Technical Report No. 218

SIMCOMP VERSION 3.0 USER'S MANUAL

Jon D. Gustafson and George S. Innis

Natural Resource Ecology Laboratory
Colorado State University
Fort Collins, Colorado

GRASSLAND BIOME
U.S. International Biological Program

June 1973

TABLE OF CONTENTS

List of Examples

ABSTRACT

SIMCOMP Version 3.0 is a FORTRAN-like computer simulation language designed to ease the development and implementation of compartmental flow/discrete event simulations. The language is an extension and refinement of SIMCOMP Version 2.0. The system is designed to reduce the programming overhead required while sufficient flexibility to solve certain problems. Compartmental-flow simulations are defined by specifying the flow rates between compartments and may be in either difference or differential equation form. Discrete events can be included in the form of event routines which are controlled by a dynamic event scheduler. Various forms of tabular and graphic output can be easily requested. An execution time interpretive debugging facility is also included. The syntactical rules for writing SIMCOMP programs are presented in this document along with a number of examples.

Introduction

The development of a second major simulation compiler, SIMCOMP Version 3.0, perhaps requires some justification. SIMCOMP Version 3.0 represents a series of refinements which have extended the capabilities of SIMCOMP Version 2.0 (Gustafson and Innis 1972). The refinements which have been incorporated into the Version 3.0 compiler are best understood in three broad categories. These categories are (i) refinements and extensions due to conceptual advances in modeling paradigms; (ii) refinements which ease the coding of SIMCOMP source programs; and (iii) the development of more efficient algorithms within the compiler and in the generated object program.

As a means of abstraction and representation of ecological systems, computer simulation models have taken many forms. Since many successful ecological simulations have adopted the state equation approach, we do not wish to abandon this paradigm. Instead, SIMCOMP Version 2.0 was designed around an extension of the state equation paradigm, namely the conceptualization of an ecological system as a set of compartmental-flow equations. SIMCOMP Version 3.0 retains the capability to represent systems as sets of compartmental-flow equations. The program organizes these equations into difference equations and provides a solution algorithm.

Certain categories of ecological phenomena appear strained when described in terms of state equations. These categories include (i) phenomena which admit to a heuristic description; (ii) phenomena with a low probability of occurrence; and (iii) phenomena which are best described as stochastic processes. The above categories are not meant to be exhaustive or mutually exclusive. The feature incorporated into SIMCOMP Version 3.0 to facilitate the modeling of such phenomena includes event routines and an event scheduler. In addition, a number of distribution functions for the generation of stochastic variates are available. Event and compartmental-flow simulations may be combined in the same simulation.

Changes in the format of SIMCOMP Version 3.0 statements are in large measure due to the comments and suggestions of the users of Version 2.0. Source program statements were simplified when possible. A number of new statements have increased the capabilities of the language. For example, the fact has been recognized that many times in large ecosystem simulations flow equations are developed which take similar mathematical form, varying only in the values of the parameters. SIMCOMP Version 3.0 provides for such cases with the capability to iteratively declare flows. Additionally, an execution time interpretive variable dump facility has been provided to ease the debugging process.

The execution characteristics of the compiler and the generated object code have been significantly improved in Version 3.0. Execution times and core requirements have been reduced. A complete description of the system and its operation is contained in the "SIMCOMP Version 3.0 Maintainance Document" (Stevens and Gustafson 1973).

1. SIMCOMP SIMULATIONS.

Every simulation language is designed to ease the translation of real-world phenomena into computer simulation models. Each language is designed to model a particular class of real systems. Some notable examples are the application of MIMIC (Control Data Corporation 1972) to the simulation of continuous physical systems and the application of SIMSCRIPT (Markowitz et al. 1963) to the simulation of queuing and inventory systems. SIMCOMP was designed primarily to model ecological systems. As is the case with other simulation languages, the realm of applicability of SIMCOMP is certainly not limited to the field for which it was designed (ecology).

The principle features of SIMCOMP were designed with the modeling of ecological systems in mind. This desire required some broad generalizations about the nature of ecological systems. These generalizations include the following:

(1) Ecological systems can be viewed, in part, as systems of continuously varying variables.

