SIMNON - User's Manual

Elmqvist, Hilding

1975

Document Version:
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Elmqvist, H. (1975). *SIMNON - User's Manual*. (Research Reports TFRT-3091). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

SIMNON

AN INTERACTIVE SIMULATION PROGRAM FOR NONLINEAR SYSTEMS

USER'S MANUAL

H. ELMQVIST

Report 7502 April 1975
Department of Automatic Control
Lund Institute of Technology

SIMNON
An interactive simulation program
for nonlinear systems
USER'S MANUAL

H. Elmqvist

ABSTRACT

SIMNON is a command driven interactive program written in FORTRAN for simulation of systems governed by ordinary differential equations and difference equations.

The system description can be done as separate descriptions of subsystems and their interconnections. Each subsystem can be described either by ordinary differential equations or difference equations. The language to describe the subsystems is either a special simulation language or FORTRAN. The program contains an editor and a compiler for the simulation language.

CONTENTS

TABLES

1. INTRODUCTION

SIMNON is a command driven interactive program written in FORTRAN for simulation of systems governed by ordinary differential equations and difference equations.

SIMNON has been designed for interactive simulation. The interaction is mainly done via a display and a keyboard. During a simulation selected variables are plotted on the display and the user can then e.g. decide to alter parameter values, to select other plot variables or to change the structure of the system before the next simulation.

The program is command driven which means that when a task has been carried out the user decides what to do next by giving a command for that. There are commands to

- define (change) the system to be simulated
- change parameter values
- change initial values of dependent variables
- perform simulation
- select variables for plotting
- draw axes.

It is often convenient to consider a system as being composed of a set of subsystems, e.g. a process, a state estimator and a state feedback or a process controlled by a computer. One of the basic ideas in SIMNON is to allow the user to describe the subsystems separately and then describe their interconnection. Each subsystem can either be a continuous time system (described by ordinary differential equations) or a discrete time system (described by difference equations). These systems will from now on be called continuous systems and discrete systems. It is also possible to include subsystems with time delays. This can e.g. be done according to the method in [2].

A continuous subsystem is described by

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

and a discrete subsystem is described by

$$x(t_{i+1}) = f(x(t_i),t_i,u(t_i))$$
$$y(t_i) = g(x(t_i),t_i,u(t_i))$$

Notations:

t - time
x - state vector
u - input vector
y - output vector
t_i - the i:th sampling instance

The connection of the subsystems is done as

$$u(t) = h(x(t),t,y(t))$$

with appropriate definitions of the outputs and the states of discrete systems between the sampling instances.

A simple continuous system

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

is obviously a special case.

The concept of discrete systems in SIMNON is powerful. Examples of what discrete systems can contain are given below.

- model of computer with control algorithm
- algorithm for the solution of a twopoint boundary value problem with a shooting method (see example 7)
- algorithm for parametric optimization in dynamic systems, [1]
- synchronization of the simulation time with real time
- input of data from files on mass storage or A/D-converters
- output of data to files on mass storage, line printer or D/A-converters
- direct manipulation of states in other subsystems.

The system description can in SIMNON be done either with a special simulation language or in FORTRAN-subroutines with special structure.

The simulation language is based on ALGOL:s assignment statements. SIMNON has a built-in compiler for this language. As the language is developed for the description of differential and difference equations it has been possible to include more extensive error checking than is possible in ordinary programming languages. It is e.g. tested that the variables have values before they are used.

The compiler is working in parallel with an editor. When the compiler discovers an error an error message is given and the user directly gets the possibility to correct the erroneous line by certain editing commands.

The two methods of describing systems complement each other in many ways:

- Systems which are described in the simulation language are easy to modify. Modifications of systems described in FORTRAN requires a recompilation of the FORTRAN-subroutine by a systems program and the reloading of the program.

- The simulation language is not as powerful as FORTRAN. It is e.g. not possible to make jumps in the system description or calls to subroutines. Use of vectors and matrices is not possible either.

- Due to the special structure of the system description and the restrictions in the simulation language it has been possible to do more error checking than is possible in FORTRAN-compilers.

SIMNON is imbedded in a general interactive language, called INTRAC. In this language it is possible to store sequences of commands in a so called MACRO (file on mass storage) and then, each time the sequence is to be executed, to give the name of the MACRO as a command. It is also possible to introduce

repetitive loops and make tests and jumps among the commands.

This manual describes the version of SIMNON (3E) that existed in April 1975. It only describes those features which are computer independent.

Chapter 2 contains a short description of the basic facilities in SIMNON. A general discussion of system description and simulation, particularly the connection of subsystems, is given in chapter 3. Chapters 4 and 5 describes in detail how system descriptions are made in the simulation language and in FORTRAN. The description of the interaction is made in chapters 6 and 7 and chapter 8 contains some examples.

2. HOW TO GET STARTED WITH SIMNON

This chapter contains the description of the basic facilities in SIMNON. It is intended that the beginner should be able to use the program on simple problems after reading this chapter. Note, that most of what is said about SIMNON here is generalized in later chapters.

2.1. System description

The problem considered in this chapter is to make a simulation of a system described by a set of ordinary differential equations

$$\dot{x}(t) = f(x(t),t) \qquad (1)$$

$$x(t_o) = x_o \qquad (2)$$

where

x is the vector of dependent variables (states)
t is the independent variable (time)
x_o is the initial values of the dependent variables
t_o is the initial value of the independent variable

This section will treat the description of such a system in the simulation language of SIMNON.

The system description contains an algorithm that defines the function f in (1), i.e. from given values of x and t the algorithm evaluates $\dot{x}$. In the simulation language this algorithm consists of a set of assignment statements of the same type as in ALGOL-60.

It is frequently desirable to change parameters in the differential equations during the simulations. To assign default values of parameters there is a special statement which is also used to give initial values for the states. The statement will later be called constant assignment.

The structure of the system description for a simple continuous system is given in table 1. The system description begins with a heading that gives the system a name. Then follow declarations of the variables corresponding to time, states and derivatives. After the declarations follow statements for computation of auxiliary variables and derivatives and the assignments of parameters and initial values of states. The system description is terminated with END. Comments can be written at the end of lines if they are preceded by a double quote (").

Table 1. Structure of simple continuous systems

```
CONTINUOUS SYSTEM identifier

TIME identifier
STATE identifier...
DER identifier...

computation of auxiliary variables (assignment statements)
computation of derivatives (assignment statements)
constant assignments

END
```

Declarations

Declarations are required to specify types of variables. They are of the form

```
type variable...
```

type is one of the words TIME, STATE and DER. DER is the type for the time derivatives of state variables.

The declarations of STATE- and DER-variables must be done so that the STATE-variables and the corresponding DER-variables appear in the same order.

Parameters and auxiliary variables are not declared.

Examples

```
TIME T
STATE U V
DER DU DV
```

Assigment statements

Assignment statements are used to assign derivatives and auxiliary variables. It is of the same type as in ALGOL-60 [3].

The basic forms of the assignment statement are:

```
variable = expression
variable = IF condition THEN expression ELSE expression
```

The expressions are composed of variables, numbers, the operators +-*/↑, parenthesis and functions. The condition is composed of expressions with the relational operators > < and the Boolean operators OR AND NOT.

Available functions are listed in table 6 on page 4.9.

Examples

```
DX1 = X2
DX2 = -(A-2*Q*COS(2*T))*X1
V = 5.75*12/(.08-2.59E-4)+2E5
U = IF T < 1 OR T > 2 THEN 0 ELSE 1
```

Constant assignment

The constant assignments are used to give values to parameters and initial values to states. Its form is

```
variable: number
```

If the initial value is not specified for some state variable the value zero is assumed.

Examples

```
X:-10
ALPHA:1.5E-3
```

Example 1. A triode oscillator.

As a first example of system description in the simulation language consider a model for a triode oscillator.

In [4] the following differential equation for the grid potential is given:

$$\frac{d^2u}{dt^2} + \varepsilon(1-g(u))\frac{du}{dt} + u = 0$$

with the function g defined as

$$g(u) = \begin{cases} 2 - u^2 & |u| < \sqrt{2} \\ 0 & |u| \geq \sqrt{2} \end{cases}$$

In order to make the system description in SIMNON the differential equation must be rewritten as a set of first order differential equations:

$$\frac{du}{dt} = v$$

$$\frac{dv}{dt} = -u - \varepsilon(1-g(u))v$$

The system description can be done as

```
CONTINUOUS SYSTEM TRIOD
STATE U V
DER DU DV
G = IF ABS(U)<SQRT(2) THEN 2-U*U ELSE 0
DU = V
U:0.25
DV = -U-EPS*(1-G)*V
EPS:1
END
```

2.2. Running the program

The use of SIMNON is controlled by commands. The program signifies that it is ready to read a command by typing a > on the terminal. A command begins with one of a set of words that specifies the action the program should take. This word can be followed by arguments that can be identifiers, numbers or delimiters.

A description of the basic commands and their function is given below.

Entering a system description into SIMNON (SYST)

The system description is stored as a symbolic file on mass storage. To demand for compilation of a system description the command SYST is used:

```
SYST filename
```

The argument of the command is the name of the file.

The compiler is working in parallel with an editor to make it possible to directly correct erroneous lines.

When making a new system description the dialogue between the user and the program is usually done in the following way (the messages from the program are underlined):

```
>SYST filename
   FILE NOT FOUND: filename
   INPUT
   input line
   ...
   input line (erroneous)
   error message
   EDIT
  >editing command (to correct the line)
  > empty line
   INPUT
   input line
   ...
```

```
        input line
        empty line
        EDIT
       >E
      >
```

If the named file does not exist on mass storage the editor will start in "INPUT-mode" which means that what is typed from the terminal is stored on the file. If an error is discovered by the compiler in a line an error message is written and the editor will switch to "EDIT-mode" which means that changes in the line can be done with certain editing commands.

To retype the entire line the command

```
R string                    (retype)
```

is used. string denotes those characters which are to replace the old line. R must be followed by one space.

The command

```
C /string1/string2/     (change)
```

can be used to change some characters in the line. The characters in string1 are then replaced by the characters in string2.

An empty line is written in order to return to INPUT-mode. When the entire system description is entered, an empty line is given to switch to EDIT-mode. After that the command

```
E                           (exit)
```

is given to finish the compilation.

To make changes in a line entered earlier it is necessary to go to the beginning of the file using the command

```
T                           (top)
```

After that it is possible to "go down" to the actual line by using the command

```
L string                    (locate)
```

which tells the editor to search for the specified string.

The command

```
N                           (next)
```

is used to go one line down the file.

When the actual line is found it can be modified using the commands R (retype) or C (change) or be deleted by the command

D (delete)

If the command SYST is given as before but the file already exits on mass storage the system description is compiled without the interaction of the user. The user gets the possibility to change the system description only if an error is detected. To change a syntactically correct system description the option EDIT must be included in the command.

The dialogues between the user and the program in these cases are shown below.

```
>SYST filename
>
...
>SYST filename
  error message
  EDIT
  > editing command
  > E
>
...
>SYST filename - EDIT
  EDIT
  >editing command
  >E
>
```

Selecting plot variables (PLOT)

Before a simulation is demanded the variables to be plotted must be named. This is done with the command PLOT:

PLOT variable ...

PLOT variable ... (variable)

The three dots denote that several variables can be named. The variables are normally plotted versus the simulation time but this can be changed so that the plotting is made versus an arbitrary variable. This variable is in that case named at the end of the command enclosed between parenthesis.

Drawing axes (AXES)

The generation of axes for the plotting is done by the command AXES:

AXES H minimum maximum V minimum maximum

The scaling of the axes is specified by entering minimum value and maximum value for each axis. H stands for horizontal and V for vertical. If the command is given without arguments the same scales of the axes as the last time is obtained.

Simulating (SIMU)

Simulation of the system is demanded with the command SIMU:

SIMU start-time stop-time
SIMU

The simulation is performed in the time interval start-time - - stop-time. If these arguments are omitted the latest specified interval is used.

Changing parameter value (PAR)

The value of a parameter can be changed by the command PAR:

PAR variable:number

Changing initial value of a state (INIT)

The initial value of a state variable can be change by the command INIT:

INIT variable:number

Leaving SIMNON (STOP)

The execution of SIMNON is stopped by the command STOP:

STOP

Example 2. Triode oscillator (continued).

A simulation of the triode oscillator from example 1 is done to demonstrate how the commands are used.

The dialogue between the user and SIMNON is shown below. The lines written by the program are underlined. Some errors were made when typing the system description to illustrate the editing facility.

```
>SYST TRIOD                    " DEFINE THE SYSTEM
   FILE NOT FOUND: TRIOD
   INPUT
  CONTINUOUS SYSTEM TRIOD
  STATE U V
  DER DU DV
  G=IF ABS U)<SQRT(2) THEN 2-U*U ELSE 0
   MISSING LEFT PARENTHESIS
         !
G=IF ABS U)<SQRT(2) THEN 2-U*U ELSE 0
   EDIT
  >                            " CORRECT THE LINE
  >C /ABS /ABS(/
G=IF ABS(U)<SQRT(2) THEN 2-U*U ELSE 0
  >                            " CHANGE MODE TO INPUT
  >
   INPUT
  DU=V
  U:0.25
  DV=-U-EPS*(1-G)*V
  END
   UNASSIGNED: EPS
   EDIT
  >                            " DEFINE THE PARAMETER EPS
  >R EPS:1
  >
   INPUT
  END

   EDIT
  >E                           " EXIT
>                              " SELECT U FOR PLOTTING
>PLOT U
>AXES X 0 25 V -5 5            " DRAW AXES
>SIMU 0 25                     " SIMULATE
>                              " SEE FIGURE 1
>INIT U:6                      " CHANGE INITIAL VALUES
>INIT V:-10
>SIMU
>                              " SEE FIGURE 1
>STOP
```

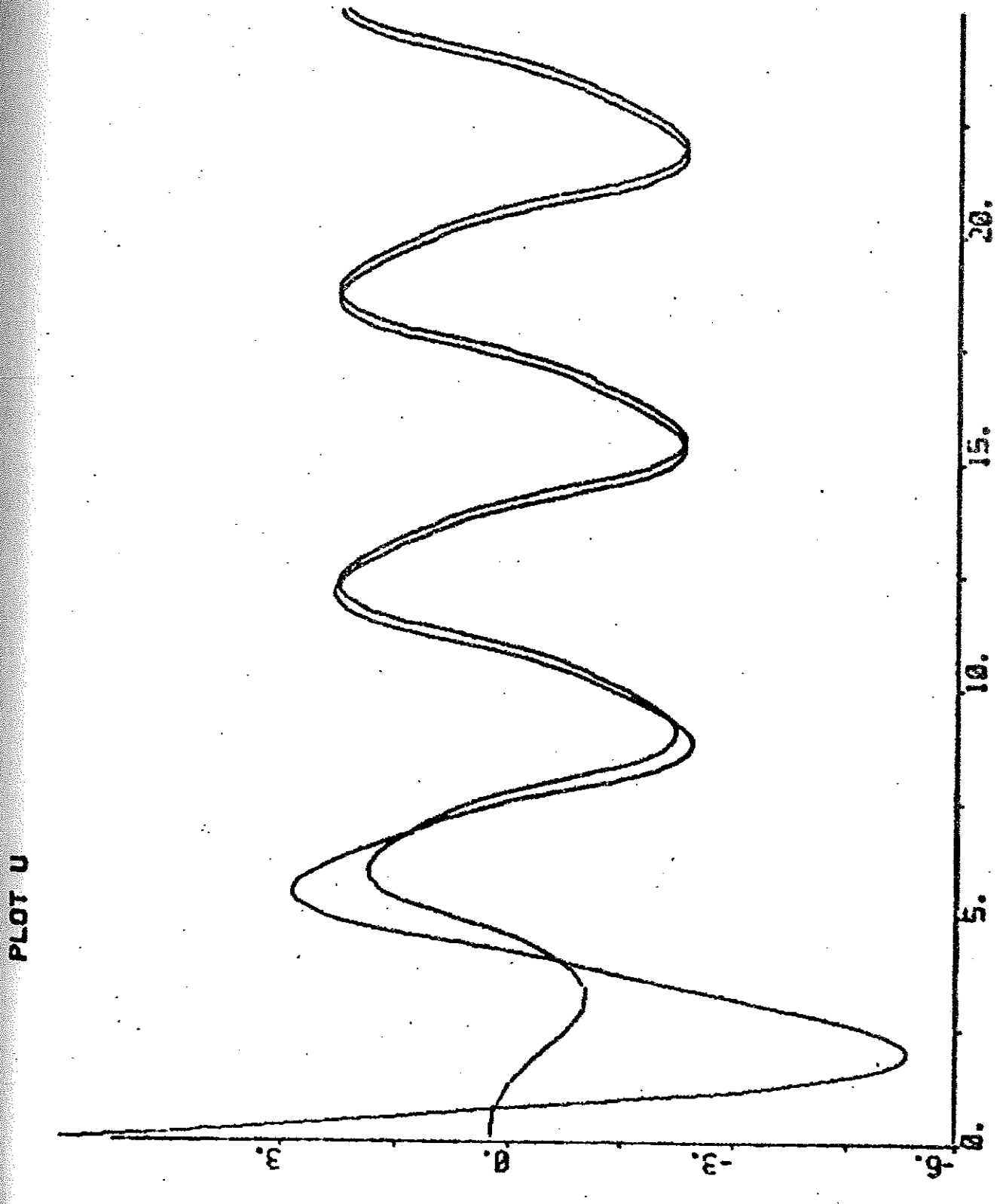


Figure 1.

