABILIZATION SYSTEM$
    MODEL:  CONTROL SYSTEM X1$
    INPUTS:  IN1, IN2$     OUTPUTS:  H1, V1$
    ELEMENTS:  RE, RESOLVER $
    SUBMODEL:  ACTUATOR STAGE 1 AND 3; INPUTS:  A1, A2, A3;
                      OUTPUTS:  ACT1, ACT2$
    SUBMODEL:  GIMBAL3; INPUTS:  G1; OUTPUTS:  GIM1, GIM2$
    CONNECT:  IN1, A1, ACT1, 1#RESOLVER#1, H1$
           :  ACT1, 2#RESOLVER#2, G1, GIM1, A3$
           :  IN2, A2, ACT2, 3#RESOLVER $
           :  GIM2, V1$
    END$

* THE FOLLOWING IS THE DESCRIPTION MODULE FOR SUBMODEL ACTUATOR STAGE 1 AND 3$
    MODEL:  ACTUATOR STAGE 1 AND 3$
    INPUTS:  A1, A2, A3$     OUTPUTS:  ACT1, ACT2$
    ELEMENTS:  AD, AD1, AD2$ :  TF, MOTORA(2, 0.0, 3.0, 5.0, 2.0, 4.0, 7.0)$
            :  LM, LIMITERC(1., -3., 3.)$
    SUBMODEL:  GIMBAL3;  INPUTS: G1;  OUTPUTS: GIM1, GIM2$
    CONNECT:  A1, AD2$  : A2, AD1, MOTORA, AD2, G1, GIM1, ACT1$
           :  GIM2, ACT2$  : GIM2, LIMITERC, AD1$  : A3, AD1$
    END$

* THE FOLLOWING IS THE DESCRIPTION MODULE FOR SUBMODEL GIMBAL3$
    MODEL:  GIMBAL3$
    INPUTS:  G1$     OUTPUTS:  GIM1, GIM2$
    ELEMENTS:  AD, AD1$  : CM, GAIN5(-10.3)$
    SUBMODEL:  CONTROLLER;  INPUTS:  X ;  OUTPUTS: Y, Z$
    CONNECT:  G1, AD1, X, Z, GAIN5, AD1$  : Z, GIM2$  : Y, GIM1$
    END$

* THE FOLLOWING IS THE DESCRIPTION MODULE FOR SUBMODEL CONTROLLER$
    MODEL:  CONTROLLER, EQUATION$
    INPUTS:  X$     OUTPUTS:  Y, Z$
    EQUATION:  Z' + Z  =  X$
            :  5.0 * Y" + 2.0 *Y' + 3.0 * Y = 2.0 * X - 5.0 * X ** 2
    END$

    SIMULATE:  CONTROL SYSTEM X1$
    INITIALIZE:  ACTUATOR STAGE 1 AND 3, MOTORA(1.5, 12.0)$
    EXCITE:  IN1, FSTEP(5.0)$  : IN2, FSIN (1.0, 3000.0, 0.0)$
    TERMINATE IF:  TIME .GT. 2.0$
    END$

    PRINT:  (VEHICLE STABILIZATION SYSTEM, RUN ONE)$
         :  IN1, IN2, H1, V1$
    FOURIER:  H1(1.3)$  : V1(1.3)$
    END$

    END:  VEHICLE STABILIZATION SYSTEM$
```

TABLE OF STANDARD ELEMENTS

TABLE OF STANDARD ELEMENTS

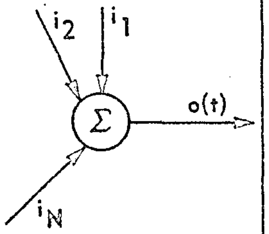
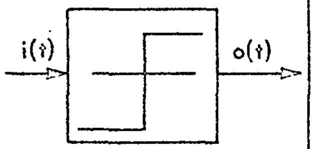
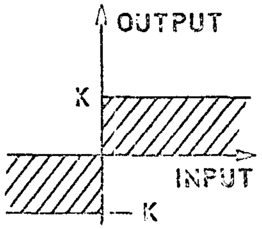
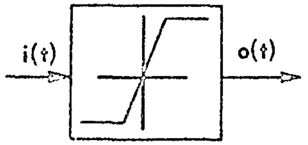
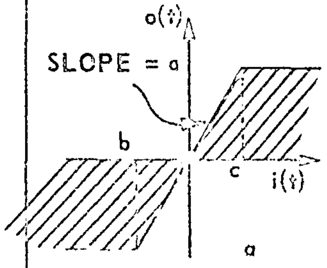
CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUT-PUTS	MNE-MONIC	INPUT-OUTPUT RELATION	LIST OF PARAMETERS IN THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