(2) Ecological systems can be visualized as a system of compartments linked by material or energy transfers or flows.

(3) The flow of material or energy depends on other states and driving variables of the system (information flows).

(4) Ecological systems contain components which can be visualized as discrete-valued variables.

(5) Ecological systems contain processes which can be visualized as events occurring discretely through time.

SIMCOMP is designed to model phenomena in the above categories. Sections 1.1 and 1.2 are presented to formally define the paradigms employed by SIMCOMP to implement these generalizations. Sections 2 and 3 describe the syntax and coding procedures for writing SIMCOMP simulations.

1.1 Flow-Oriented Continuous Simulations.

A broad variety of techniques have been developed to model and simulate many systems. Differential equations and difference equations have been most used in the development of ecological models. As greater reality and resultant complexity are introduced, the solutions have become more intractable and computers have been employed. Computer simulation in this context has come to mean the numerical solution of a simultaneous set of differential or difference equations.

Although many solution schemes are available, one of the most versatile approaches is to view the simulation as an initial value problem. The initial value problem for first-order difference equations takes the following form. Let the amount of material or energy in the i^{th} compartment at time t be represented by $x_i(t)$. For a system of n compartments, the state of the system at any time t can be expressed as a vector

$$\underset{\sim}{x}(t) = \Big(x_1(t), x_2(t), \ldots , x_n(t)\Big)$$

Let a change in the state of the system over some time interval, say Δt from time t to time $t + \Delta t$, be represented by

$$\Delta\underset{\sim}{x}(t) = \Big(\Delta x_1(t), \Delta x_2(t), \ldots , \Delta x_n(t)\Big) .$$

In general, the $\Delta x_i(t)$ are functions which may depend upon,

(1) the values of the state variables at time t, $\underset{\sim}{x}(t)$.

(2) the values of a set of informational variables, say $v_j(t)$ for $j = 1, \ldots , m$, which, in general, vary with time (these may depend upon or include driving variables).

(3) the values of a set of parameters or constants, say p_k, $k = 1, \ldots , s$, which do not vary with time.

(4) and time itself.

The change in the i^{th} state variable $\Delta x_i(t)$ at time t over the time interval Δt may be functionally written as

$$\Delta x_i(t) = F_i[\underset{\sim}{x}(t), \underset{\sim}{v}(t), \underset{\sim}{p}, t, \Delta t] \cdot \Delta t,$$

where the function F_i is the change per unit time in the state variable x_i. Note that if the system modeled is to be represented by *differential* as opposed to *difference* equations, then the dependence of F_i upon Δt should not exist.

Given the initial values of the state variables at time $t = t_0$, that is $\underset{\sim}{x}(t_0)$, and the changes in the state variables $\Delta\underset{\sim}{x}(t)$, we can find the state of the system at any time $t_m = t_0 + m\Delta t$ for $m = 0, 1, 2, \ldots , M$. The state of the system at any time t_M is iteratively computed as

$$\underset{\sim}{x}(t_M) = \underset{\sim}{x}(t_0) + \Delta t \cdot \sum_{m=0}^{M-1} F_i[\underset{\sim}{x}(t_m), \underset{\sim}{v}(t_m), \underset{\sim}{p}, t_m, \Delta t]$$

In order to simulate biological systems, we postulate the following three principles.

(1) A biological system can be viewed as a collection of smaller subsystems. (Indeed some systems might consist of a single subsystem.)

(2) A change of state in any subsystem must result from the flow of material or energy between compartments contained in that subsystem.

(3) The identity of the material or energy flowing in any subsystem must maintain its physical identity throughout the subsystem.