3. SYSTEM DESCRIPTION AND SIMULATION IN GENERAL

3.1. Introduction

This chapter contains a general discussion of system description and simulation of sampled data systems [5] (systems described by both ordinary differential equations and difference equations).

One of the basic ideas in SIMNON is to allow the user to make the system description as separate descriptions of subsystems and their connections. This is often natural both physically and logically. Each subsystem can be either continuous or discrete.

3.2. Continuous systems

A continuous subsystem is described by a set of first order ordinary differential equations for the states and a set of equations for the outputs:

$$\dot{x}^k(t) = f^k(x^k(t),t,u^k(t)) \tag{1}$$

$$y^k(t) = g^k(x^k(t),t,u^k(t)) \tag{2}$$

$$x^k(t_o) = x_o^k \tag{3}$$

Notations:

- t - time
- x^k - the state vector in the k:th subsystem
- u^k - the input vector to the k:th subsystem
- y^k - the output vector from the k:th subsystem
- t_o - initial value for the time
- x_o^k - initial value for the state vector

The notations x, u and y will be used for the total state vector, total input vector and total output vector, i.e.

$$x = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^m \end{bmatrix} \qquad u = \begin{bmatrix} u^1 \\ u^2 \\ \vdots \\ u^m \end{bmatrix} \qquad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

where m is the number of subsystems.

An input to a subsystem can be a function of time, outputs from other subsystems and outputs from the subsystem itself. Sometimes state variables are introduced so that they correspond to outputs, thus it is desirable to allow inputs to depend on states as well. The connection of the subsystems is then defined by

$$u(t) = h(x(t),t,y(t)) \tag{4}$$

Analysis of the interconnections

The total system can be written

$$\dot{x}(t) = f(x(t),t,u(t)) \tag{1'}$$
$$y(t) = g(x(t),t,u(t)) \tag{2'}$$
$$x(t_o) = x_o \tag{3'}$$
$$u(t) = h(x(t),t,y(t) \tag{4 }$$

with the functions f and g defined by f^k and g^k ($k = 1,2,\ldots,m$).

The integration routines require $\dot{x}$ to be computed for given values of x and t. To do this u must first be computed by eliminating y between (2') and (4). Then $\dot{x}$ can be computed from (1').

Elimination of y gives

$$u = h(x,t,g(x,t,u)) \tag{5}$$

The result is a nonlinear system of equations that should be solved for given x and t. There is in the general case no guarantee that there exists an unique solution.

If an unique solution of (5) exists it can be written

$$u = H(x,t) \tag{6}$$

Inserting (6) into (1') gives

$$\dot{x} = f(x,t,H(x,t)) \tag{7}$$

or

$$\dot{x} = F(x,t) \tag{8}$$

Algorithm for the solution

The notation algebraic dependence will now be introduced. An output y_i is said to depend algebraically on an input u_i if

$$\frac{\partial g_i}{\partial u_j}(x,t,u) \neq 0 \quad \text{for some } x,t,u$$

If there exists a path in the system joined by algebraic dependence in the subsystems then this path is called an algebraic loop.

If the system does not contain any algebraic loops then there exists a unique solution of (5) and it is possible to find the solution directly by computing outputs and inputs in an appropriate order.

An algorithm for the solution of u (and y) is given below.

1. Compute all inputs u_i that only depends on x and t.
2. Compute all output vectors y^k, not computed earlier, which only depends on x, t and earlier computed inputs u_i.
3. Compute all inputs u_i, not computed earlier, which only depends on x, t and earlier computed outputs y_i.
4. If not all inputs and outputs are computed go to step 2.

It is often practical to compute all outputs from a subsystem at the same time. The algorithm is designed for this case.

System description and compilation

The description of a subsystem must contain one algorithm that for given values of x^k, t and u^k computes y^k i.e. defines the function g^k and one algorithm that for given values of x^k, t and u^k computes $\dot{x}^k$ i.e. defines the function f^k.

For the connection of the subsystems an algorithm is needed which computes the inputs u_i in accordance with the steps 1 and 3 in the method. The insertion of calls to the algorithms for output computations is easily done when compiling the algorithm for the connection. This is done by finding out, for each expression, which outputs are used and then generating calls to algorithms for computation of those outputs that are not earlier computed.

The work of the compiler when compiling the connection algorithm is:

1. Indicate all inputs and outputs as undefined.

2. Read the next input expression.

3. Find which outputs are used in the expression.

4. Find which outputs of stage 3 that are undefined.

5. Find which subsystems the outputs of stage 4 belong to.

6. Generate calls to the algorithms for output computation of the subsystems of stage 5. Indicate all outputs from these subsystems as defined.

7. Give warnings about undefined inputs to the subsystems of stage 5.

8. Generate executable code for the expression. Indicate the corresponding input as defined.

9. If more expressions then go to stage 2.

10. Give error message if any input is still undefined.

11. Generate calls to the algorithms for output computation of the subsystems whose output are still undefined.

12. Generate call to the algorithms for derivative computation in all systems.

The compiler gives warnings in stage 7 in order to increase the security when connecting subsystems.

When connecting subsystems the user should do the following.

1. Define some, not earlier defined, input that only depend on x, t and such outputs that belong to subsystems whose all outputs are not algebraically dependent of not earlier defined inputs.

2. Check the warnings if any.

3. If not all inputs are defined then go to stage 1.

Example

As an example of connecting subsystems consider the system in figure 2.

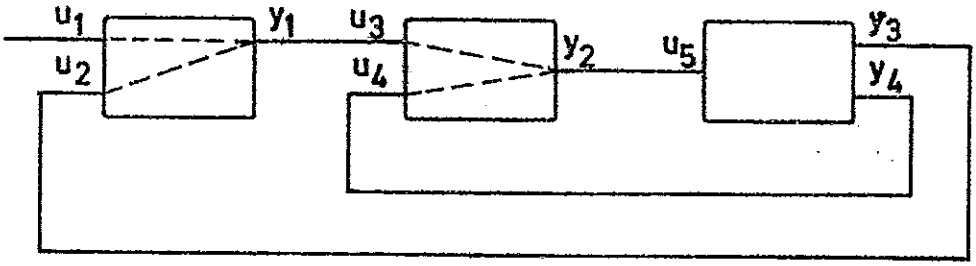


Figure 2.

When connecting the subsystems only the outputs algebraic dependence or independence on the corresponding inputs are of importance. For the particular example assume that

y_1 is algebraically dependent of u_1 and u_2,

y_2 is algebraically dependent of u_3 and u_4,

y_3 is algebraically independent of u_5,

y_4 is algebraically independent of u_5.

The following discussion shows the order in which the inputs should be specified. Since y_3 and y_4 do not algebraically depend on inputs and u_1 only depend on t, the inputs u_1, u_2 and u_4 can be specified in any order. For example

$$u_1 = h(t)$$

$$u_2 = y_3$$

$$u_4 = y_4$$

When u_1 and u_2 are defined, y_1 can be computed and u_3 specified as

$$u_3 = y_1$$

y_2 can now be computed because u_3 and u_4 are specified. u_5 can then be specified as

$$u_5 = y_2$$

All inputs are now defined.

Below is shown the users statements and what the compiler inserts.

the user	the compiler
$u_1 = h(t)$	
	call: compute y_3, y_4
	warning: u_5 not defined when computing y_3, y_4
$u_2 = y_3$	
$u_4 = y_4$	
	call: compute y_1
$u_3 = y_1$	
	call: compute y_2
$u_5 = y_2$	

The condition that y_3 and y_4 must be algebraically independent of u_5 is fulfilled. The inputs have thus been defined in a correct order.

If the inputs are defined in an other order, e.g. as below, other conditions will be set:

the user	the compiler
$u_1 = h(t)$	
	call: compute y_1
	warning: u_2 is not defined when computing y_1
$u_3 = y_1$	
	call: compute y_2
	warning: u_4 is not defined when computing y_2
$u_5 = y_2$	
	call: compute y_3, y_4
$u_2 = y_3$	
$u_4 = y_4$	

The given conditions are not fulfilled and therefore the order of the statements must be changed.

3.3. Discrete and mixed systems

A discrete subsystem is described by a set of difference equations for the states and a set of equations for the outputs:

$$x^k(t^k_{i+1}) = f^k(x^k(t_i),\ t^k_i,\ u^k(t^k_i)) \tag{9}$$

$$y^k(t^k_i) = g^k(x^k(t^k_i),\ t^k_i,\ u^k(t^k_i)) \tag{10}$$

$$x^k(t^k_1) = x^k_o \tag{11}$$

$$t^k_i < t^k_{i+1} \quad ; \qquad t_o \leq t^k_1 \tag{12}$$

Notation:

t^k_i - the i:th sampling instance for the k:th subsystem.

The outputs of the discrete subsystems are not defined between the sampling instances and before the first sampling instance. In order to allow for connection of discrete systems and

continuous systems and discrete systems with different sampling instances the definition of the outputs must be extended. This can be done by "connecting" zero order hold circuits to the outputs.

The outputs are then defined by

$$y^k(t) = \begin{cases} y^k(t_i^k) & t_i^k \leq t < t_{i+1}^k \\ y_o^k & t < t_1^k \end{cases} \qquad (13)$$

where y_o^k is the initial value for the output vector in the k:th subsystem.

To make it possible to do the connections in accordance with (4) the states of the discrete systems must also be defined for all $t \geq t_o$.

This is done as

$$x^k(t) = \begin{cases} x^k(t_i^k) & t_i^k \leq t < t_{i+1}^k \\ x^k(t_1^k) & t < t_1^k \end{cases} \qquad (14)$$

Analysis of the interconnections

Consider a system composed of a set of continuous subsystems (1-3) and a set of discrete subsystems (9-14) connected according to (4). The equations will be investigated between the sampling instances.

The equation (4) can be split up into a continuous part and a discrete part:

$$u_C = h_C(x_C, x_D, t, y_C, y_D) \qquad (4')$$

$$u_D = h_D(x_C, x_D, t, y_C, y_D) \qquad (4")$$

Notations:

x_C - the continuous part of the state vector
x_D - the discrete part of the state vector
etc.

The continuous subsystems are described by

$$\dot{x}_C = f_C(x_C, t, u_C) \qquad (1")$$

$$y_C = g_C(x_C, t, u_C) \qquad (2")$$

Insertion of (2") into (4') gives

$$u_C = h_C(x_C, x_D, t, g_C(x_C, t, u_C), y_D) \qquad (5')$$

If there exists an unique solution of (5') it can be written

$$u_C = H_C(x_C, t, x_D, y_D) \qquad (6')$$

The equations (1") and (6') gives

$$\dot{x}_C = f_C(x_C, t, H_C(x_C, t, x_D, y_D)) \qquad (7')$$

or

$$\dot{x}_C = F_C(x_C, t, x_D, y_D) \qquad (8')$$

The states and outputs of the discrete subsystems thus appears as parameters in the differential equations.

3.4. Simulation

Between the sampling instances the differential equations (8') are solved using an ordinary integration routine. At a sampling instance of a subsystem the states of the subsystem first has to be updated according to the equation (9) that was evaluated at the last sampling instance. New values of the outputs are then computed from (10) and the values of the states during the next sampling interval are determined from (9).

If several subsystems are to be sampled at the same instance it is very important that their outputs are computed in the correct order. The calls to the algorithms for output compu-

tation thus must be inserted among the connecting statements in the same way as for continuous systems. In this case, however, the compiler must generate a conditional call since the algorithms are to be executed just at the sampling instances and not during the integration of the differential equations between the sampling instances.

A natural way to specify the sampling instances $\{t_i\}$, i = 1,2,... of a discrete subsystem is to include one special variable in the system description that will contain the time for the next sampling. This variable has to be initialized to the time of the first sampling and then updated at each sampling instance.

Table 2 describes what happens during the simulation of a system containing continuous and discrete subsystems.

Notations

K_C = {k: k:th subsystem is continuous}

K_D = {k: k:th subsystem is discrete}

z^k - the value of the state vector of the k:th subsystem after the "next" sampling ($z^k(t_i^k) = x^k(t_{i+1}^k)$)

t_s^k - the time for the next sampling of the k:th subsystem

t_f - stop time for the simulation.

The simulation starts with an initialization of certain variables. Those subsystems that are to be sampled at the start time are then sampled and the variable t_s is computed as the time when the next sampling is to be done. The integration of the differential equations is then performed until time equals t_s and sampling is perform etc..