TRANSFER FUNCTION	$i(t) \rightarrow \boxed{\frac{\sum_{i=0}^{N} a_i s^i}{\sum_{i=0}^{N} b_i s^i}} \rightarrow o(t)$	1	1	TF	$\sum_{i=0}^{N} b_i \frac{d^i o(t)}{dt^i} = \sum_{i=0}^{N} a_i \frac{d^i i(t)}{dt^i}$	N = HIGHEST POWER OF TRANSFER FUNCTION $(N, a_N, a_{N-1}, a_{N-2}, ..., a_0, b_N, b_{N-1}, b_{N-2}, ..., b_0)$
CONSTANT MULTIPLIER	$i(t) \rightarrow \boxed{K} \rightarrow o(t)$	1	1	CM	$o(t) = K\, i(t)$	K
ADDER		N	1	AD	$o(t) = \sum_{i=1}^{N} i_i(t)$	NONE
IDEAL RELAY		1	1	IR		K $(K > 0)$
LIMITER		1	1	LM		(a,b,c) (EITHER $a > 0$, $b < 0$, $c > 0$ or $a < 0$, $b > 0$, $c < 0$)

TABLE OF STANDARD ELEMENTS (CONT'D)

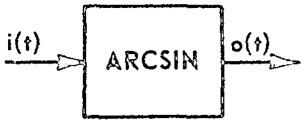
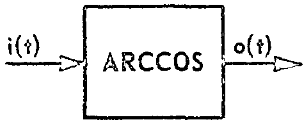
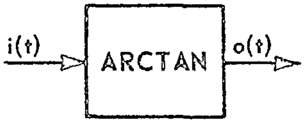
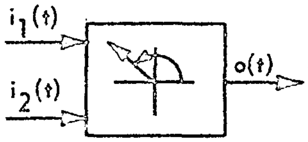
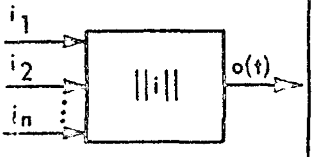
CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUTPUTS	MNE-MONIC	INPUT- OUTPUT RELATION	LIST OF PARAMETERS IN THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
ARCSINE		1	1	AS	$o(t) = \arcsin(i(t))$ $-1 \leq i(t) \leq 1$ $-\frac{\pi}{2} \leq o(t) \leq \frac{\pi}{2}$	NONE
ARCCOSINE		1	1	AC	$o(t) = \arccos(i(t))$ $-1 \leq i(t) \leq 1$ $0 \leq o(t) \leq \pi$	NONE
ARCTANGENT		1	1	AT	$o(t) = \arctan(i(t))$ $-\frac{\pi}{2} < o(t) < \frac{\pi}{2}$	NONE
INVERSE TANGENT		2	1	IT	$o(t) = \tan^{-1}\left(\frac{i_2(t)}{i_1(t)}\right)$ $0 \leq o(t) < 2\pi$	NONE
GENERALIZED DISTANCE		n (arbi-trary)	1	DI	$o(t) = \sqrt{i_1(t)^2 + \cdots + i_n(t)^2}$	n IS THE NUMBER OF INPUTS

TABLE OF STANDARD ELEMENTS (CONT'D)

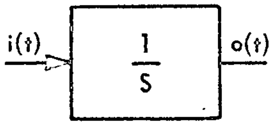
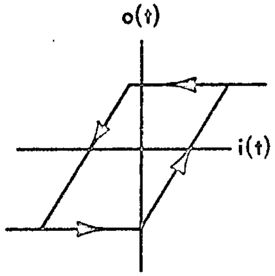
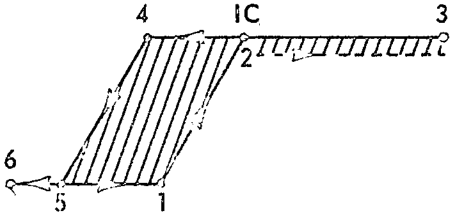
CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUTPUTS	MNEMONIC	INPUT-OUTPUT RELATION	LIST OF PARAMETERS IN THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
INTEGRATOR		1	1	IN	$o(t) = \int_0^t i(t)$	NONE
HYSTERESIS		1	1	HS		n = NUMBER OF POINTS $(n+1, x_1, y_1, x_2, y_2, \ldots x_n, y_n, \underbrace{x_{ic}, y_{ic})}_{\text{STARTING POINTS}}$ NOTE: THE DEFINITION OF THE POINTS COMPRISING THE CLOSED SYSTEM CAN START AT ANY POINT OF THE FIGURE. THE ORDER IN WHICH THE POINTS ARE GIVEN DETERMINES THE DIRECTION OF THE HYSTERESIS.