As a result of the second postulate, we have further required that the change of state of any particular compartment be expressed as the algebraic sum of the flows to or from that compartment. Let the *net* flow per unit time from compartment i to compartment j be represented by

$$f_{ij} = f_{ij}[\underset{\sim}{x}(t), \underset{\sim}{v}(t), \underset{\sim}{p}, t].$$

Note that $f_{ij} = -f_{ji}$, that is the net flow into compartment j, is reflected by a corresponding loss from compartment i, and by necessity the identity of the material flowing must remain unique. Therefore, expanding upon our formulation of the solution of the initial value problem, we find that the rate of change of material in some compartment i, above expressed as $F_i\ [\underset{\sim}{x}(t), \underset{\sim}{v}(t), \underset{\sim}{p}, t]$, is the sum of the net flows from each of the associated compartments. Formally this requires that $F_i = \sum_{j\varepsilon S} f_{ij}$, where S is the set of compartments which are coupled to compartment i by flows.

A nine-compartment system comprised of two subsystems is illustrated in Fig. 1.1-1. The compartments are represented by the boxes. Material or energy flows are represented by solid arrows between the compartments. A flow which is always in one direction is represented by a single-headed arrow, such as the flow from compartment 1 to compartment 3. Flows in which the net flow may be in either direction are represented by a double-headed arrow, such as the flow between compartment 2 and compartment 3. Note that there are no material flows between compartments in separate subsystems. Informational flows are represented by dotted arrows. The rate of flow between compartment 5 and compartment 7 for example, is controlled by the amount of material in compartment 3. These informational flows are represented by $\underset{\sim}{v}(t)$ in our mathematical formulation. We may further identify compartment 1 as a source, provided the flow from 1 to 3 does not depend upon the quantity of material in 1, and likewise identify compartment 8 as a sink.

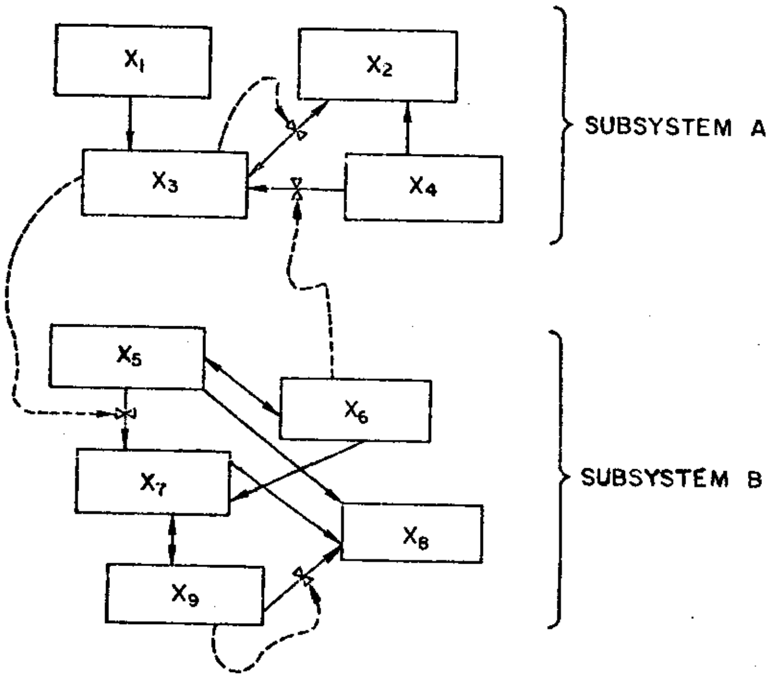


Fig. 1.1-1. A nine-compartment system comprised of two subsystems illustrating actual flows (solid arrows) and informational flows (dotted arrows).

We have now identified the elements necessary to specify a computer simulation of a compartmental-flow model. We require,

(1) initial values for the state variables.

(2) mathematical expressions which calculate the net flows between compartments.

(3) mathematical formulas which calculate the informational flows.

(4) the identification and values for parameters and constants used in (2) and (3).

(5) the starting and final times over which the simulation is to be run and the time step for the numerical solution of the initial value problem.

SIMCOMP is designed so that these five items are easily specified by the user. SIMCOMP organizes the flow expressions into difference equations and provides the solution. SIMCOMP further requires the user to specify what information is to be printed and plotted. The syntactical definitions for writing SIMCOMP programs are explained in section 2. The mathematical formulation of any compartmental-flow simulation, by first specifying the above five items, perhaps with the aid of a flow diagram, should precede the formulation of a SIMCOMP program.