Table 2. Simulation of sampled data systems

Initialization

$t = t_o$

$x = x_o$

$y^k = y_o^k \qquad \forall k \in K_D$

$t_s^k = t_1^k \qquad \forall k \in K_D$

$z^k = x^k \qquad \forall k \in K_D$

Sampling

$x^k = z^k \qquad \forall k \in K_D : t_s^k = t$

$u = h(x,t,y)\ ;\ y^k = g^k(x^k,t,u^k) \qquad \forall k \in K_C$

$y^k(t_i^k) = g^k(x^k(t_i^k),t_i^k,u^k(t_i^k)) \qquad \forall k \in K_D : t_s^k = t$

$z^k(t_i^k) = f^k(x^k(t_i^k),t_i^k,u^k(t_i^k)) \qquad \forall k \in K_D : t_s^k = t$

$t_s^k = t_{i+1}^k \qquad \forall k \in K_D : t_s^k = t$

Simulation finished if $t = t_f$

$t_s = \min(t_f, \min(\{t_s^k\})) \qquad \forall k \in K_D : t < t_s^k$

Integration in the interval $t - t_s$

$u = h(x,t,y)\ ;\ y^k = g^k(x^k,t,u^k) \qquad \forall k \in K_C$

$\dot{x}^k = f^k(x^k,t,u^k) \qquad \forall k \in K_C$

go to sampling

4. THE SIMULATION LANGUAGE

4.1. Introduction

The description of the system to be simulated can be made as a connection of a set of continuous or discrete subsystems as discussed in chapter 3.

The description of the subsystems can either be done in a special simulation language or in FORTRAN. The connecting system (containing the statements that define the inputs to the subsystems) must be described in the simulation language.

This chapter describes the simulation language.

4.2. The structure of the system description

The algorithms in the system description consist of assignment statements of the same type as in ALGOL-60.

There is a special statement to assign values of parameters and to give initial values for the states. The statement will later be called constant assignment.

Each system description begins with a heading which gives the system type (continuous, discrete or connecting) and its name. Then follows declarations of the variables (not parameters and auxiliary variables) which are used. Variables are declared as inputs, outputs, states, derivatives etc..

After the declarations follow the algorithms of the system description. All types of systems can contain an initialization algorithm (INITIAL-section) which is executed only once before the simulation. Continuous and discrete systems can also contain one algorithm for the computation of outputs (OUTPUT-section) and one algorithm for the computation of derivatives and the new value of states (DYNAMICS-section). The connecting system contains one algorithm to compute the inputs (CONNECT-section).

The structures of the system descriptions are given in the tables 3, 4 and 5 containing heading, legal declarations and what kind of computations that are allowed in the different sections.

Table 3. Structure of continuous systems

Heading

CONTINUOUS SYSTEM <system identifier>

Declarations

INPUT
OUTPUT
TIME
STATE
DER

INITIAL-section

Computation of auxiliary variables (parametric expressions)
Computation of initial values for state variables

OUTPUT-section

Computation of auxiliary variables
Computation of output variables
Computation of derivatives

DYNAMICS-section

Computation of auxiliary variables
Computation of derivatives

Table 4. Structure of discrete systems

Heading

DISCRETE SYSTEM <system identifier>

Declarations

INPUT
OUTPUT
TIME
STATE
NEW
TSAMP

INITIAL-section

Computation of auxiliary variables (parametric expressions)
Computation of initial values for state variables
Computation of initial values for output variables
Computation of initial value for the TSAMP-variable

OUTPUT-section

Computation of auxiliary variables
Computation of output variables
Computation of new values for the states
Updating of the TSAMP-variable
Modification of states in continuous subsystems

DYNAMICS-section

Computation of auxiliary variables
Computation of new values for the states
Updating of the TSAMP-variable
Modification of states in continuous subsystems

Table 5. Structure of connecting systems

Heading

CONNECTING SYSTEM <system identifier>.

Declaration

TIME

INITIAL-section

Computation of auxiliary variables (parametric expressions)

CONNECT-section

Computation of auxiliary variables
Computation of input variables

A parameter can be assigned in an arbitrary section and need not be assigned before it is used in an assignment statement.

Initial value of a state variable can be given in two ways, in a constant assignment or by computation in the INITIAL-section. If a constant assignment is used (can be placed in an arbitrary section) it is possible to alter the initial value between the simulations with the command INIT. Initial value need not be given in which case the value zero is assumed.

If no initial value is given for the TSAMP-variable (the variable containing the time for next sampling) in a discrete system the first sampling is performed at the time the simulation starts. The updating of the TSAMP-variable can be done either in OUTPUT- or DYNAMICS-section.

Initial values for the outputs of discrete systems need not be given in which case zero is assumed.

It is possible to modify the states of continuous subsystems from a discrete subsystem, (see example 7).

The computation of derivatives and new values of states is usually done in the DYNAMICS-section but can also be done in the OUTPUT-section. It must, however, then be checked that these variables do not depend on inputs that are undefined at the execution of the OUTPUT-section. Check the warnings which are given at the compilation!

4.3. Elements of the language

The simulation language has six types of language elements:

- system heading
- declaration
- section heading
- assignment statement
- constant assignment
- end statement

These language elements are combined to a continuous, a discrete or a connecting system as the tables 3-5 illustrate.

Each language element constitute one line of the system description. Comments can be written at the end of the lines if they are preceded by a " and empty lines can be inserted to improve layout.

Syntactical descriptions of the different language elements are given in appendix A.

System heading

Each system description begins with a system heading which submits the type and the name of the system. The system heading has three forms:

CONTINUOUS SYSTEM <system identifier>
DISCRETE SYSTEM <system identifier>
CONNECTING SYSTEM <system identifier>

Example

CONTINUOUS SYSTEM MOTOR

Declarations

Declarations are used to specify the types of the variables and are of the form

<type> <simple variable>...

<type> is one of the words INPUT, OUTPUT, TIME, STATE, DER, NEW and TSAMP.

The derivatives of the states in a continuous system are of type DER, the new states in a discrete system are of type NEW and the variable containing the time for the next sampling in a discrete system is of type TSAMP.

The types of declarations which are legal for the different systems are shown in the tables 3-5.

The declarations can be done in an arbitrary order except that the STATE-variables and the corresponding DER- or NEW-variables must appear in the same order.

Several declaration statements of the same type can be used.

Only one variable of the type TIME and TSAMP is allowed in each system.

Parameters and auxiliary variables are not declared. A variable is considered as a parameter if its first appearence is in a constant assignment or in the right part of an assignment statement. A variable is considered as auxiliary if its first appearance is in the left part of an assignment statement.

Examples

```
INPUT U
OUTPUT Y1 Y2
STATE X1 X2
DER DX1 DX2
TIME T
```

Section headings

The section headings are used to define the beginning of the different algorithms of the system description. There are four section headings:

```
INITIAL
OUTPUT
DYNAMICS
CONNECT
```

The sections must be given in the order INITIAL, OUTPUT, DYNAMICS for continuous and discrete systems and INITIAL, CONNECT for connecting systems. If a section is empty then its heading can be omitted.

If no section heading preceeds the first assignment statement or constant assignment then OUTPUT-section is assumed for continuous and discrete systems and CONNECT-section for connecting systems.

Assignment statements

The assignment statements are of the same type as in ALGOL-60 [3]. Below is given a list of deviations.

- in a <number> $_{10}$ is replaced by E
- <number> can not be just an <exponent part>
- variables with index are not allowed
- a variable can contain a system identifier between square brackets

- the operators ÷ ≤ ≠ ≥ ⊃ ≡ are not allowed
- logical or, and, not are written OR, AND, NOT

In a discrete or connecting system a reference to a variable outside the actual system is done by placing the system identifier of the referenced system between square brackets after the identifier of the variable.

Spaces must be inserted between <identifier> and <unsigned integer> and can be inserted everywhere except in an <identifier> or in a <number>.

All computations are performed using floating arithmetic.

The result of a Boolean expression is represented by the number 1.0 if it is true and by 0.0 if it is false. A Boolean expression is considered true if the corresponding value is greater that or equal to 0.5 else false. This fact should be remembered when using parameters as switches.

A list of available functions is given in table 6.

Examples

```
DX1=X2
DX2=-(A-2*Q*COS(2*T))*X1
V=5.75*12 / (.08-2.59E-4) + 2E5
DV  = -U-EPS*(1-(IF ABS(U)<SQRT(2) THEN 2-U*U ELSE 0))*V

U1[SYST] = (IF T<1 OR T>2 THEN 0 ELSE 1) + LEVEL
U[S1] = Y[S2]
Q[SYST5] = IF PAR THEN FI[CONT] ELSE FI[DISCR]
U[S] = IF T<1 THEN LEV1 ELSE IF T<T2 THEN LEV2 ELSE LEV3
U[S] = IF T<T2 THEN (IF T<T1 THEN LEV1 ELSE LEV2) ELSE LEV3
P[SYST5] = IF IF SW THEN T<10 ELSE T>5 THEN 0 ELSE 1
```

Table 6. Available functions

SQRT(X)	square root of x (x ≥ 0)
EXP(X)	exponential function of x
LN(X)	natural logarithm of x (x > 0)
LOG(X)	logarithm (base 10) of x (x > 0)
SIN(X)	sine of x (x in radians)
COS(X)	cosine of x (x in radians)
TAN(X)	tangent of x (x in radians)
ATAN(X)	arctanget of x (result in radians in the interval (-Π/2,Π/2])
ATAN2(X,Y)	arctanget of x/y (result in radians in the interval (-Π,Π])
ABS(X)	absolute value of x
SIGN(X)	the sign of x: +1.0 if x > 0 0.0 if x = 0 -1.0 if x < 0
INT(X)	integer part of x
MOD(X,Y)	the remainder when dividing x by y (X - INT(X/Y)*Y)
MIN(X,Y)	minimum of x and y
MAX(X,Y)	maximum of x and y

Constant assignments

The constant assignment is used to give values to parameters and initial values to states. The form is

<simple variable>:<number>

Examples

X:-10
ALPHA:1.5E-3

End statement

The end statement ends the system description and has the form

END

4.4. Diagnostics during the compilation

During the compilation of the system descriptions there is an extensive error checking including tests that

- the system description is syntactically correct
- the system description is made in accordance with the tables 3-5
- all variables are given values

If an error is detected a detailed error message is given.

Warnings about undefined inputs are given when compiling the connecting system if not all inputs to a system are assigned when an output from that system is used. The user then has to check that all variables which are computed in the OUTPUT-section of the system are algebraically independent of the undefined inputs. If this is not the case the inputs to the systems have to be defined in an other order.

A message, indicating where the calls for output computation is inserted, is also given when compiling the connecting system (see example 5).

5. SYSTEM DESCRIPTION IN FORTRAN

The description of a continuous or discrete subsystem can be made as a FORTRAN-subroutine having a special structure. The structure of the subroutine is given in table 7. It consists of eight parts which is executed independently. By giving different values (1,2,...,8) to the variable IPART, SIMNON decides which part that is to be executed.

A description of the different parts is given below.

Part 1: Identification of the system.

SIMNON has to know the type (continuous or discrete) and the name of each system. These data is transfered in part one by a call to the subroutine IDENT:

```
CALL IDENT (STYPE,SYSID)
```

with

```
STYPE  - system type
         'CONT': continuous system
         'DISCR':discrete system
SYSID  - system identifier
```

Example

```
CALL IDENT('CONT','MOTOR')
```

Table 7. Structure of FORTRAN systems

```
      SUBROUTINE SYSTEM
C
      COMMON /DESTIN/ IDUM, IPART
      COMMON /TIME/ T
C
      GOTO (1,2,3,4,5,6,7,8), IPART
C
 1    CONTINUE
C                 IDENTIFICATION OF THE SYSTEM
      RETURN
C
 2    CONTINUE
C                 DECLARATION OF VARIABLES
      RETURN
C
 3    CONTINUE
C                 ASSIGNMENT OF CONSTANTS
      RETURN
C
 4    CONTINUE
C                 INITIAL-SECTION
      RETURN
C
 5    CONTINUE
C                 OUTPUT-SECTION
      RETURN
C
 6    CONTINUE
                  DYNAMICS-SECTION
      RETURN
C
 7    CONTINUE
C                 COMPUTATION ON ACCEPTED VALUES
      RETURN
C
 8    CONTINUE
C                 FINAL COMPUTATIONS
      RETURN
      END
```

Part 2: Declaration of variables

The communication, between SIMNON and the subroutine, concerning variables for states, parameters, etc, is not done using formal parameters or COMMON-blocks. It has been necessary to communicate using absolute addresses of the variables. References to the variables using symbolic names are allowed during the interaction, i.e. the identifier of each variable must be available to the program. To transfer this information the variables must be declared by calling special subroutines.

The declaration subroutines are INPUT, OUTPUT, STATE, INIT, DER, NEW, TSAMP, PAR and VAR. They have all two arguments:

VAR - the variable

VARID - the variable identifier (hollerith variable)

The first argument (the variable itself) is used by the program to calculate the absolute address of the variable. The second argument transfers the corresponding identifier. The different subroutines indicate the type of each variable.

The subroutine INIT declares a variable for the initial value of a state. The subroutines PAR and VAR declare parameters and auxiliary variables.

Example

The statement

```
CALL STATE(X,'X')
```

declares the variable X as a state variable with identifier X.

The declarations can be done in an arbitrary order except that the STATE-variables and the corresponding DER- and INIT-variables must be declared in the same order. The same rule applies to STATE-, NEW- and INIT-variables.

Only one variable of type TSAMP is allowed in each system.

The simulation time is available in the COMMON-block TIME.

Note that only REAL variables can be declared.

Vectors can be declared by calling the subroutines INPUTV, OUTPUV, STATEV, INITV, DERV, NEWV, PARV and VARV. These subroutines have three arguments:

VEC	- the vector
N	- number of components
FCHARS	- the first characters of the identifiers (hollerith variable)

Example

The statement

```
CALL DERV(DX,5,'DX')
```

declares DX(1),DX(2),...,DX(5) as derivatives with identifiers DX1,DX2,...,DX5.

Part 3: Assignment of constants

This part of the subroutine corresponds to the constant assignments in the simulation language, i.e. assignment of PAR- and INIT-variables should be done here. These variables can then be changed with the PAR- and INIT-commands. PAR- and INIT-variables need not be assigned since the value zero is assigned automatically before the part 3 is called.

Part 4, 5 and 6: INITIAL-, OUTPUT- and DYNAMICS-sections.

The contents of these parts are given in the tables 3 and 4 (page 4.2 and 4.3).

Part 7: Computation on accepted values

This part of the subroutine is called every time the integration routine has found new values belonging to the solution of the differential equations. If extensive computations are to be done on variables that are used only for plotting these computations can be done here in order to increase the simulation speed. Storing values for the implementation of systems having time delays is also done in this part.

Part 8: Final computations

In this part it is possible to make final computations or e.g. to close files on mass storage after the simulation is completed.

Part 1 of the subroutine is called at the initialization of SIMNON to display information about systems included. The parts 1, 2 and 3 are called when the command SYST is used and the parts 4-8 are called when the command SIMU is used. The part 4 is called once before the simulation. The parts 5 and 6 in continuous systems are called every time the integration routine needs the values of the derivatives and in discrete systems at the sampling instances. Every time the integration routine has accepted a point belonging to the solution then part 7 is called. Part 8 is called once after the simulation.

The initial value of a state variable can be given in two ways, via an INIT-variable or by direct assignment in the INITIAL-section. If an INIT-variable is used it is possible to change the initial value between the simulations using the command INIT. An INIT-variable must be declared in both cases.

If no initial value is given for the TSAMP-variable in a discrete system the first sampling is performed at the time the simulation starts.

Initial values for the outputs from discrete subsystems need not be assigned since zero is assigned automatically.

The computation of derivatives and new values of states can be done in OUTPUT-section. It must however then be checked that these variables do not depend on inputs that are undefined at the execution of the OUTPUT-section. Check the warnings which are given during the compilation!