TABLE OF STANDARD ELEMENTS (CONT'D)

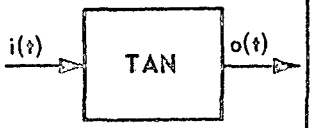
CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUTPUTS	MNE-MONIC	INPUT-OUTPUT RELATION	LIST OF PARAMETERS IN THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
EXPO-NENTIATOR		1	1	EX	$o(t) = e^{i(t)}$	NONE
LOGARITHMIC		1	1	LN	$o(t) = \log_e i(t)$, $i(t) > 0$	NONE
SINE		1	1	SI	$o(t) = \sin(ai(t) + b)$ (*i* assumed to be in radians)	(a,b)
COSINE		1	1	CO	$c(t) = \cos(ai(t) + b)$ (*i* assumed to be in radians)	(a,b)
TANGENT		1	1	TN	$o(t) = \tan(ai(t) + b)$ (*i* assumed to be in radians)	(a, b)

TABLE OF STANDARD ELEMENTS (CONT'D)

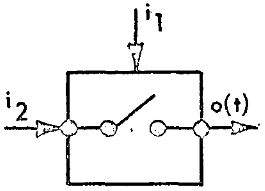
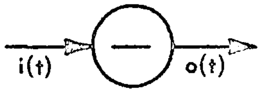
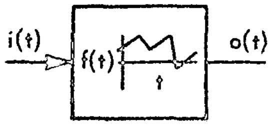
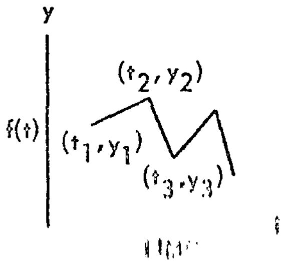
CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUTPUTS	MNE-MONIC	INPUT-OUTPUT RELATION	LIST OF PARAMETERS IN THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
SWITCH		2	1	SW	$o(t) = \begin{cases} i_2(t), \ i_1(t) \geq T \\ 0, \ i_1(t) < T \end{cases}$	T
SIGN CHANGER		1	1	SC	$o(t) = -\ i(t)$	NONE
TIME VARYING COEFFICIENT		1	1	TV	$o(t)=i(t) \circ f(t)$ 	$(n,\ t_1,\ y_1,\ t_2,\ y_2, \ldots, t_n,\ y_n)$

TABLE OF STANDARD ELEMENTS (CONT'D)

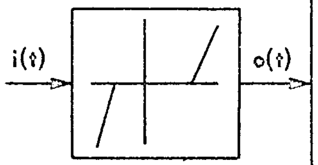
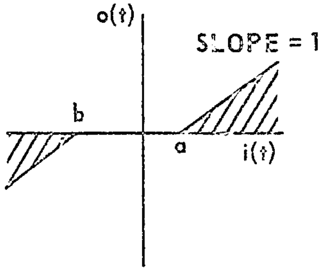
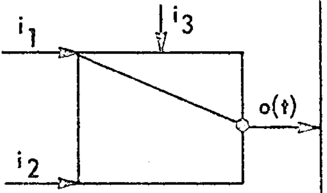
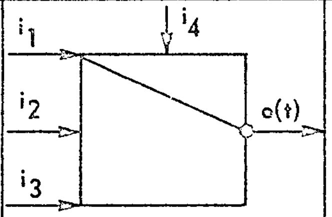
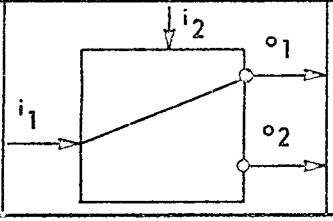
CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUT-PUTS	MNE-MONIC	INPUT-OUTPUT RELATION	LIST OF PARAMETERS IN THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
DEAD SPACE		1	1	DS		(a,b) WHERE a > b a AND/OR b MAY BE POSITIVE OR NEGATIVE.