1.2 Event Simulations.

While many ecological processes can be visualized as material or energy transfers between compartments, some processes are not so easily represented. Processes in the following categories can often be most easily described by an event-oriented simulation:

- Processes involving discrete-valued variables.
- Processes which can be visualized as a queuing problem.
- Processes which do not occur uniformly through time.
- Stochastic processes.

An event in SIMCOMP can be formally defined by specifying the following two items:

(1) A computation or set of computations, referred to as the *action* of the event, which represents the effect of the event on the system.

(2) A specification of the *time of occurrence* of the event.

All events will be assumed to have the above two attributes. The development of an event-oriented simulation model will then involve abstracting from real-world phenomena, processes which can be completely described by specifying the action of the process on the system and the timing of the process. For example, if a birth process is to be simulated, the action of the process would be to increase the number of individuals in the population. The time of occurrence of the birth would also have to be determined.

There exists two methods by which the time of occurrence of an event can be determined: events generated within the model and events fed to the model from the outside world. The first method of event scheduling is termed *internal* or *endogenous* while the second method is termed *external* or *exogenous*. The difference between the types is that endogenous events are triggered by the explicit reaction of the model to its operations, i.e., the model generates internal events as it progresses, while exogenous events are fed to the model from an external data source.

Changes which take place in the state of the system when an event occurs are termed *actions*. Central to the concept of an event is that an action requires zero-simulated time to occur. This is the crucial difference between discrete-event and continuous-time simulations. In discrete-event simulations, state-changes take place only at specified points in simulated time at which interactions between system components occur. In continuous-time simulations, interactions and state-changes take place continuously. To model continuous changes, numerical integration procedures must be employed, but are not required for discrete-event simulations.

2. SIMCOMP PROGRAMMING.

SIMCOMP is a FORTRAN-like language designed to implement both continuous and event simulations as described in section 1. Continuous and discrete-event simulations may be combined in the same simulation.

The fundamental elements of a continuous-variable simulation are *flow definitions*. Likewise *event definitions* are the fundamental elements of discrete simulations. Storage allocation for globally defined variables (i.e., variables accessible by all portions of the simulation) is provided by means of *storage declarations*. FORTRAN subroutines and functions may be supplied by the user. The *source section* of a SIMCOMP simulation is specified by the inclusion of any or all of the above statement types. The format and usage of SIMCOMP source statements are described in section 2.1.

A SIMCOMP simulation source section is a mixture of FORTRAN statements and SIMCOMP processor directives. This manual assumes a basic knowledge of FORTRAN programming and the user is referred to any good instructional FORTRAN manual such as "Computer Programming - FORTRAN IV" (Anderson 1966). The SIMCOMP processor produces code which is compiled by Control Data Corporation's FORTRAN Extended Version 3.0 compiler. It is recommended that all FORTRAN coding contained in a SIMCOMP source program conform to the specifications in the "FORTRAN Extended Reference Manual" (Control Data Corporation 1973).

The initial values of state variables and parameters are specified in the SIMCOMP *data section.* Requests for tabular and graphic output are also included in the data section. The format and usage of SIMCOMP data section specifications are described in section 2.2. The data section is read in and processed by the SIMCOMP-generated simulation program. The sequence of operations in the processing of the source and data sections is outlined in Fig. 2-1.