The COMMON-block USER contains two logical variables LSTOP and LDARK:

```
COMMON/USER/LSTOP,LDARK
```

If LSTOP is set to .TRUE. the simulation will be stopped (the part 8 is first called). If LDARK is set to .TRUE. the plot routine will not draw visible lines to the next plot points.

The subroutines that describe the systems can have almost arbitrary names. As a link between SIMNON and the subroutines describing systems there is a subroutine SYSTS. This subroutine has to contain calls to the user subroutines. The structure of the subroutine SYSTSis given in table 8. By varying the variable ISYST, SIMNON decides what system should be called. The allowable range of ISYST should be assigned the variable NSYST.

Table 8. Structure of the subroutine SYSTS

```
      SUBROUTINE SYSTS
C
      COMMON /DESTIN/ ISYST,IDUM
      COMMON /NSYSTS/ NSYST
C
      NSYST = range-of-ISYST
      GOTO (1,2,...), ISYST
C
 1    CALL system 1
      RETURN
C
 2    CALL system 2
      RETURN
C
      .
      .
      .
      END
```

Example 3: Triode oscillator (continued)

As an example of how the system description can be done using a FORTRAN-subroutine consider the triode oscillator from example 1. The system description can be done as shown in listing 1 (compare with the system description in the simulation language, page 2.4).

The linking subroutine SYSTS can be written as shown in listing 2.

```
      SUBROUTINE TRIODE
C
      COMMON /DESTIN/ IDUM,IPART
C
      GO TO(1,2,3,4,5,6,7,8),IPART
C
 1    CALL IDENT('CONT','TRIOD')
      RETURN
C
 2    CALL STATE(U,'U')
      CALL STATE(V,'V')
      CALL DER(DU,'DU')
      CALL DER(DV,'DV')
      CALL INIT(UI,'UI')
      CALL INIT(VI,'VI')
      CALL PAR(EPS,'EPS')
      CALL VAR(G,'G')
      RETURN
C
 3    UI=0.25
      VI=0.
      EPS=1.
      RETURN
C
 4    SQRT2=SQRT(2.0)
      RETURN
C
 5    RETURN
C
 6    G=0.
      IF(ABS(U).LT.SQRT2) G=2.-U*U
      DU=V
      DV=-U-EPS*(1-G)*V
      RETURN
C
 7    RETURN
 8    RETURN
C
      END
```

Listing 1.

```
      SUBROUTINE SYSTS
C
      COMMON /DESTIN/ ISYST,IDUM
      COMMON /NSYSTS/ NSYST
C
      NSYST=1
C
      CALL TRIODE
      RETURN
C
      END
```

Listing 2.

6. INTERACTION

The use of SIMNON is controlled with the aid of commands. The program reads the commands either from the terminal or from a file on mass storage. A command begins with one of a set of words that specifies what action the program should take. This word can be followed by arguments that can be identifiers, numbers or delimiters. If the command input is from the terminal the program signifies its readiness to accept a command by typing a >.

SIMNON is imbedded in a general interactive language, called INTRAC. In this language it is possible to store sequences of commands in a so called MACRO (file on mass storage). Each time the sequence should be executed the name of the MACRO is used as a new command. It is also possible to introduce repetitive loops and make tests and jumps among the commands. Chapter 8 contains one example which uses a MACRO.

This chapter contains a description of the structure of the available commands and their use. When describing the structure of the commands the notations listed below are used.

|	or (separates terms in a list from which one and only one must be chosen)
{ }	groups terms together
[ ]	groups terms together and denotes that the group is optional
{ }*	denotes the repetition of a choice in the group one or more times
[ ]*	denotes the repetition of a choice in the group none or more times

The syntax for identifiers, variables and numbers is given in appendix A.

The arguments must not contain spaces but they may be separated by an arbitrary number of spaces. If no doubts can arise the arguments need not be separated by any space (e.g. when one of the arguments is a delimiter).

If a command is given which is syntactically incorrect or contains bad values of the arguments the execution of the command is aborted and an error message is given.

If a reference should be made to a variable in a subsystem which has a unique identifier then only the identifier need to be given otherwise the reference must be specified by givning the identifier of the actual subsystem within square brackets. STATE- and INIT-variables are special since they can have the same identifier even within one subsystem. This case is treated in a way that the STATE-variable always is reached except for the first argument of the INIT-command.

The description of each command contains the command structure, the actual description and some examples.

Defining the system

SYST{<identifier>}*[-EDIT]

The system that is to be simulated is defined by the command SYST.

Each <identifier> is either the name of an external system (system described in FORTRAN) or the file name for a system description file. If only one <identifier> is given it is assumed that the corresponding system is either a continuous or a discrete system with no inputs. If several <identifier>:s are given the last <identifier> must be the name of the file containing the description of the connecting system.

To determine if an <identifier> is the <system identifier> for an external system or a file name, the subroutine SYSTS is called with different values of ISYST and with IPART = 1. The <identifier> is then compared with the <system identifier>:s given by the calls to the subroutine IDENT.

The external systems which should be activated are then called with IPART = 2 to allow declaration of variables. The remaining <identifier>:s are assumed to be file names for system description files and therefore the editor is initialized with these file names in turn. The editor can now be used for generation or modification of the files as described in chapter 7.

The compliation of the system description is done in the following way. When the editor has accepted a line and is going to save it on mass storage it calls the compiler. If the compiler does not accept the line an error message is written and the editor will immediately be forced to EDIT-mode. The erroneous line can then be corrected.

When the editing of a file is finished the command E (exit) is used. If more files have been given in the command SYST the editor will be initialized with the next file else the external subsystems are called with IPART = 3 for assignment of values of parameters and initial values of states. The activation of the system is then ready.

The editor is usually initialized in a way such that it automatically exists if the file exists and if the file does not contain any detectable errors. This implies that if all files in a system description are correct then no editor commands are needed. If one wants to modify a file that is correct, the option EDIT must be included in the SYST command.

During the compilation of the connecting system warnings about undefined input variables are given if the switch WARN is ON (see the command TURN). If the switch COMPU is ON messages are written to indicate when the execution of the OUTPUT-sections are performed.

The editor can not exit from a file which contains errors. If an unrecoverable error is made (e.g. a subsystem in the SYST command is forgotten) it is necessary to leave the editor and the compilation using the command LEAVE.

Examples

```
SYST TRIOD
SYST TRIOD-EDIT
SYST SYS REG CONN
SYST SYS REG CONN-EDIT
```

Selecting variables to be plotted

PLOT[{<variable>}*[(<variable>)]]

If certain variables should be plotted during a simulation those variables must be selected using the command PLOT before the simulation is demanded.

The variables are usually plotted versus the simulation time but they can also be plotted versus an arbitrary variable. This variable is given within parenthesis at the end of the command.

The plotting facility is turned off if the command PLOT is given with no argument. If the switch PLCOM is ON and axes are drawn on the display the PLOT-command is written on the display.

Examples

```
PLOT X1
PLOT U V
PLOT U (V)
PLOT Y[SYST1] Y[SYST2]
PLOT Y[SYST1] U (V)
PLOT
```

Drawing axes

AXES[{H|V}<minimum><maximum>[{H|V}<minimum><maximum>]]

This command determines the scales of the axes and draws axes.

The scales of the axes are specified by giving minimum value and maximum value for each axis. The program then selects standardized scales. The values are preceeded by H (horizontal) or V (vertical) to indicate which axis that is concerned. If no values are given for some axis then the same values are used as the last time the command AXES was given.

If the switch PLCOM is ON the last PLOT-command is written on the display.

Examples

```
AXES H 0 1000 V -0.5 2.5
AXES V -1.5 5 H 0 500
AXES H -50 50
AXES V -7.5 -5
AXES
```

Simulating

SIMU[<start time><stop time>[<time increment>]][-{CONT|MARK}
[CONT|MARK]][/<file name>]

This command demands simulation of the system defined by the previous SYST command.

The simulation is performed according to table 2 (page 3.11). The initialization is done in the following way. The simulation-routines sets the simulation time to <start time>, the states to their initial values, the output-variables and auxiliary variables (VAR-variables) in discrete systems to zero and the TSAMP-variables to <start time>. Then the INITIAL-sections of all systems are executed possibly altering the values of the states, the outputs from discrete systems and the TSAMP-variables.

The maximal time increment for a variable step size integration routine or the time increment for a fixed step size integration routine is entered as <time increment>. If the <time increment> is zero, an increment of one hundredth of the length of the simulation interval is used.

To continue a simulation the option CONT should be used. The initialization mentioned above is then inhibited. The continued simulation will be performed over a time interval of length <stop time> - <start time> beginning at the time the last simulation was stopped.

If plotting of certain variables has been demanded they will be plotted during the simulation. The plotted curves will be marked with digits corresponding to the order of the variables in the PLOT-command if the option MARK is used.

The variables which are selected by the STORE-command are stored on a file named <file name>.

If any of the arguments <start time>, <stop time>, <time increments> or <file name> is omitted from the command the same values are used as the last time the command SIMU was given (values at start up: 0 1 0 STORE).

The integration routine (if needed) is selected by the command ALGOR and the error bound for variable step size integration routines is entered by the command ERROR.

Examples

```
SIMU 0 25
SIMU -2.5 2.5/SIM1
SIMU -CONT
SIMU 10 20 0.02 - MARK CONT/SIM2
SIMU
```

Changing a parameter value

PAR<variable>:{<number>|<variable>}

This command is used to change the values of parameters.

The new value of a parameter is specified either by a number or as the value of a variable in the system.

Examples

```
PAR P:100
PAR BETA[SYST]:-1E-6
PAR P1[SYST1]:P1[SYST2]
```

Changing the initial value of a state variable

INIT<variable>:{<number>|<variable>}

This command is used to change initial values of state variables.

The new initial value is specified either by a number or as the value of a variable in the system.

Note that if the initial value of a state variable is computed in the INITIAL-section then it is not possible to use this command.

Examples

```
INIT X:-5.5
INIT F1[SYST]:2
INIT X:X
INIT X1[SYST1]:V[SYST2]
```

Selecting variables to be stored

STORE[{<variable>}* [-ADD]]

This command selects variables to be stored during the simulations.

The selected variables are stored on a file together with the simulation time every time there are available values. The stored variables can be plotted by the command SHOW after the simulation.

The option ADD is used to extend the set of variables to be stored. The facility to store variables is turned off when the command SYST is used or if the command STORE is given without arguments.

The file is named in the SIMU-command.

Examples

```
STORE X1
STORE X2 X3 - ADD
STORE Y[SYST1] Y[SYST2]
STORE
```

Plotting stored variables

SHOW{{<variable>}* [(<variable>)][-MARK]|-LIST}[/<file name>]

This command plots variables which have been stored on a file.

The variables are plotted versus the variable within parenthesis. If this variable is omitted the plot will be made versus the simulation time.

The plotted curves will be marked with digits corresponding to the order of the variables in the SHOW-command if the option MARK is used.

A list of the variables stored on the file will be given on the display if the option LIST is given.

If the file name is omitted then the last file name entered by the SIMU-command will be used.

Axes must be drawn using the command AXES before the SHOW-command is given.

Examples

```
SHOW X1 Y[SYST1]
SHOW X2 (X1)
SHOW X1 X2 - MARK/SIM1
SHOW -LIST
SHOW -LIST/SIM2
```

Displaying variables

DISP[({DIS|TP|LP})][<variable>]*

This command is used to display the current values of variables on selected device.

If no variable is named then all variables will be displayed.

The device is specified as the display (DIS) (default), the teleprinter (TP) or the line printer (LP).

Example

If the triode oscillator (page 2.4) has been activated then the command

```
DISP
```

would generate the printout:

```
                              CONTINUOUS SYSTEM TRIOD
STATE : U        2.23021          V       0.711000
INIT  : U       0.250000          V       0.000000
DER   : DU      0.711000          DV      -2.94121
PAR   : EPS      1.00000
VAR   : G       0.000000
```

The command

```
DISP U V EPS
```

would generate the printout:

```
U=2.23  V=0.711  EPS=1.
```

Examples

```
DISP
DISP(LP)
DISP(TP) X1 X2 FI[SYST1]
```

Listing files

LIST[({DIS|TP|LP})]{<file name>}*

This command is used to transfer symbolic files from mass storage to different listing devices.

The device is specified as the display (DIS) (default), the teleprinter (TP) or the lineprinter (LP).

Examples

```
LIST FIL1
LIST(LP) FIL1 FIL2
```

Editing files

EDIT<filename>

This command enables the user to create and modify files containing symbolic text.

The manipulation of the file is performed using subcommands as described in chapter 7.

All types of symbolic files can be edited, e.g. system description files and macro files. Note that if a system description is modified using this command then the command SYST must be used to compile the system description afterwards.

Example

```
EDIT FILE
```

Saving parameter values and initial values on a file

SAVE<file name> [<system identifier>][-{PAR|INIT}]

This command is used to save the current parameter values and initial values of states on a file. The values can later be restored from the file using the command GET. The format of the generated file is described under the GET-command.

The command is useful for example to save parameter values before a modification of the system description is done. A recompilation implies that the parameters get the values given in the system description.

If only a file name is given as argument all parameter values and all initial values in all subsystems are saved. If values from only one subsystem should be saved the system identifier for that subsystem must be given. If only parameter values or only initial values should be saved the options PAR or INIT should be used.

Examples

```
SAVE FILE1
SAVE FILE2 SYST2
SAVE FILE3 SYST2-PAR
SAVE FILE4-INIT
```

Getting parameter values and initial values from a file

GET<filename>

This command is used to get values for parameters and initial values for states from a file. The file is an ordinary text file and can be generated using the command SAVE and can be modified by the editor.

The lines in the file are grouped in the following way. The first line of a group is of the form:

[<system identifier>]

telling to what subsystem the following lines in the group apply. The rest of the lines in the group are of the form

<identifier>:<number>

Specifying values for parameters and initial values for states of the actual subsystem.

Example

```
[SYST1]
A:5
X1:-2.5
P:1E-6
[SYST2]
G:1
```

The parameters and the states that are not specified are not affected.

If an erroneous line is detected during the reading of the file, an error message is printed together with the erroneous line and the execution of the command is aborted.

Example

```
GET FILE1
```

Selecting integration routine

ALGOR{HAMPC|RK|RKFIX}

This command specifies which integration routine should be used.

An integration routine is needed only if the active system contains continuous subsystems. There are at present three different integration routines available. Short descriptions of the routines are given below.

HAMPC: Hammings predictor corrector method [6]

This routine uses Hammings modified predictor corrector method as the main method. A fourth order Runge-Kutta method is used to compute starting values for the predictor corrector method. The routine uses variable step size and is able to handle discontinuities in the derivatives.

RK: Runge-Kutta method [2]

This routine uses the classical fourth order Runge-Kutta method with variable step size. The routine can handle discontinuities in the derivatives.

RKFIX: Runge-Kutta method with fixed step size

This routine uses the classical fourth order Runge-Kutta method with fixed step size. The <time increment> in the SIMU-command must be chosen very carefully to ensure an accurate solution.

The HAMPC routine is default.

Examples

```
ALGOR RK
ALGOR HAMPC
```

Choosing error bound for the integration routine

ERROR<error bound>

This command enters the error bound used for the step size policy in variable step size integration routines.

The HAMPC routine tests the estimated absolute local error against the error bound to determine the step size policy. The RK routine divides the estimated absolute local error with $\max(1, \|x\|_2)$ before testing. The weighting factors when computing the estimated error are equal so the state variables must be scaled appropriately.