INPUT RELAY		3	1	RI	$o(t) = \left\{ \begin{array}{l} i_1(t), i_3(t) \geq T \\ i_2(t), i_3(t) < T \end{array} \right\}$	T
FUNCTION SWITCH		4	1	FS	$o(t) = \left\{ \begin{array}{l} i_1(t), i_4(t) > o \\ i_2(t), i_4(t) = o \\ i_3(t), i_4(t) < o \end{array} \right\}$	NONE
OUTPUT RELAY		2	2	RO	$\begin{array}{c\|c\|c} & i_2 < T & i_2 \geq T \\ \hline o_1(t) & i_1(t) & 0 \\ \hline o_2(t) & 0 & i_1(t) \end{array}$	T

TABLE OF STANDARD ELEMENTS (CONT'D)

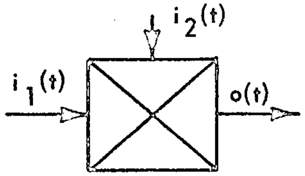
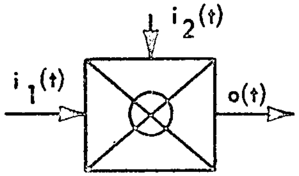
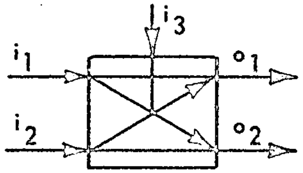
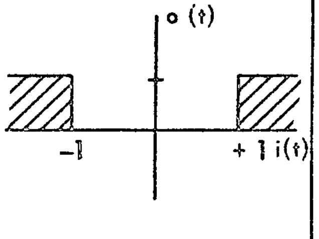
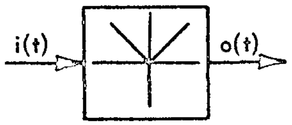
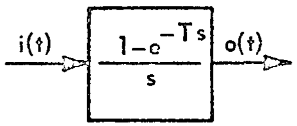
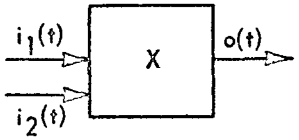
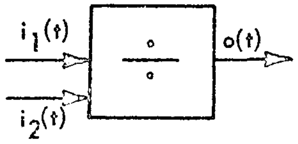
CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUTPUTS	MNE-MONIC	INPUT-OUTPUT RELATION	LIST OF PARAMETERS ON THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
BOOLEAN RELAY		2	1	BM	$o(t) = \begin{cases} i_1(t), & i_2(t) \neq 0 \\ 0, & i_2(t) = 0 \end{cases}$	NONE
		2	1	BR	$o(t) = \begin{cases} 0, & i_2 \neq 0 \\ i_1(t), & i_2 = 0 \end{cases}$	NONE
RESOLVER		3	2	RE	$\begin{bmatrix} o_1 \\ o_2 \end{bmatrix} = \begin{bmatrix} \cos(i_3) & \sin(i_3) \\ -\sin(i_3) & \cos(i_3) \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix}$	NONE
THRESHOLD		1	1	TH	$o(t) = \begin{cases} 1, & \lvert i(t) \rvert > I \\ 0, & \lvert i(t) \rvert \leq I \end{cases}$	I $(I > 0)$

TABLE OF STANDARD ELEMENTS CONT'D

CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUTPUTS	MNE-MONIC	INPUT-OUTPUT RELATION	LIST OF PARAMETERS IN THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
ABSOLUTE VALUE		1	1	AB	$o(t) = \lvert i(t) \rvert$	NONE
SAMPLE * AND HOLD		1	1	SH	$o(t) = i(nT),$ $nT \leq t < (n+1)\,T,$ $n = 0, 1, 2, \ldots$	T $(T > 0)$
MULTIPLIER		2	1	ML	$o(t) = i_1(t) \cdot i_2(t)$	NONE
DIVIDER		2	1	DV	$o(t) = \dfrac{i_1(t)}{i_2(t)}$	NONE

* IF SUCH A DEVICE IS INCLUDED IN THE SYSTEM, THE RUNGE KUTTA SCHEME FOR INTEGRATION MUST BE SPECIFIED IN AN INTEGRATE STATEMENT IN THE SIMULATION MODULE WITH TIMESTEP LESS THAN T.