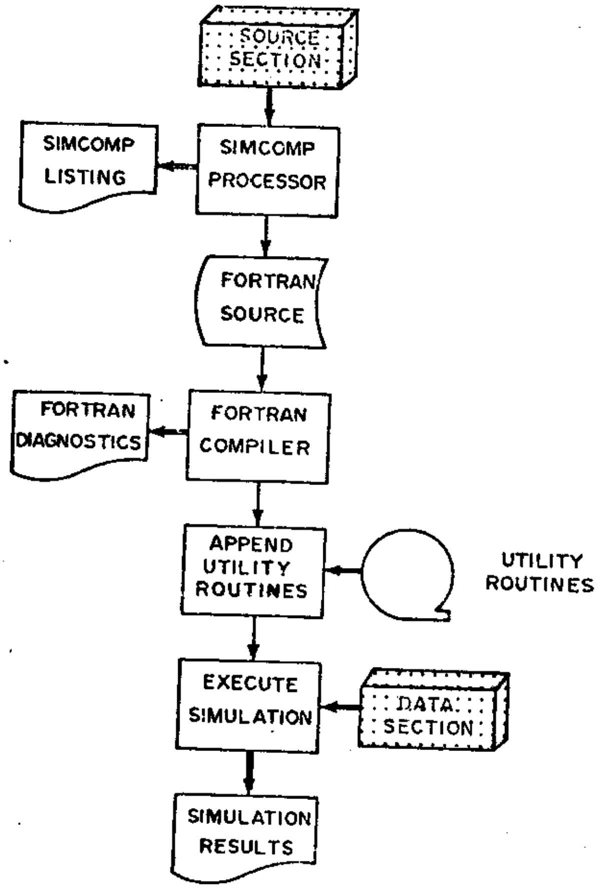


Fig. 2-1. Sequence of operations in a typical SIMCOMP job. Dotted blocks indicate user-supplied portions.

The detection and notification to the user of execution errors is a task usually assigned to the computer operating system. SIMCOMP provides an execution time debugging facility which recovers control from the operating system when an error is detected. A mnemonic dump of user variables along with an explanation of the nature of the error is printed in the output. Section 2.3 describes how this information may be utilized in debugging programs.

2.1 Source Program.

SIMCOMP source programs are a mixture of *SIMCOMP processor directives* and *FORTRAN statements*. The SIMCOMP compiler is actually a pre-processor which converts SIMCOMP source language statements into segments of FORTRAN compilable code.

SIMCOMP recognizable processor directives are generally comprised of a key word followed by a period. These statements may be contained anywhere in columns 1 through 72. FORTRAN statements included in the text must begin in or after column 7. Columns 1 through 5 are used for statement labels and columns 73 through 80 are ignored. Column 6 is used for statement continuation. *FORTRAN statement labels may take on any value with the exception of five digit labels beginning with 9.*

SIMCOMP reserves certain variable names as attributes of the system. Any of these variables may be used (or altered at the user's discretion) in the computation at any time. These variables and their meaning are shown in Table 2.1-1.

Table 2.1-1. Reserved variable names (simulation control variables).

Variable	Meaning
X(i) where $1 \le i \le 999$	Current amount of material in compartment i.
TIME	Current simulated time.
TSTRT	Starting time of the simulation.
TEND	Ending time of the simulation.
DT	Integration step size.
DTPR	Time step between print-outs.
DTPL	Time step between plotted values.
DTFL	Time step between flow print-outs.
FLOW	Value of the currently computed flow.

The user is cautioned against the use of any variable name beginning with the letter X. Variables which are internal to the operation of the SIMCOMP system use the convention of beginning with the letter X. This avoids potential conflicts between the user-supplied code and the system routines. This precaution deserves special cognizance for "canned" FORTRAN subroutines where such variables might be used.

2.1.1 Parameter declarations.

Parameter declarations are used for two purposes. These are (i) storage allocation and (ii) stochastic function definition. All parameter declarations consist of a key word followed by a period, followed by a list of names delimited by commas of the following form:

key word. name_1, name_2, ... , name_n

The key word may begin in any column. The entire statement should be contained in or before column 72. Parameter declaration statements may not be continued on successive cards. As many parameter declaration statements may be included as are required. Parameter declaration statements can appear anywhere in the source program with the following exceptions:

(1) within the text of a flow.

(2) within an event routine or subprogram.

Variable storage allocation.

Storage allocation statements consist of statements of the following form:

STORAGE. var_1, var_2, ... , var_n
INTEGER. var_1, var_2, ... , var_n
REAL. var_1, var_2, ... , var_n

The names of variables in the variable declaration list may be from 1 to 5 alphanumeric characters in length and must begin with a letter other than X.

Variables which fall in the following categories should be declared in a variable storage allocation statement.

(1) Variables which are subscripted.

(2) Variables whose values are to be assigned via data assignment statements in the data section (refer to section 2.2.1).