Default value of the error bound is 0.001.

Example

```
ERROR 1E-5
```

Modifying switches

TURN {WARN|COMPU|PLCOM|DARK}{ON|OFF}

By this command switches can be turned on or off.

The actions when the switches are ON are listed below.

WARN Warnings about undefined input variables will be given during the compilation.

COMPU Messages are written during compilation of the connecting system indicating when the executions of the OUTPUT-sections are performed.

PLCOM The PLOT command will be written as a heading on the display when axes are drawn.

DARK The plotted curves will not be connected at the sampling instances.

Default values for the switches are ON except for DARK.

Examples

```
TURN WARN ON
TURN COMPU OFF
```

Leaving SIMNON

STOP

This command stops the execution of SIMNON.

Example

```
STOP
```

7. THE EDITOR

The editor enables the user to create and modify files containing symbolic text.

The interaction between the user and the editor is done in one of two modes, EDIT-mode or INPUT-mode. In EDIT-mode the input lines are interpreted as commands for certain editing actions.

The editor works on a line-by-line basis thus, at any time, the user has only one line, the "current line", available for modification. By the use of commands the user can specify which line should be made current. This is done either by specifying the location of the new line, relative the current line or by demanding the editor to search for a specified string.

If the editor is in INPUT-mode the input lines are just added to the file after the current line.

The file is processed sequentially from the beginning to the end.

When initializing the editor with a certain filename the editor will, if the file already exists, start in EDIT-mode and write the message

```
EDIT
>
```

The > signifies that the editor waits for a command. If the named file does not exist the editor will start in INPUT-mode and write the message

```
FILE NOT FOUND: filename
INPUT
```

and wait for lines to be typed.

To change the mode of the editor an empty line should be typed.

Description of the commands

Each command has either a positive integer or a string of characters as argument or no argument. The command word and the argument must be separated by a space. Strings are defined to begin after the space. If the integer is omitted the value is assumed to be one. Comments can be given after a double quote (") if the command does not contain a string.

In the description of the commands the symbols [ ] denote that what is inside is optional. The value of the integer is denoted by n.

A description of each command follows below. It contains the command structure, the actual description and examples of command structure.

Next

N[EXT][<integer>]

Makes the line that is n lines below the current line available for modification.

```
N
N 5
NEXT
```

Print

P[RINT][<integer>]

Prints n lines including the current line on the terminal and makes the last line printed available for modification.

```
P
P 2
PRINT 5
```

Locate

L[OC] <string>

Searches down the file (not including the current line) for a line that contains the specified string and makes that line current.

```
L  IF
LOC +X1
```

Find

F[IND] <string>

Searches down the file (not including the current line) for a line that begins with the specified string and makes that line current.

```
F  A =
FIND ABC
```

Delete

D[EL][<integer>]

Deletes n lines including the current line and makes the following line current.

```
D
D  5
DEL
```

Overlay

O[VERL][<integer>]

Deletes n lines including the current line and changes mode to INPUT.

```
O
OVERL
```

Retype

R[ETYP] <string>

Replaces the current line with the specified string.

```
R  A = B
RETYP C = D+E
```

Insert

I[NS] <string>

Inserts the string as a line following the current line and makes that line current.

```
I  A=IF B THEN C ELSE D
INS " COMMENT
```

Append

A[PPND] <string>

Appends the string on the current line.

```
A  + 1
APPND " COMMENT
```

Change

C[HANG] <quote><string><quote><string>[<quote>]

Changes the characters in the first occurance of the first string to the characters in the second string in the current line. The <quote> can be any characters.

```
C /ALPHA/BETA/
C ./.*.
C //"/
CHANG /+1//
```

Bottom

B[OTT]

Goes to the end of the file.

```
B
BOTT
```

Top

T[OP]

Goes to the beginning of the file.

```
T
TOP
```

Exit

E[XIT]

Closes the file and makes exit.

```
E
EXIT
```

Display

DIS[ON|OFF]

Sets the display flag ON or OFF. When DIS is ON the current line will successively be displayed as the editing goes down the file. If DIS is OFF the current line will automatically be printed on the terminal only after the execution of the find, locate and change commands. Default value of the argument is ON.

```
DIS
DIS ON
DIS OFF
```

Leave

LEAVE

Exits leaving the file as it was when the last TOP-command was executed or when the editor was initialized.

```
LEAVE
```

Example. Using the editor

This example will illustrate the use of the editor to modify a file.

The original content of the file was

```
TEST FILE
---------
THE PURPOSE IS TO ILLUSTRATE
THE USE OF THE EDITOR
TO CORRCT A FILE.
```

After modification the content of the file became

```
TEST FILE
---------
THIS FILE CONTAINS SYMBOLIC TEXT.
THE EDITOR
CAN BE USED TO
CREATE OR
TO CORRECT A FILE.
```

The dialogue when performing the modification is shown below.

```
   EDIT
  >P 6                    " PRINT THE FILE
TEST FILE
---------
THE PURPOSE IS TO ILLUSTRATE
THE USE OF THE EDITOR
TO CORRCT A FILE.
  >T                      " TOP - GO TO THE BEGINNING OF THE FILE
   EDIT
  >N 2                    " NEXT - ADVANCE TWO LINES
  >                       " INSERT A LINE
  >I THIS FILE CONTAINS SYMBOLIC TEXT
  >                       " APPEND A . ON CURRENT LINE
  >A .
  >P
THIS FILE CONTAINS SYMBOLIC TEXT.
  >N
  >P
THE PURPOSE IS TO ILLUSTRATE
  >D                      " DELETE CURRENT LINE
  >P
THE USE OF THE EDITOR
  >                       " RETYPE CURRENT LINE
  >R THE EDITOR
  >                       " CHANGE MODE TO INPUT
  >
   INPUT
  CAN BE USED TO
  CREATE OR

   EDIT
  >N
  >P
TO CORRCT A FILE.
  >                       " CHANGE 'RC' TO 'REC'
  >C /RC/REC/
TO CORRECT A FILE.
  >T
   EDIT
  >P 8
TEST FILE
---------
THIS FILE CONTAINS SYMBOLIC TEXT.
THE EDITOR
CAN BE USED TO
CREATE OR
TO CORRECT A FILE.
  >E                      " EXIT FROM THE EDITOR
```

8. EXAMPLES

This chapter contains three complete examples, the simulation of a servo system, the generation of a phase plane (the macro facility) and the solution of a two point boundary value problem.

Example 5. Servo system

Consider the servo system in figure 3.

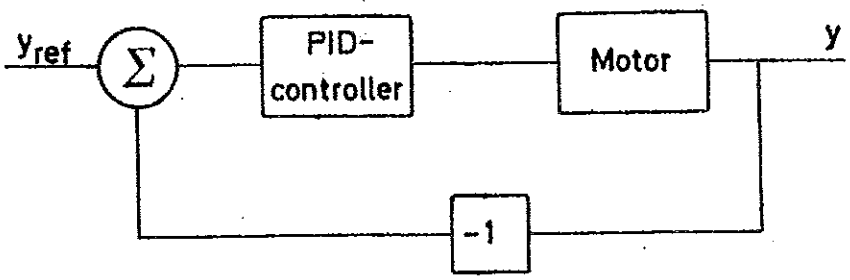


Figure 3.

The system is composed of two subsystems, a DC-motor and a PID-controller. It will be simulated with both a continuous and a discrete PID-controller.

Model for the motor

The equations for the motor are

$$i = (u - K_m\dot{\theta})/R$$
$$M_e = K_m i$$
$$\ddot{\theta} = M_e/J$$
$$y = C_T\theta$$

where u is the input voltage, i the current, M_e the electrical torque, θ the rotor angle and y the output voltage from a transducer. K_m, R, J and C_T are parameters.

The description of the motor as done in the simulation language is shown in listing 3.

Note that the model of the motor is linear and characterized by

$$y(t) = \frac{5.3}{p(1 + 0.1p)}\, u(t)$$

with the given values of the parameters (p denotes the differentiation operator).

```
CONTINUOUS SYSTEM MOTOR
" A MODEL FOR A DC-MOTOR WITH AMPLIFIER AND TRANSDUCER.
INPUT U                 " INPUT VOLTAGE TO AMPLIFIER
OUTPUT Y                " OUTPUT VOLTAGE FROM TRANSDUCER
STATE TH THDOT
DER DTH DTHDT

OUTPUT
Y=CT*TH                 " TRANSDUCER
CT:0.033                " [V/RAD]

DYNAMICS
I=(U-KM*THDOT)/R        " CURRENT
ME=KM*I                 " ELECTRICAL TORQUE

DTH=THDOT
DTHDT=ME/J

KM:6.2E-3               " [NM/A]
R:5.3                   " [OHM]
J:7.5E-7                " [KG M↑2]

END
```

Listing 3.

A continuous PID-controller

One version of a continuous PID-controller is defined by

$$u = (G + \frac{1}{T_i p})(y_{ref} - y) - \frac{T_d p}{1+\frac{G_d}{T_d}p} y$$

Note that the "differentiation" is performed on the measured variable itself, not on the control error.

The equations for this controller are

$$e = y_{ref} - y$$

$$p = G\ e$$

$$\frac{di}{dt} = e/T_i$$

$$\frac{dx}{dt} = -\frac{G_d}{T_d}(x-y)$$

$$d = -G_d(y - x)$$

$$u = p + i + d$$

The description of the controller as done in the simulation language is given in listing 4.

A discrete PID-controller

One version of a discrete PID-controller is defined by

$$u = (G + \frac{\Delta t}{T_i(1-q^{-1})})(y_{ref} - y) - \frac{T_d}{\Delta t}(1 - q^{-1})y$$

where q denotes the forward shift operator ($q^{-1}y(t) = y(t-\Delta t)$) and Δt is the sampling interval.

```
CONTINUOUS SYSTEM CPID

INPUT YREF Y
OUTPUT U
STATE I X
DER DI DX

OUTPUT
E=YREF-Y

P=G*E
D=-GD*(Y-X)

U=P+I+D

DYNAMICS
DI=E/TI
DX=-GD/TD*(X-Y)

G:1
TI:1E10
GD:0
TD:1

END
```

Listing 4.

The equations for the discrete PID-controller are

$$e = y_{ref} - y$$

$$p = G\ e$$

$$i(t + \Delta t) = i + \Delta t\ e/T_i$$

$$x(t + \Delta t) = y$$

$$d = -\frac{T_d}{\Delta t}(y - x)$$

$$u = p + i + d$$

The description of this controller as done in the simulation language is given in listing 5.

```
DISCRETE SYSTEM DPID

INPUT YREF Y
OUTPUT U
STATE I X
NEW NI NX
TIME T
TSAMP TS

OUTPUT
E=YREF-Y

P=G*E
D=-TD*(Y-X)/DT

U=P+I+D

DYNAMICS
NI=I+E/TI*DT
NX=Y

TS=T+DT

G:1
TI:1E10
TD:0

DT:1

END
```

Listing 5.

Connecting the subsystems

The three subsystems are now described and the connection is to be done. In fact only two of the subsystems will be used at each simulation, one PID-controller and the motor. In order to make it possible to change the controller quickly, it is, however, practical to include both of the controllers in the system and then select the actual controller with a switch.

As reference input a step will be used. Both of the controllers have algebraic dependence between the inputs and the output. The output from the motor is not algebraically dependent on its input. The connections in the feedback loop must then start with connecting the output from the motor with the corresponding inputs of the controllers. The inputs of the controllers are now assigned so their outputs can be used. The switch to determine actual regulator is now introduced.

The listing of the connecting system is shown in listing 6.

```
CONNECTING SYSTEM SERVO

TIME T

YR=IF T<0.5 THEN 0 ELSE 1
YREF[CPID]=YR
Y[CPID]=Y[MOTOR]
YREF[DPID]=YR
Y[DPID]=Y[MOTOR]
U[MOTOR]=IF CONT THEN U[CPID] ELSE U[DPID]
CONT:1

UP=A*U[MOTOR]+B
A:1
B:0

END
```

Listing 6.

Simulation

During the simulation four cases will be studied, continuous and discrete P-controller and continuous and discrete PD-controller.

In order to be able to show the control-signal adequately, a new variable U_p has been introduced in the connecting system:

$$U_p = 0.5u + 1.5$$

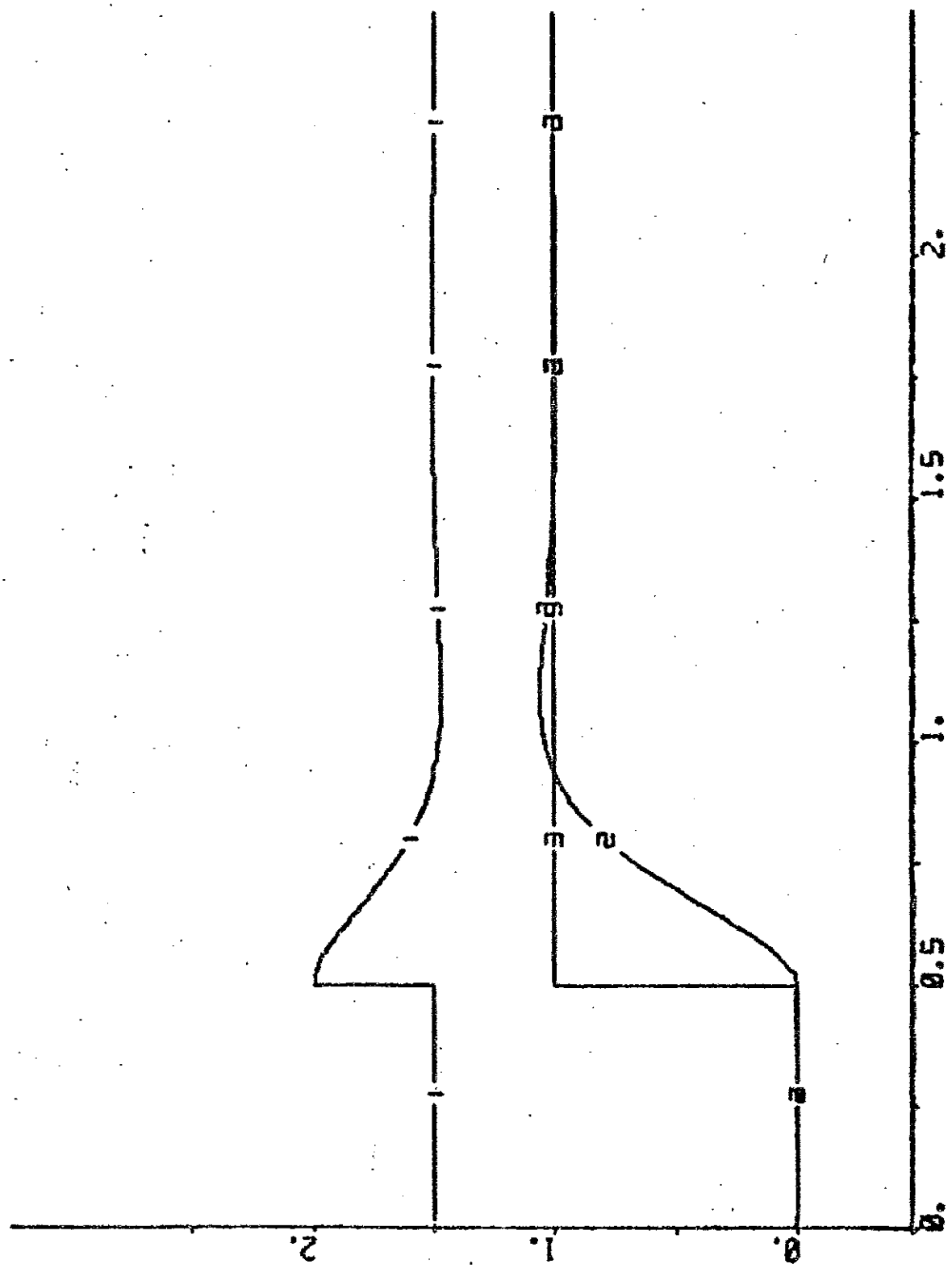
The dialogue when performing the simulation is shown below.

```
>SYST MOTOR CPID DPID SERVO - EDIT
   EDIT
  >E     " EXIT FROM MOTOR
   EDIT
  >E     " EXIT FROM CPID
   EDIT
  >E     " EXIT FROM DPID
   EDIT
  >DIS ON
  >B
  >      " SEE FIGURE 4. CHECK THE WARNINGS!
  >E     " EXIT FROM SERVO
>PLOT UP Y[MOTOR] YR
>AXES H 0 2.5 V -0.1 2.5
>       " SCALE AND MOVE THE CONTROL SIGNAL
>PAR A:0.5
>PAR B:1.5
>
>PAR DT:10
>       " CONTINUOUS P-CONTROLLER
>SIMU 0 2.5 - MARK
>DISP G[CPID] TD[CPID] GD[CPID]
>       " SEE FIGURE 5
>
>       " CONTINUOUS PD-CONTROLLER
>PAR G[CPID]:3
>PAR TD[CPID]:0.08
>PAR GD[CPID]:5
>AXES
>SIMU
>DISP G[CPID] TD[CPID] GD[CPID]
>       " SEE FIGURE 6
>
>       " DISCRETE P-CONTROLLER
>PAR CONT:0
>PAR DT:0.05
>AXES
>SIMU
>DISP G[DPID] TD[DPID]
>       " SEE FIGURE 7
>
>       " DISCRETE PD-CONTROLLER
>PAR G[DPID]:3
>PAR TD[DPID]:0.15
>AXES
>SIMU
>DISP G[DPID] TD[DPID]
>       " SEE FIGURE 8
>
```

```
CONNECTING SYSTEM SERVO

TIME T

YR=IF T<0.5 THEN 0 ELSE 1
YREF[CPID]=YR
Y[CPID]=Y[MOTOR]
   PRECEDED BY EXECUTION OF OUTPUT-SECTION OF MOTOR
   WARNING: U IS UNDEFINED IN OUTPUT-SECTION OF MOTOR
YREF[DPID]=YR
Y[DPID]=Y[MOTOR]
U[MOTOR]=IF CONT THEN U[CPID] ELSE U[DPID]
   PRECEDED BY EXECUTION OF OUTPUT-SECTION OF CPID
   PRECEDED BY EXECUTION OF OUTPUT-SECTION OF DPID IF SAMPLING
CONT:1

UP=A*U[MOTOR]+B
A:1
B:0

END
```

Figure 4.

PLOT UP YMOTOR1 YR
GICPID1=1. TDICPID1=1. GDICPID1=0.



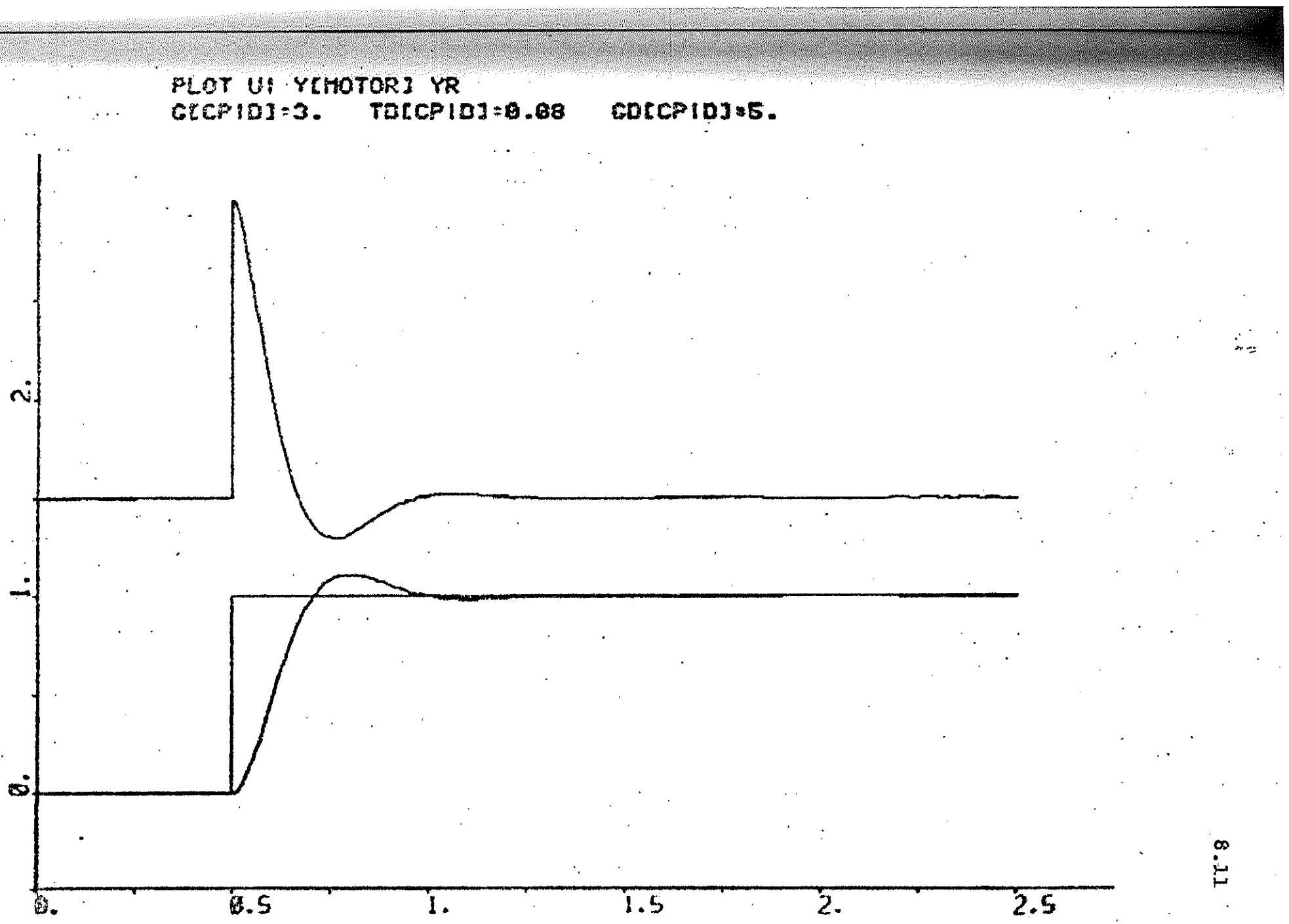


Figure 6.

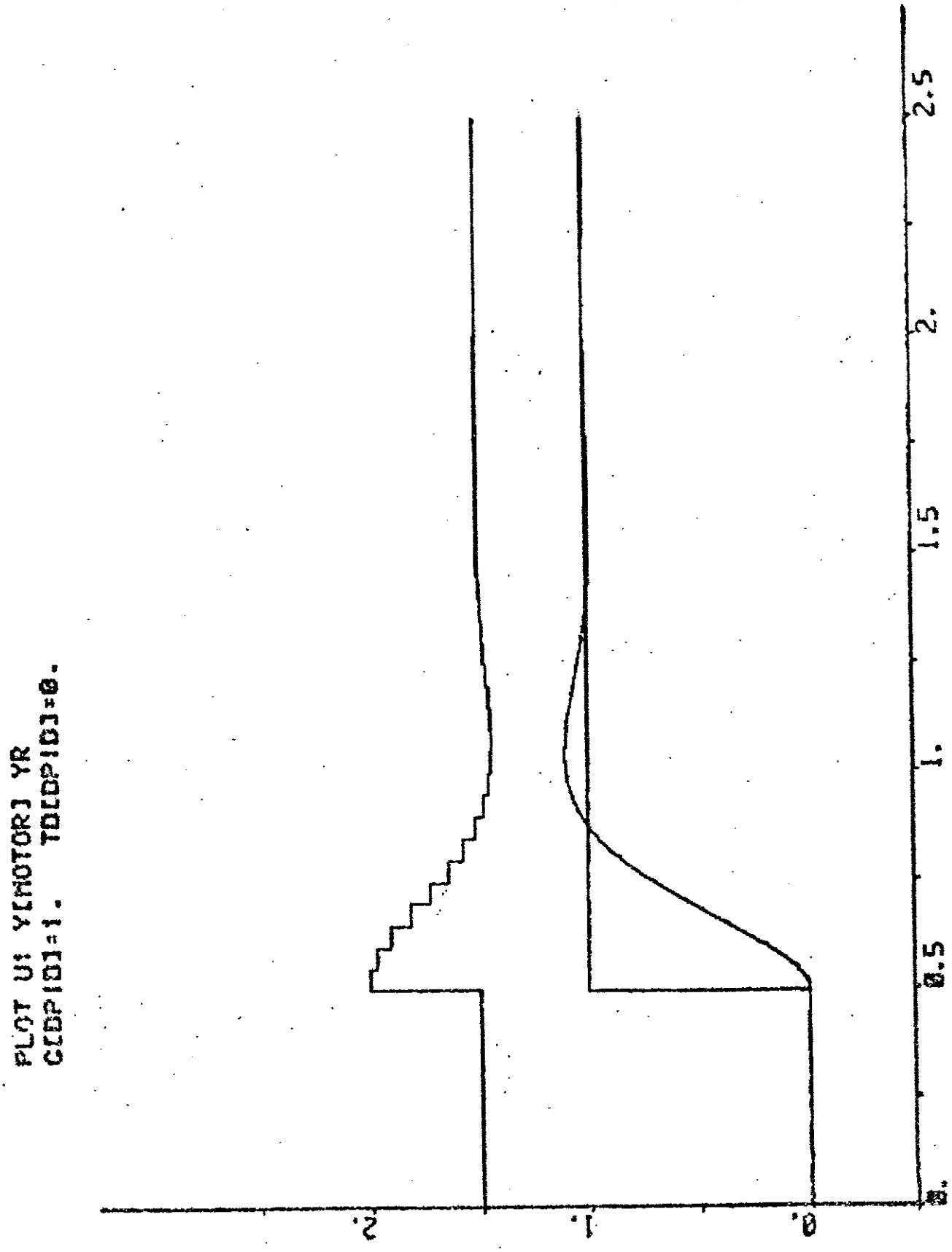


Figure 7.

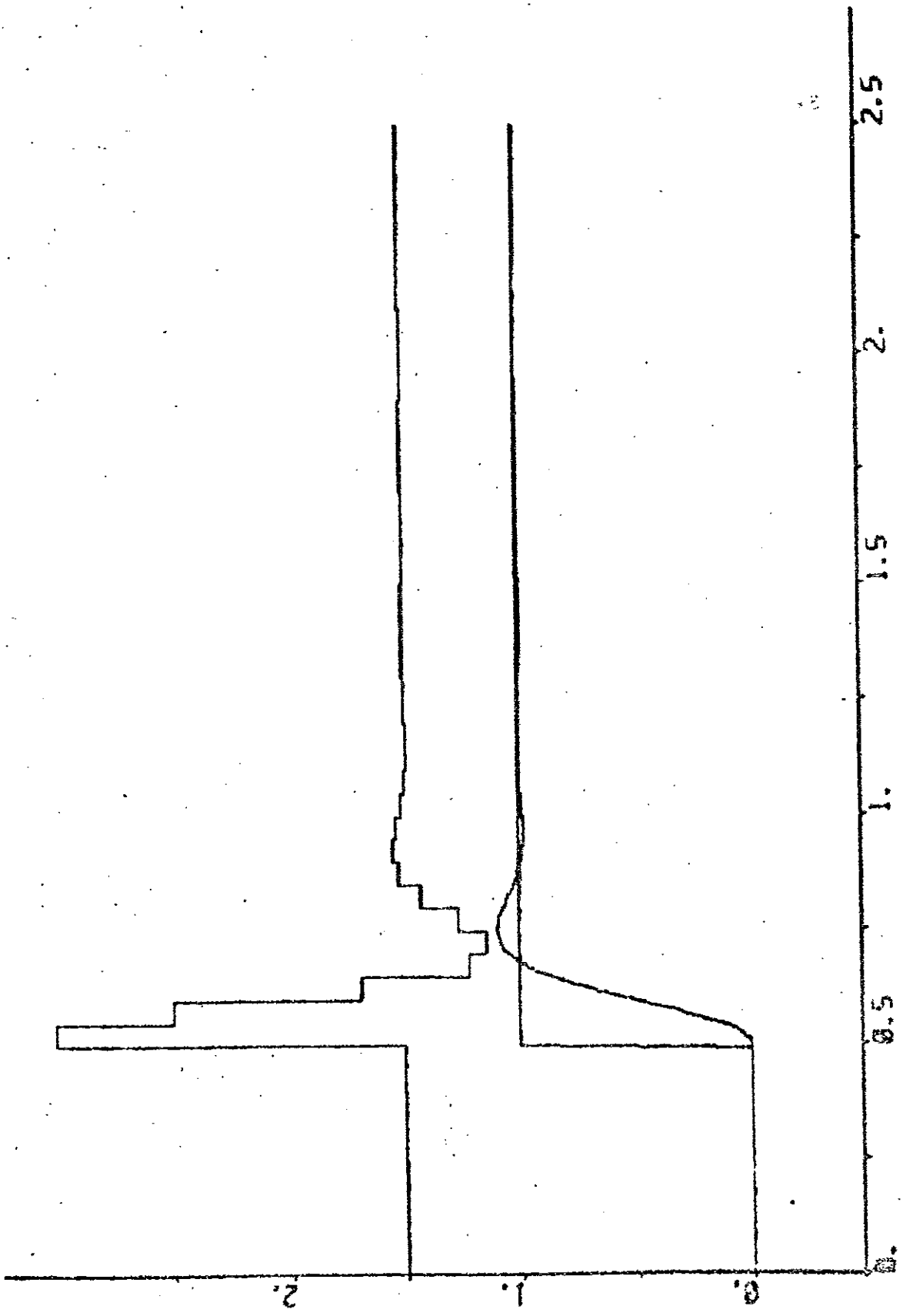


Figure 8.

Example 6. Triode oscillator (continued), The MACRO facility

This example contains a generation of a phase plane for the triode oscillator (example 1 and 2). The system thus is to be simulated with different initial values of U and V while plotting V versus U.

In general three commands are needed for the generation of each trajectory:

```
INIT U:number
INIT V:number
SIMU 0 25
```

where the two numbers specify the starting point of the trajectory.

The facility included in the program which allows for the execution of commands stored on a file (MACRO) will be used. MACRO-files have the following format.

```
MACRO <macro identifier>[<formal argument>]*
command
.
.
.
END
```

The execution of the commands in a macro is performed when the macro identifier and actual values of formal arguments are given as an ordinary command.