TABLE OF STANDARD ELEMENTS (CONT'D)

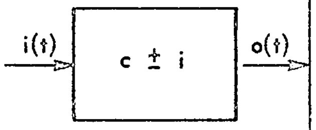
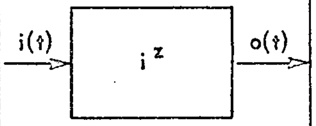
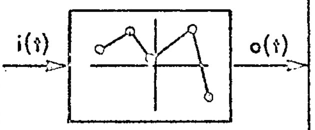
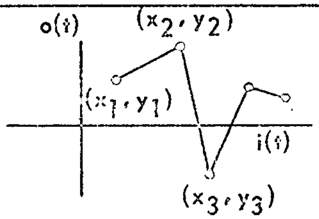
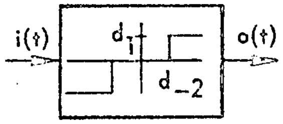
CLASS	BLOCK DIAGRAM SYMBOL	# OF INPUTS	# OF OUT-PUTS	MNE-MONIC	INPUT-OUTPUT RELATION	LIST OF PARAMETERS IN THE ORDER IN WHICH THEY APPEAR IN THE ELEMENTS STATEMENT
CONSTANT ADDER		1	1	CA	$o(t) = c \pm i(t)$	(c, + i)
POWER FUNCTION		1	1	PF	$o(t) = (i(t))^z$ z REAL $i(t) > o$	z
ARBITRARY FUNCTION GENERATOR		1	1	AF		n = NUMBER OF POINTS x_i, y_i = COORDINATES OF POINTS $(n, x_1, y_1, x_2, y_2, \ldots x_n, y_n)$
NORMAL RELAY		1	1	NR	$o(t) = \begin{cases} d_1, i(t) \geq d_2 \\ o, -d_2 < i(t) < d_2 \\ -d_1, i(t) \leq -d_2 \end{cases}$	(d_1, d_2)

TABLE OF STANDARD EXCITATION FUNCTIONS

TABLE OF STANDARD EXCITATION FUNCTIONS

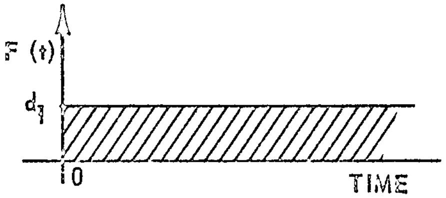
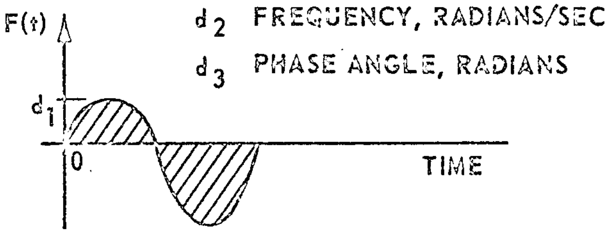
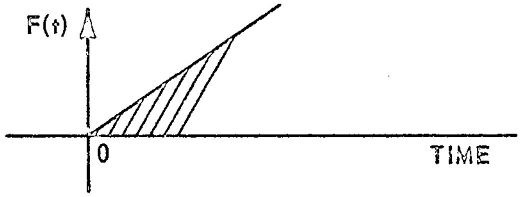
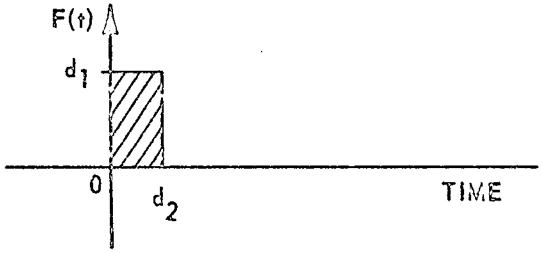
FUNCTION AND ARGUMENTS	TYPE	MATHEMATICAL DESCRIPTION	GRAPH
FSTEP (d_1)	STEP INPUT	$F(t) = d_1 \, , \, t \geq 0$	
FSIN (d_1, d_2, d_3)	SINUSOID	$F(t) = d_1 \sin(d_2 \, t + d_3)$	
FRAMP (d_1)	RAMP INPUT	$F(t) = d_1 \, t$	 d_1 = SLOPE
FPULSE (d_1, d_2)	SQUARE PULSE	$F(t) \begin{cases} = d_1, \, 0 \leq t \leq d_2 \\ = 0, \, t > d_2 \end{cases}$	

TABLE OF STANDARD EXCITATION FUNCTIONS (CONT'D)

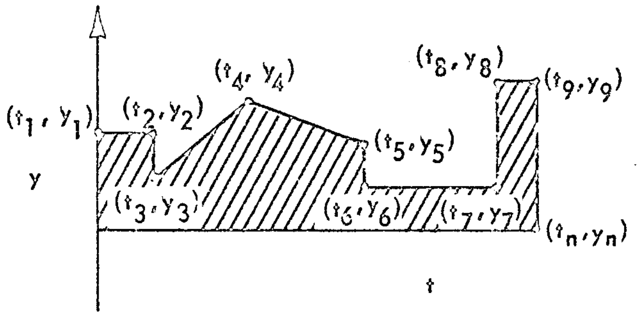
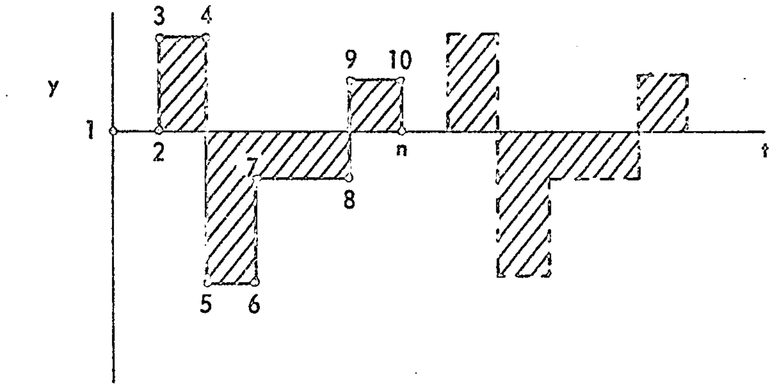
FUNCTION AND ARGUMENTS	TYPE	DESCRIPTION	GRAPH
TRAIN ($n, t_1, y_1\ t_2, y_2, t_3, y_3, \ldots t_n, y_n$)	ARBITRARY PULSE TRAIN	F(t)=y	 (t_1, y_1) (t_2, y_2) (t_4, y_4) (t_8, y_8) (t_9, y_9) (t_5, y_5) y (t_3, y_3) (t_6, y_6) (t_7, y_7) (t_n, y_n) t
PTRAIN ($n, t_1, y_1, t_2, y_2, t_3, y_3, \ldots t_n, y_n$)	PERIODIC ARBITRARY PULSE TRAIN	F(t)=y	 3 4 9 10 y 1 2 n t 7 8 5 6