(3) Variables where values are to be printed or plotted via PRINT.[1] or PLOT. requests. (refer to section 2.2.2).

(4) Variables whose values are computed in flows and are used in events or subprograms and vice versa.

(5) Variables whose implicit type must be altered, i.e., from integer to real or from real to integer.

STORAGE., INTEGER., and REAL. statements can be thought of as FORTRAN COMMON, INTEGER, and REAL declaration statements. This is true with the exception that any variable declared in an INTEGER. or REAL. statement is treated as though the variable were also declared in a STORAGE. statement. As such any variable declared in one of the three storage allocation statements can be considered to be globally defined in all segments of the simulation. *All events and subprograms, in addition*

---

[1] Note that the period is part of the command verb.

to all flows, have access via its mnemonic name to the value of any declared variable. The maximum number of dimensions allowed for any subscripted variable is three.

Example 2.1.1-1. Storage allocation statements.

```
STORAGE. A,B,P(3),Q(2,3),INDEX
REAL. M(3),N,L(2,2)
INTEGER. D,E(2,3,2),F,B,P(6),L
```

If a variable is declared more than once, the last declared mode of the variable is assumed. In the above example variable "B" would be assumed type integer. If a variable is dimensioned more than once, the last declared dimensions hold. In the above example the variable "P" is assumed to be an integer one-dimensional array with six locations. Once a variable is dimensioned, the dimensionality of the variable remains in force regardless of changes in type. The variable "L" above is assumed to be an integer two-dimensional array with four locations (two by two).

Primary and secondary class storage.

Normally, the initial values of all user-declared variables are printed in the output after the data section has been processed, prior to the start of the simulation. User-declared variables can be segregated into two classes of variables by the following convention.

Any user-declared variables named in storage allocation statements in the normal manner will be considered a primary-class variable. All primary-class variables will be printed in the initial-conditions output unless otherwise requested. Secondary-class variables are prefaced by an asterisk. Normally, secondary-class variables will not be printed in the initial-conditions output. Secondary-class variables are treated just as primary-class variables in all other respects. This feature is useful especially in the case of large arrays whose initial conditions are not of interest, thus minimizing the amount of output produced in the initial-conditions output. Data section commands which will alter the normal procedure taken for selecting variables for printing in the initial-conditions output are described on page 2.2.2-17. An example of secondary-class variables is also presented in section 2.2.2.

Stochastic function definitions.

Stochastic function definition statements consist of statements of the following form:

UNIFORM. $\text{name}_1, \text{name}_2, \ldots, \text{name}_n$
NORMAL. $\text{name}_1, \text{name}_2, \ldots, \text{name}_n$
EXPONENT. $\text{name}_1, \text{name}_2, \ldots, \text{name}_n$
LOGNORMAL. $\text{name}_1, \text{name}_1, \ldots, \text{name}_n$

The names contained in the variable list must contain from 1 to 5 alphanumeric characters beginning with a letter. Variable names starting with the letter X should not be used. Similarly variable names which are implicitly type integer should not be used. Continuation cards are not allowed. As many stochastic function definition statements as required may be included.