A formal argument is an identifier which can be used as an argument in the commands in the macro. The identifier is replaced by the actual value before the execution of each command.

For the actual problem it is natural to define a macro TRAJ containing the three commands needed for the generation of each trajectory. The starting coordinates for the trajectory are entered via two formal arguments X and Y. The macro TRAJ then becomes:

```
MACRO TRAJ X Y
INIT U:X
INIT V:Y
SIMU 0 25
END
```

To obtain a phase plane, TRAJ can now be used as a new command.

```
>TRAJ  6    8
>TRAJ -6    8
>TRAJ -6   -8
>TRAJ  6   -8
>TRAJ 0.25  0
```

The interaction needed to obtain the phase plane is shown below. To generate the macro the special command MACRO has been used. In order to study the execution of the macro the switch MACOM in INTRAC was switched on.

```
>SYST TRIOD                    " ACTIVATE TRIOD
>
>                              " DEFINE THE MACRO TRAJ
>MACRO TRAJ X Y
  >INIT U:X
  >INIT V:Y
  >SIMU 0 25
  >END
>
>                              " PLOT V VERSUS U
>PLOT V (U)
>AXES H -10 10 V -8 8
>TURN MACOM ON                 " ECHO THE MACRO COMMANDS
>TRAJ 6 8
  <MACRO TRAJ X Y
  <INIT U : 6
  <INIT V : 8
  <SIMU 0 25
  <END
>TRAJ -6 8
  <MACRO TRAJ X Y
  <INIT U : -6
  <INIT V : 8
  <SIMU 0 25
  <END
>TURN MACOM OFF                " DON'T ECHO THE MACRO COMMANDS
>TRAJ -6 -8
>TRAJ 6 -8
>TRAJ 0.25 0
>                              " SEE FIGURE 9
>
```

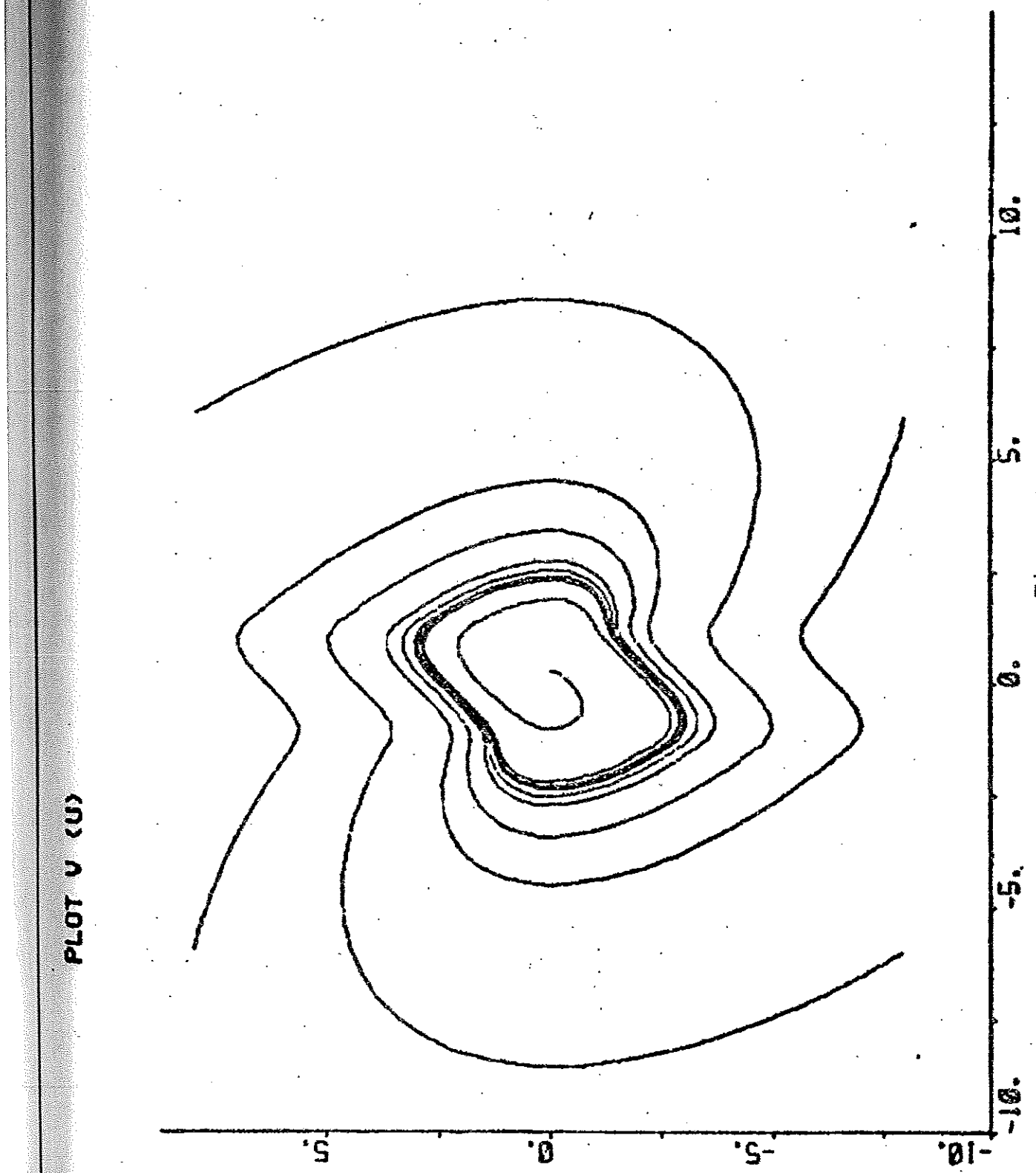


Figure 9.

Example 7. Two point boundary value problem

Consider the equations

$$u = -\lambda/\sqrt{4\pi^2 r^2 - \lambda^2}$$

$$\frac{dr}{dx} = u$$

$$\frac{d\lambda}{dx} = -2\pi\sqrt{1+u^2}$$

with the boundary conditions

$$r(x_1) = r_1$$
$$r(x_2) = r_2$$

The problem comes from finding the curve r(x) joining the points (x_1,r_1) and (x_2,r_2) which generates the surface of minimum area when rotated about the x-axis.

The problem will be solved by a shooting method i.e. the system will be simulated over intervals of length x_2-x_1 with different initial values of λ. The error $r-r_2$ at the end of each interval is used in a secant method to determine new initial values of λ.

The secant method for the iterative solution of the equation $f(y) = 0$ is defined by

$$y_{i+1} = y_i - f(y_i) \cdot \frac{y_i - y_{i-1}}{f(y_i)-f(y_{i-1})} \qquad i = 1,2,...$$

with initial values y_o and y_1.

The differential equations will be described in a continuous system (AREA) and the secant method in a discrete system (SECAN) with the sampling interval x_2-x_1. The manipulation of r and λ at the beginning of each interval will also be done in a discrete system (RESET).

The situation is pictured in figure 10.

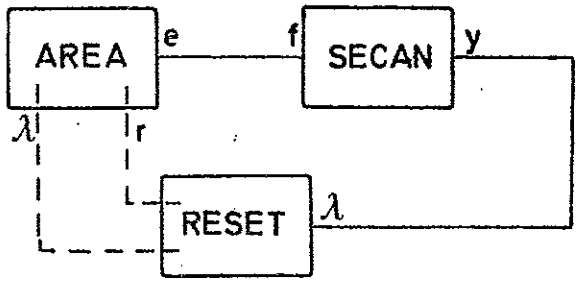


Figure 10.

The listings of the subsystems AREA and RESET and the connecting system TWOP are given in listing 7. The subsystem SECAN uses the flags LSTOP and LDARK and is thus described in a FORTRAN-subroutine (listing 8).

```
CONTINUOUS SYSTEM AREA

OUTPUT E
STATE R LAM
DER DR DLAM

U=-LAM/SQRT(4*PI*PI*R*R-LAM*LAM)
DR=U
DLAM=-2*PI*SQRT(1+U*U)

E=R-R2
R2:1

PI:3.14159

END

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

DISCRETE SYSTEM RESET

TIME T
INPUT LAM
TSAMP TS

R1:1
R[AREA]=R1
LAM[AREA]=LAM
TS=T+DT
DT:1

END

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

CONNECTING SYSTEM TWOP

TIME T

F[SECAN]=E[AREA]
LAM[RESET]=Y[SECAN]

X=T-TBEG[SECAN]

END
```

Listing 7.

```
      SUBROUTINE SECAN
C
C       DISCRETE SYSTEM TO BE INCLUDED IN SIMNON THAT PERFORMS
C       THE SECANT METHOD FOR THE ITERATIVE SOLUTION OF THE
C       EQUATION F(Y)=0.
C
C       INPUT:  F     - VALUE OF THE FUNCTION
C       OUTPUT: Y     - NEXT VALUE OF THE ARGUMENT
C               TBEG  - START TIME FOR EACH ITERATION
C       PAR:    Y0    - FIRST VALUE OF Y
C               Y1    - SECOND VALUE OF Y
C               EPS   - ACCURACY
C               DT    - SAMPLING INTERVAL
C
C       NOTE. THE OUTPUT Y DEPENDS ALGEBRAICALLY ON F.
C
C
      REAL NYY,NYOLD,NFOLD
      LOGICAL LSTOP,LDARK
      COMMON /DESTIN/ IDUM,IPART
      COMMON /TIME/ T
      COMMON /USER/ LSTOP,LDARK
C
C
      GO TO(1,2,3,4,5,6,7,8),IPART
C
C       IDENTIFICATION
 1    CALL IDENT('DISCR','SECAN')
      RETURN
C
C       DECLARATIONS
 2    CALL INPUT(F,'F')
      CALL OUTPUT(Y,'Y')
      CALL OUTPUT(TBEG,'TBEG')
      CALL PAR(Y0,'Y0')
      CALL PAR(Y1,'Y1')
      CALL PAR(EPS,'EPS')
      CALL PAR(DT,'DT')
C
      CALL STATE(YY,'YY')
      CALL STATE(YOLD,'YOLD')
      CALL STATE(FOLD,'FOLD')
      CALL NEW(NYY,'NYY')
      CALL NEW(NYOLD,'NYOLD')
      CALL NEW(NFOLD,'NFOLD')
      CALL INIT(YYI,'YYI')
      CALL INIT(YOLDI,'YOLDI')
      CALL INIT(FOLDI,'FOLDI')
      CALL TSAMP(TS,'TS')
      RETURN
C
C       CONSTANT ASSIGNMENTS
 3    Y0=0.0
      Y1=1.0
      EPS=1.0E-3
      DT=1.0
      RETURN
C
```

Listing 8.

```
C           INITIAL
 4        Y=Y0
          YY=Y0
          YOLD=Y1
          FOLD=0.0
          TS=T+DT
          TBEG=T
          RETURN
C
C           OUTPUT
 5        IF(ABS(F).LT.EPS) GO TO 50
          DIFF=F-FOLD
          IF(ABS(DIFF).LT.EPS) DIFF=-F
          Y=YY-F*(YY-YOLD)/DIFF
C
          TBEG=T
          RETURN
C
 50       LSTOP=.TRUE.
          RETURN
C
C           DYNAMICS
 6        NYY=Y
          NYOLD=YY
          NFOLD=F
C
          TS=T+DT
          LDARK=.TRUE.
          RETURN
C
 7        RETURN
 8        RETURN
C
          END
```

Listing 8 (continued).

The subsystem SECAN sets the flag LDARK and in connection with the variable X in the connecting system it is possible to plot the curves above each other.

The interaction when simulating the system is shown below.

```
>SYST AREA SECAN RESET TWOP
>PLOT R LAM[AREA]
>AXES H 0 5 V -10 5
>PAR R1:1
>PAR R2:0.6
>PAR Y0:-2
>PAR Y1:2
>SIMU 0 5
>
>MACRO DISVA
  >DISP R1 R2
  >DISP Y0 Y1
  >END
>
>DISVA
>" SEE FIGURE 11
>
>PLOT R(X)
>AXES H -0.5 2 V 0 1.5
>SIMU 0 1000
>DISVA
>" SEE FIGURE 12
>
>PAR R2:1.5
>AXES
>SIMU
>DISVA
>" SEE FIGURE 13
>
```

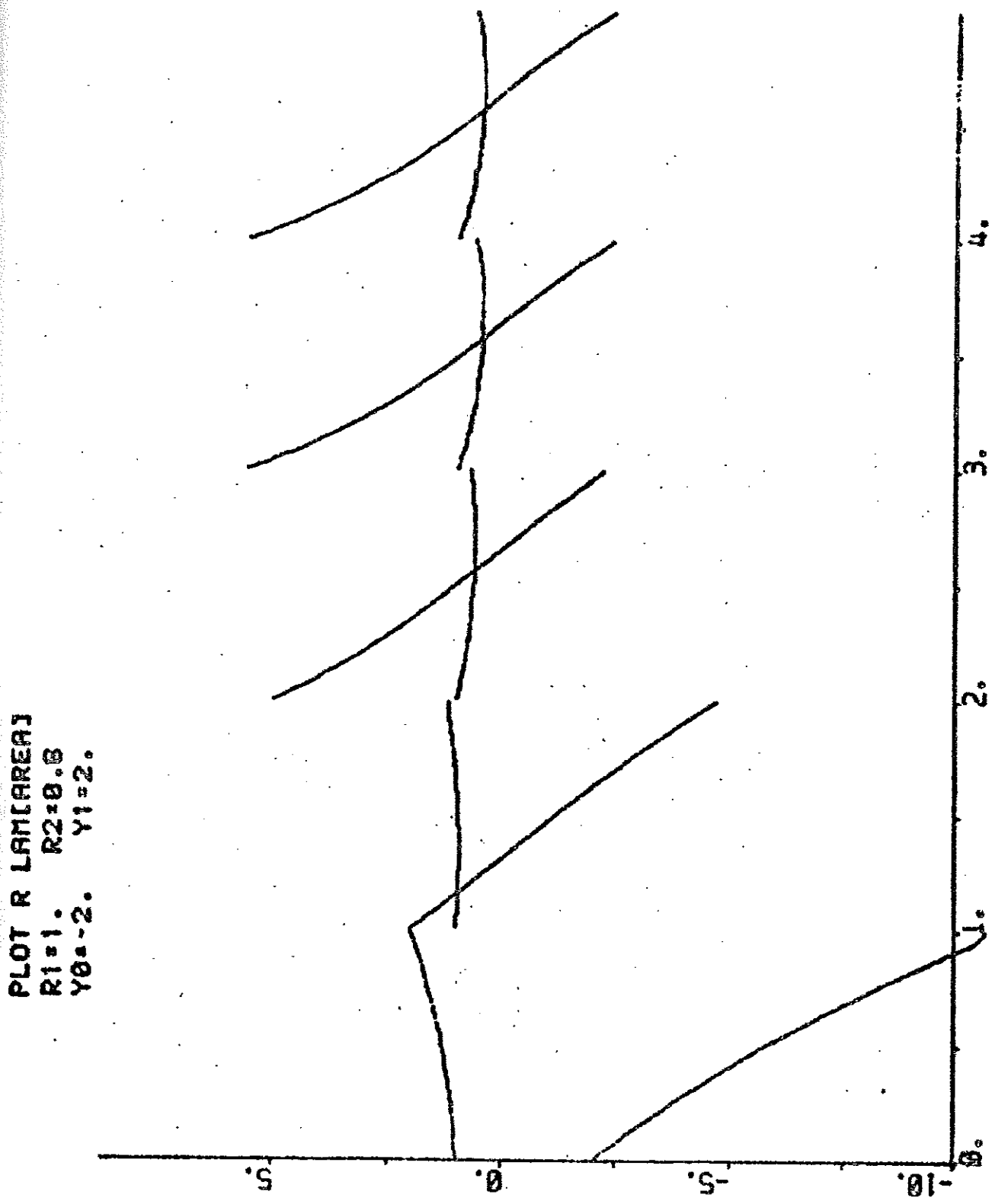


Figure 11.