TABLE OF STANDARD EXCITATION FUNCTIONS (CONT'D)

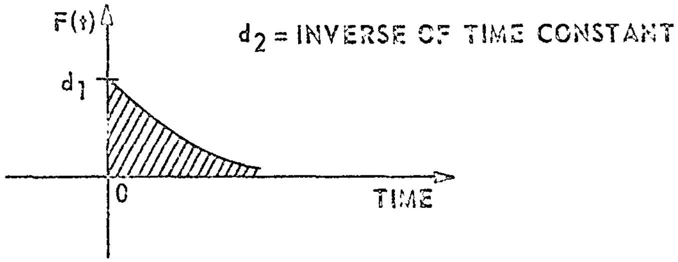
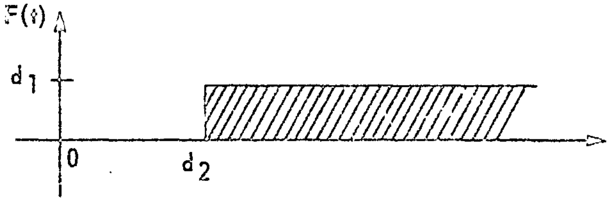
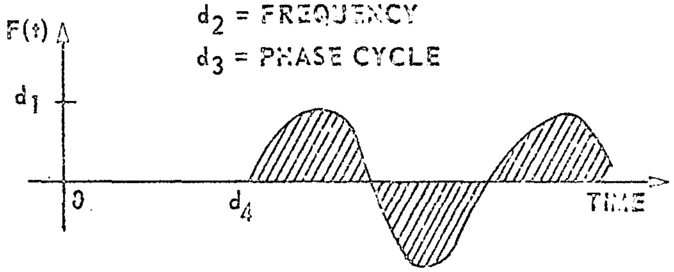
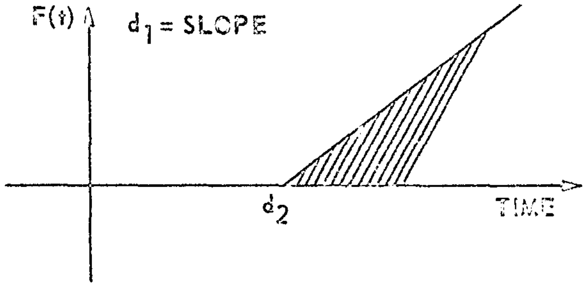
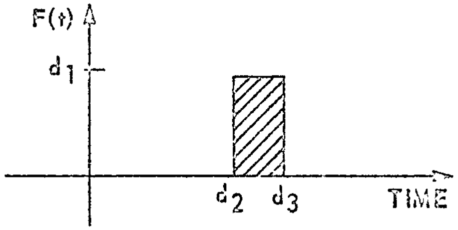
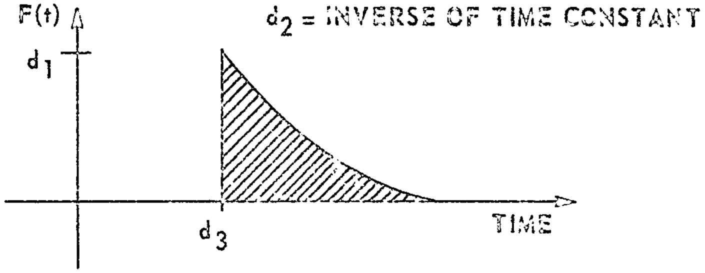
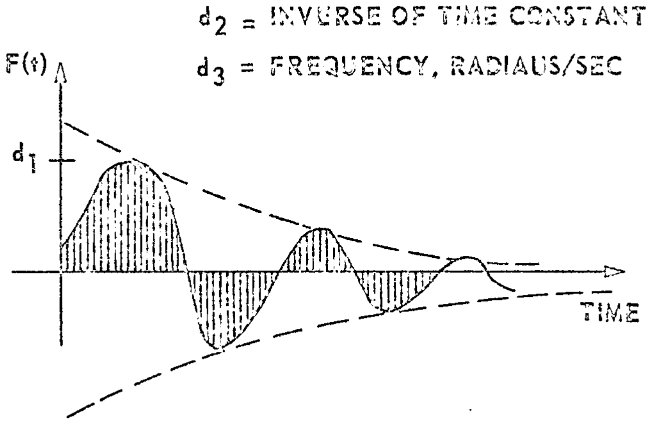
FUNCTION AND ARGUMENTS	TYPE	MATHEMATICAL DESCRIPTION	GRAPH
FEXP (d_1, d_2)	EXPONENTIAL FUNCTION	$F(t) = d_1 e^{-d_2 t}$	
DSTEP (d_1, d_2)	DELAYED STEP	$F(t) = \begin{cases} 0, & 0 \le t < d_2 \\ d_1, & t \ge d_2 \end{cases}$	
DFSIN (d_1, d_2, d_3, d_4)	DELAYED SINUSOID	$F(t) = \begin{cases} 0, & 0 \le t < d_4 \\ d_1 \sin(d_2 (t-d_4) + d_3), & t \ge d_4 \end{cases}$	
DRAMP (d_1, d_2)	DELAYED RAMP	$F(t) = \begin{cases} 0, & 0 \le t < d_2 \\ d_1(t-d_2), & t \ge d_2 \end{cases}$	

TABLE OF STANDARD EXCITATION FUNCTIONS (CONT'D)

FUNCTION AND ARGUMENTS	TYPE	MATHEMATICAL DESCRIPTION	GRAPH
DPULSE (d_1, d_2, d_3)	DELAYED PULSE	$F(t) \left\{ \begin{array}{l} = 0, 0 \le t < d_2 \;\&\; t > d_3 \\ = d_1, d_2 \le t \le d_3 \end{array} \right\}$	
DEXP (d_1, d_2, d_3)	DELAYED EXPONENTIAL	$F(t) = d_1 e^{-d_2(t-d_3)}$	
DAMP (d_1, d_2, d_3)	DAMPED SINUSOID	$F(t) = d_1 e^{-d_2 t} \sin d_3 t$	
ZERO	ZERO FOR ALL TIME	$F(t) = 0$	

REFERENCES

(1) H. Trauboth and N. Prasad, "MARSYAS-A Software System for the Digital Simulation of Physical Systems," Proc. of Spring Joint Computer Conference, May 1970.

(2) N. Prasad and J. Reiss, "The Digital Simulation of Interconnected Systems," Proc. of International Association of Cybernetics Conference, Namur, Belgium, September 1970.

(3) H. Trauboth and N. Prasad, "MARSYAS-A Software Engineering System for the Digital Simulation and Analysis of Physical Systems," International Federation of Automatic Control (IFAC), Budapest Symposium on Digital Simulation of Continuous Process (DISCOP), Gyor, Hungary, September 1971.

(4) N. Prasad and H. Gabow, "ADEPT-An Algebraic and Differential Equations Processor and Translator," Proc. of the 1971 Summer Computer Simulation Conference, pp. 50-63, Vol. 1, Boston, Massachusetts, July 1971.

(5) S. P. Singh, "Digital Simulation of Serpentuator Using MARSYAS," NASA TR R-414, Computation Laboratory, Marshall Space Flight Center, Alabama, December 1972.

BIBLIOGRAPHIC DATA SHEET

1. Report No. EERC 72-12

2.

3. Recipient's Accession No. PB-220 207

4. Title and Subtitle
SHAKE: a computer program for earthquake response analysis of horizontally layered sites

5. Report Date
December 1972

6.

7. Author(s) Per B. Schnabel, John Lysmer and H. Bolton Seed

8. Performing Organization Rept. No.

9. Performing Organization Name and Address
University of California
Earthquake Engineering Research Center
1301 South 46th Street
Richmond, Calif. 94804

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Address
National Science Foundation
1800 G Street
Washington, D.C. 20550

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

The program computes the response in a horizontally layered soil rock system subjected to transient, vertical travelling shear waves. The method is based on Kanai's solution to the wave equation and the Fast Fourier Transform algorithm. The motion used as basis for the analysis can be applied to any layer in the system. Systems with elastic base and with variable damping in each layer can be analyzed. Equivalent linear soil properties are used with an iterative procedure to obtain soil properties compatible with the strains developed in each layer. A varied set of operations of interest in earthquake response analysis can be performed.