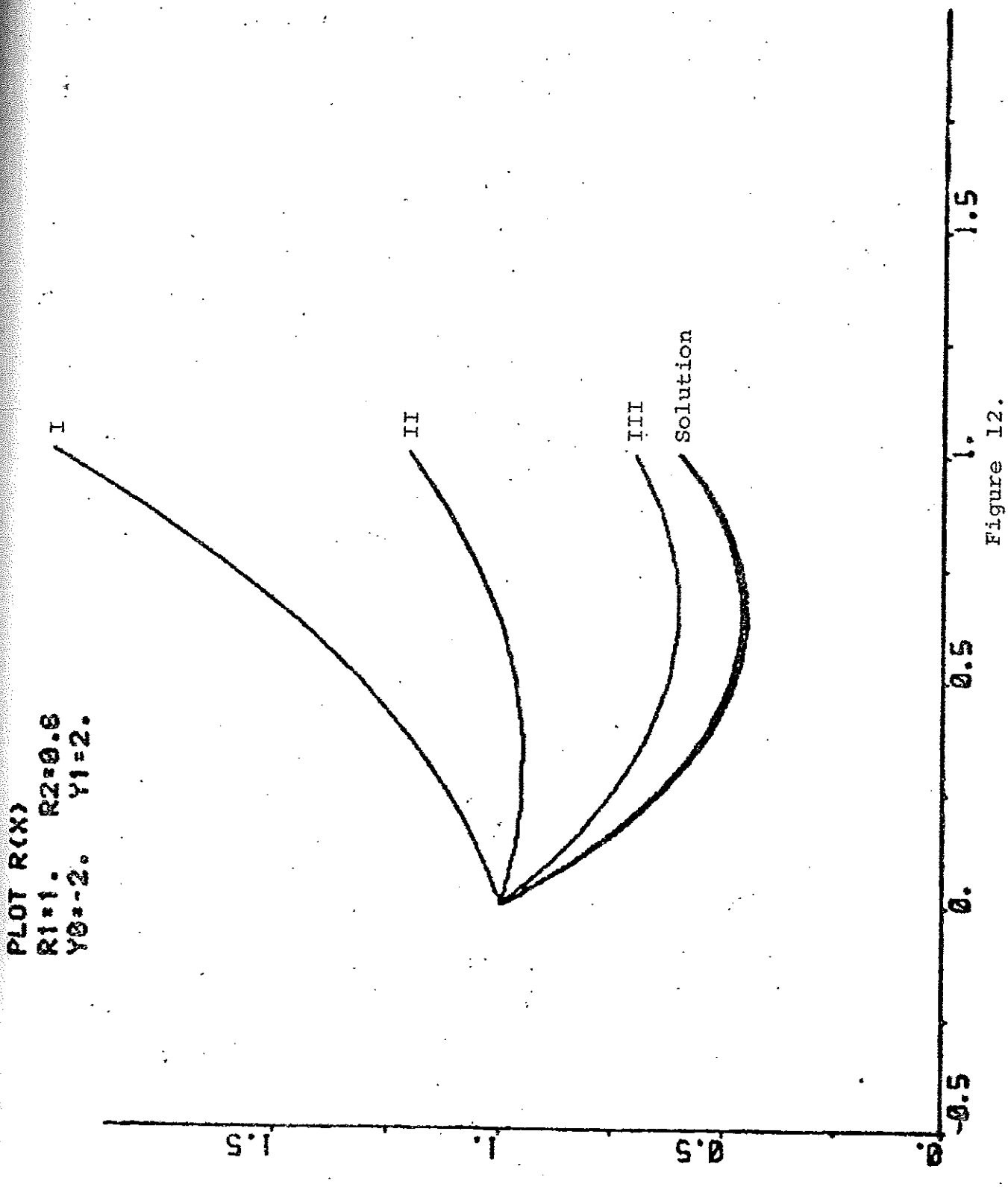


Figure 12.

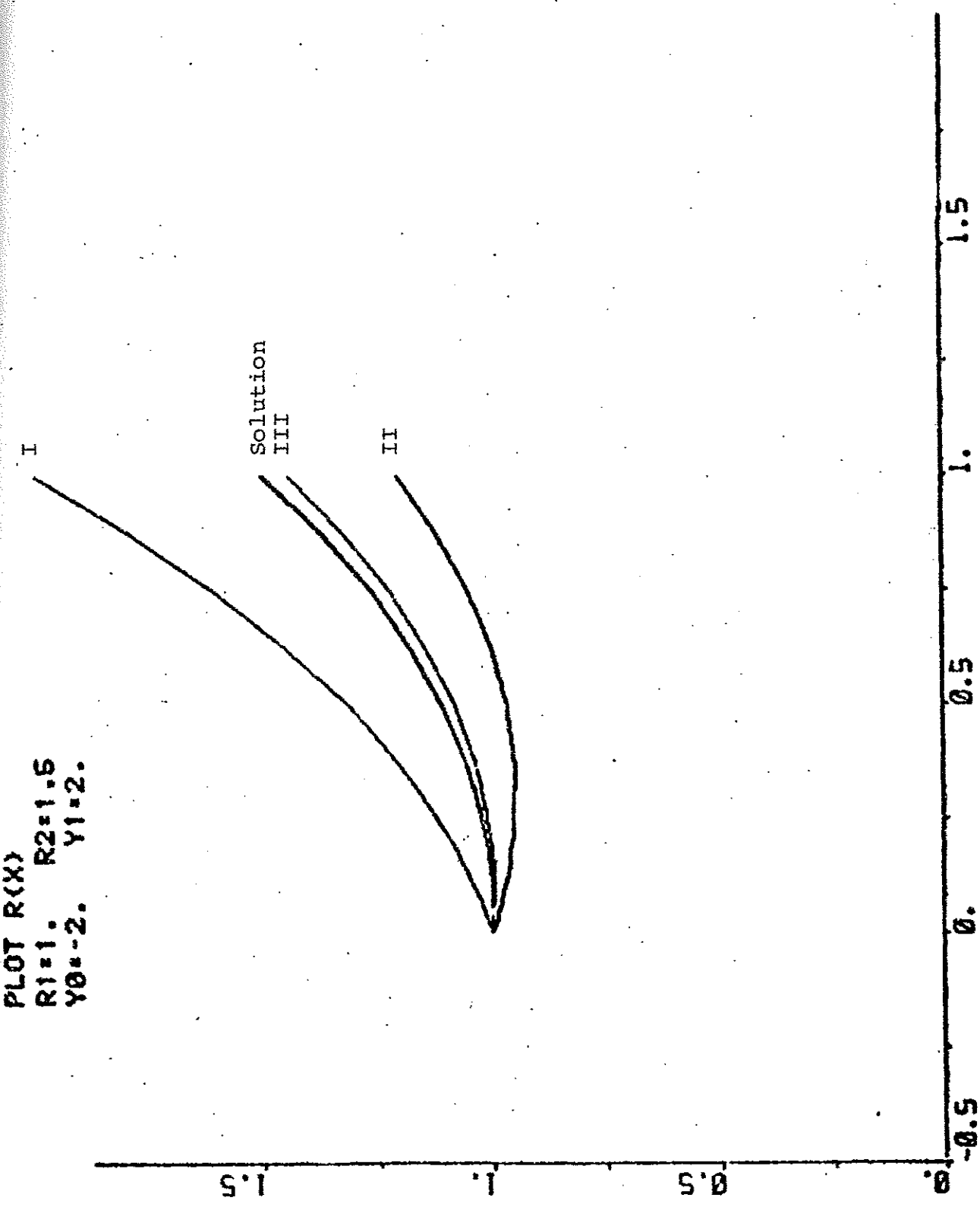


Figure 13

9. ACKNOWLEDGEMENTS

The author wants to express his great gratitude to tekn. lic. Johan Wieslander whose ideas about interactive programs greatly influenced the design of SIMNON. He has also contributed with many of the basic ideas of SIMNON.

The author also wants to thank Prof. Karl-Johan Åström and Tekn. lic. Sture Lindahl for many valuable ideas and stimulating discussions about the design of SIMNON.

It is also a pleasure to thank for all suggestions of improvements which have been received from the staff of the division.

The work has been partly supported by the Swedish Institute of Applied Mathematics and the Swedish Board for Technical Development.

10. REFERENCES

[1] T.Glad: "A program for the interactive solution of parametric optimization problems in dynamic systems", Report 7424, Lund Institute of Technology, Div. of Automatic Control, November 1974.

[2] J. Oppelstrup: "DELAY - A program for integrating systems of delay-differential equations", TRITA-NA-7311, Royal Institute of Technology, Stockholm.

[3] T. Ekman and C-E. Fröberg: "Introduction to ALGOL programming", Studentlitteratur, Lund, 1967.

[4] K.J. Åström: "Nonlinear systems", Lecture notes, Lund Institute of Technology, Div. of Automatic Control (in Swedish).

[5] H. Freeman: "Discrete time systems", John Wiley & sons Inc., New York, 1965.

[6] G. Fick: "DHAMDI - A FORTRAN subroutine to integrate a set of first order, ordinary differential equations containing discontinuities", FOA 2 report C 2504-E5, Research Institute for National Defense, November 1971.

APPENDIX A

Syntax

LETTERS

<letter>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|X|Y|Z|
a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|x|y|z
<digit>::=0|1|2|3|4|5|6|7|8|9

IDENTIFIERS

<identifier>::=<letter>|<identifier><letter>|<identifier><digit>

NUMBERS

<unsigned integer>::=<digit>|<unsigned integer><digit>
<integer>::=<unsigned integer>|+<unsigned integer>|
-<unsigned integer>
<decimal fraction>::=.<unsigned integer>
<exponent part>::=E<integer>
<decimal number>::=<unsigned integer>|<unsigned integer>.|
<decimal fraction>|<unsigned integer><decimal fraction>
<unsigned number>::=<decimal number>|<decimal number><exponent part>
<number>::=<unsigned number>|+<unsigned number>|-<unsigned number>

VARIABLES

<system identifier>::=<identifier>
<simple variable>::=<identifier>
<variable>::=<simple variable>|<simple variable>[<system identifier>]

System headings

<system heading>::=CONTINUOUS SYSTEM<system identifier>|
DISCRETE SYSTEM<system identifier>|
CONNECTING SYSTEM<system identifier>

Declarations

<declaration>::=<type><type list>
<type>::=INPUT|OUTPUT|TIME|STATE|DER|NEW|TSAMP
<type list>::=<simple variable>|<simple variable><type list>

Section headings

<section heading>::=INITIAL|OUTPUT|DYNAMICS|CONNECT

Assignment statements

FUNCTION DESIGNATORS

<procedure identifier>::=<identifier>
<actual parameter>::=<arithmetic expression>|<Boolean expression>
<actual parameter list>::=<actual parameter>|
    <actual parameter list>,<actual parameter>|
<actual parameter part>::=(<actual parameter list>)
<function designator>::=<procedure identifier><actual parameter part>

ARITHMETIC EXPRESSIONS

<adding operator>::=+|-
<multiplying operator>::=*|/
<primary>::=<unsigned number>|<variable>|<function designator>|
    (<arithmetic expression>)
<factor>::=<primary>|<factor>↑<primary>
<term>::=<factor>|<term><multiplying operator><factor>
<simple arithmetic expression>::=<term>|<adding operator><term>|
    <simple arithmetic expression><adding operator><term>
<if clause>::=IF<Boolean expression>THEN
<arithmetic expression>::=<simple arithmetic expression>|
    <if clause><simple arithmetic expression>ELSE<arithmetic
                                                expression>

BOOLEAN EXPRESSIONS

<relational operator>::=<|>
<relation>::=<simple arithmetic expression><relational operator>
<simple arithmetic expression>
<Boolean primary>::=<variable>|<function designator>|
<relation>|(<Boolean expression>)
<Boolean secondary>::=<Boolean primary>|NOT<Boolean primary>
<Boolean factor>::=<Boolean secondary>|
<Boolean factor>AND<Boolean secondary>
<Boolean term>::=<Boolean factor>|<Boolean term>OR<Boolean factor>
<Boolean expression>::=<Boolean term>|
<if clause><Boolean term>ELSE<Boolean expression>

ASSIGNMENT STATEMENTS

<left part>::=<variable>=
<assignment statement>::=<left part><arithmetic expression>|
<left part><Boolean expression>

Constant assignments

<constant assignment>::=<simple variable>:<number>

End statement

<end statement>::=END

APPENDIX B

Summary of commands

SYST{<identifier>}*[-EDIT]

Defines the system.

PLOT[{<variable>}*[(<variable>)]]

Selects variables for plotting.

AXES[{H|V}<minimum><maximum>[{H|V}<minimum><maximum>]]

Draws axes.

SIMU[<start time><stop time>[<time increment>]]
[-{CONT|MARK}[{CONT|MARK}]][/<file name>]

Simulates the active system.

PAR<variable>:{<number>|<variable>}

Changes the value of a parameter.

INIT<variable>:{<number>|<variable>}

Changes the initial value of a state variable.

STORE[{<variable>}*[-ADD]]

Selects variables to be stored.

SHOW{{<variable>}*[(<variable>)][-MARK]|-LIST}[/<file name>]

Plots stored variables.

DISP[({DIS|TP|LP})][<variable>]*

Displays the current value of variables.

LIST[({DIS|TP|LP})]{<file name>}*

Lists files.

EDIT<file name>
Initializes the editor.

SAVE<file name>[<system identifier>][-{PAR|INIT}]
Saves parameter values and initial values of states on a file.

GET<filename>
Gets parameter values and initial values of states from a file.

ALGOR{HAMPC|RK|RKFIX}
Selects integration routine.

ERROR<error bound>
Sets the error bound for the integration routine.

TURN{WARN|COMPU|PLCOM|DARK}{ON|OFF}
Turns switches on or off.

STOP
Stops the execution of SIMNON.

APPENDIX C

Summary of editor commands

N[EXT][<integer>]
Goes a number of lines down the file.

P[RINT][<integer>]
Goes a number of lines down the file printing the lines on the terminal.

L[OC] <string>
Locates a line containing the string.

F[IND] <string>
Locates a line beginning with the string.

D[EL][<integer>]
Deletes lines.

O[VERL][<integer>]
Deletes lines and changes the editor mode to INPUT.

R[ETYP] <string>
Replaces the current line with the string.

I[NS] <string>
Inserts the string as a line following the current line.

A[PPND] <string>
Appends the string on the current line.

C[HANG] <quote><string><quote><string>[<quote>]
Changes the characters in the current line specified by the first string to the characters in the second string.

B[OTT]
Goes to the end of the file.

T[OP]

Goes to the beginning of the file.

E[XIT]

Closes the file and exists.

DIS[ON|OFF]

Enables or disables the printout on the display.

LEAVE

Exits in a special way.