17. Key Words and Document Analysis. 17a. Descriptors

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement
Release unlimited

i-a

19. Security Class (This Report)
UNCLASSIFIED

20. Security Class (This Page
UNCLASSIFIED

21. No. of Pages

THIS FORM MAY BE REPRODUCED

INSTRUCTIONS FOR COMPLETING FORM NTIS-35 (10-70) (Bibliographic Data Sheet based on COSATI Guidelines to Format Standards for Scientific and Technical Reports Prepared by or for the Federal Government, PB-180 600).

1. **Report Number.** Each individually bound report shall carry a unique alphanumeric designation selected by the performing organization or provided by the sponsoring organization. Use uppercase letters and Arabic numerals only. Examples FASEB-NS-87 and FAA-RD-68-09.

2. Leave blank.

3. **Recipient's Accession Number.** Reserved for use by each report recipient.

4. **Title and Subtitle.** Title should indicate clearly and briefly the subject coverage of the report, and be displayed prominently. Set subtitle, if used, in smaller type or otherwise subordinate it to main title. When a report is prepared in more than one volume, repeat the primary title, add volume number and include subtitle for the specific volume.

5. **Report Date.** Each report shall carry a date indicating at least month and year. Indicate the basis on which it was selected (e.g., date of issue, date of approval, date of preparation.

6. **Performing Organization Code.** Leave blank.

7. **Author(s).** Give name(s) in conventional order (e.g., John R. Doe, or J.Robert Doe). List author's affiliation if it differs from the performing organization.

8. **Performing Organization Report Number.** Insert if performing organization wishes to assign this number.

9. **Performing Organization Name and Address.** Give name, street, city, state, and zip code. List no more than two levels of an organizational hierarchy. Display the name of the organization exactly as it should appear in Government indexes such as **USGRDR-I.**

10. **Project/Task/Work Unit Number.** Use the project, task and work unit numbers under which the report was prepared.

11. **Contract/Grant Number.** Insert contract or grant number under which report was prepared.

12. **Sponsoring Agency Name and Address.** Include zip code.

13. **Type of Report and Period Covered.** Indicate interim, final, etc., and, if applicable, dates covered.

14. **Sponsoring Agency Code.** Leave blank.

15. **Supplementary Notes.** Enter information not included elsewhere but useful, such as: Prepared in cooperation with . . . Translation of . . . Presented at conference of . . . To be published in . . . Supersedes . . . Supplements . . .

16. **Abstract.** Include a brief (200 words or less) factual summary of the most significant information contained in the report. If the report contains a significant bibliography or literature survey, mention it here.

17. **Key Words and Document Analysis. (a). Descriptors.** Select from the Thesaurus of Engineering and Scientific Terms the proper authorized terms that identify the major concept of the research and are sufficiently specific and precise to be used as index entries for cataloging.
(b). Identifiers and Open-Ended Terms. Use identifiers for project names, code names, equipment designators, etc. Use open-ended terms written in descriptor form for those subjects for which no descriptor exists.
(c). COSATI Field/Group. Field and Group assignments are to be taken from the 1965 COSATI Subject Category List. Since the majority of documents are multidisciplinary in nature, the primary Field/Group assignment(s) will be the specific discipline, area of human endeavor, or type of physical object. The application(s) will be cross-referenced with secondary Field/Group assignments that will follow the primary posting(s).

18. **Distribution Statement.** Denote releasability to the public or limitation for reasons other than security for example "Release unlimited". Cite any availability to the public, with address and price.

19 & 20. **Security Classification.** Do not submit classified reports to the National Technical

21. **Number of Pages.** Insert the total number of pages, including this one and unnumbered pages, but excluding distribution list, if any.

22. **Price.** Insert the price set by the National Technical Information Service or the Government Printing Office, if known.

EARTHQUAKE ENGINEERING RESEARCH CENTER

SHAKE

A COMPUTER PROGRAM FOR EARTHQUAKE RESPONSE ANALYSIS OF HORIZONTALLY LAYERED SITES

by

Per B. Schnabel

John Lysmer

H. Bolton Seed

A report on research sponsored by the
National Science Foundation

Report No. EERC 72-12

December 1972

College of Engineering
University of California
Berkeley, California

i-6