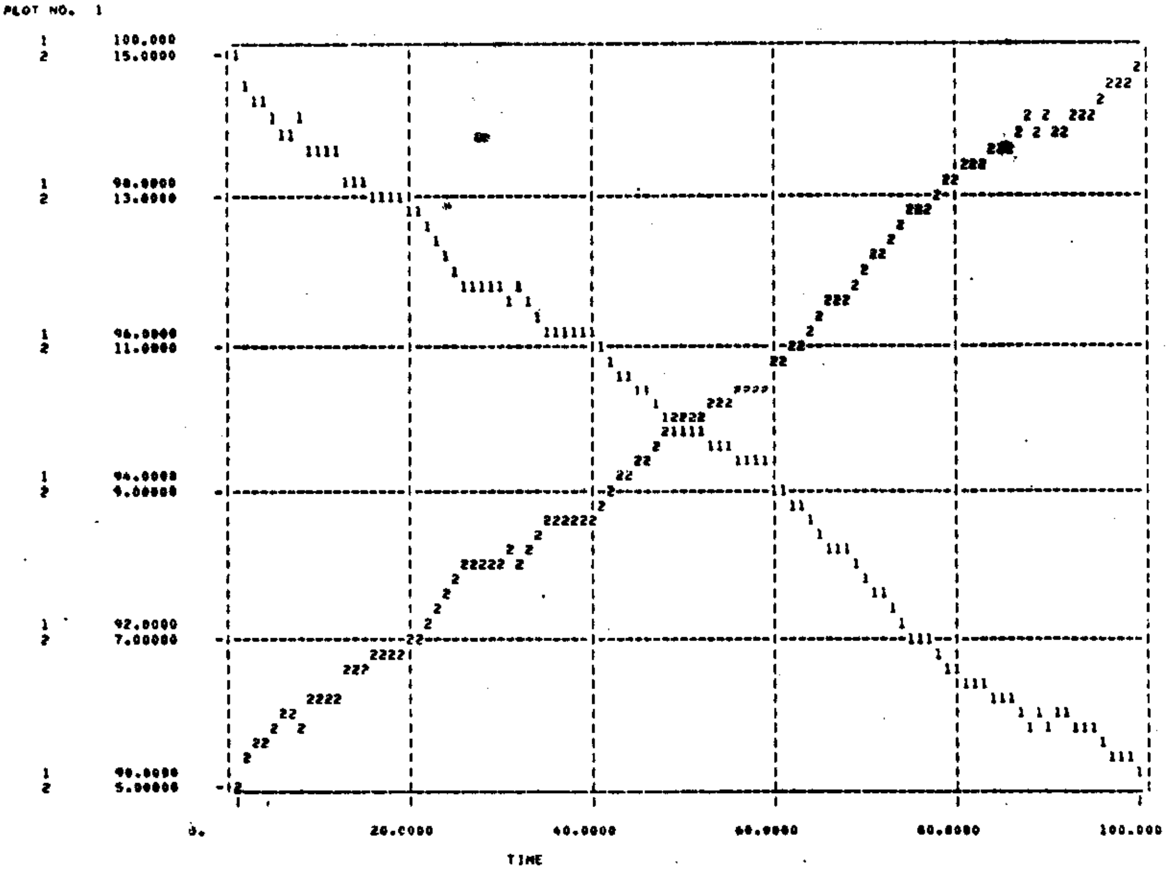
Each entry in the list does not actually allocate storage to the named variable, but generates a function subprogram of that name. The function is called by using the variable name in an arithmetic expression. In the expression the variable name must be followed by an argument list containing the correct number of parameters which specify the particular distribution function. The parameters in the argument lists must be real-valued constants or variables. Each call returns a value from the indicated distribution as the value of the function. The number of parameters and their meanings for each of the distributions are given in Table 2.1-2.

Table 2.1-2. Stochastic variable parameters.

Distribution Function	No. of Parameters	Meaning of Parameters
Uniform	2	(1) Minimum value. (2) Maximum value.
Normal	2	(1) Mean value. (2) Standard deviation.
Exponential	1	(1) Expected value.
Lognormal	2	(1) Mean value. (2) Standard deviation.

Example 2.1.1-2. A flow simulation containing stochastic parameters.

```
NORMAL. V
STORAGE. VMEAN,VSTD,P
(1-2). RV=V(VMEAN,VSTD)
       FLOW=RV*(P-X(2))
789    end-of-record separator2/
VMEAN=0.01 $ VSTD=0.01 $ P=20. $ X(1)=100. $ X(2)=5. $
TSTRT=0. $ TEND=100. $ DT=1. $
PLOT. (X(1)=1),(X(2)=2)
```



The parameter RV in the above example might represent the value of some variable which was experimentally determined to have a mean value of 0.01 with a standard deviation of 0.01. The above flow would be computed

---

2/ An end-of-run separator is a single card with a 7-8-9 multipunched in column 1.

using a randomly sampled value from a normal distribution with the given mean and standard deviation at each time step of the simulation. Refer to section 2.1.2 for a description of flow definitions.

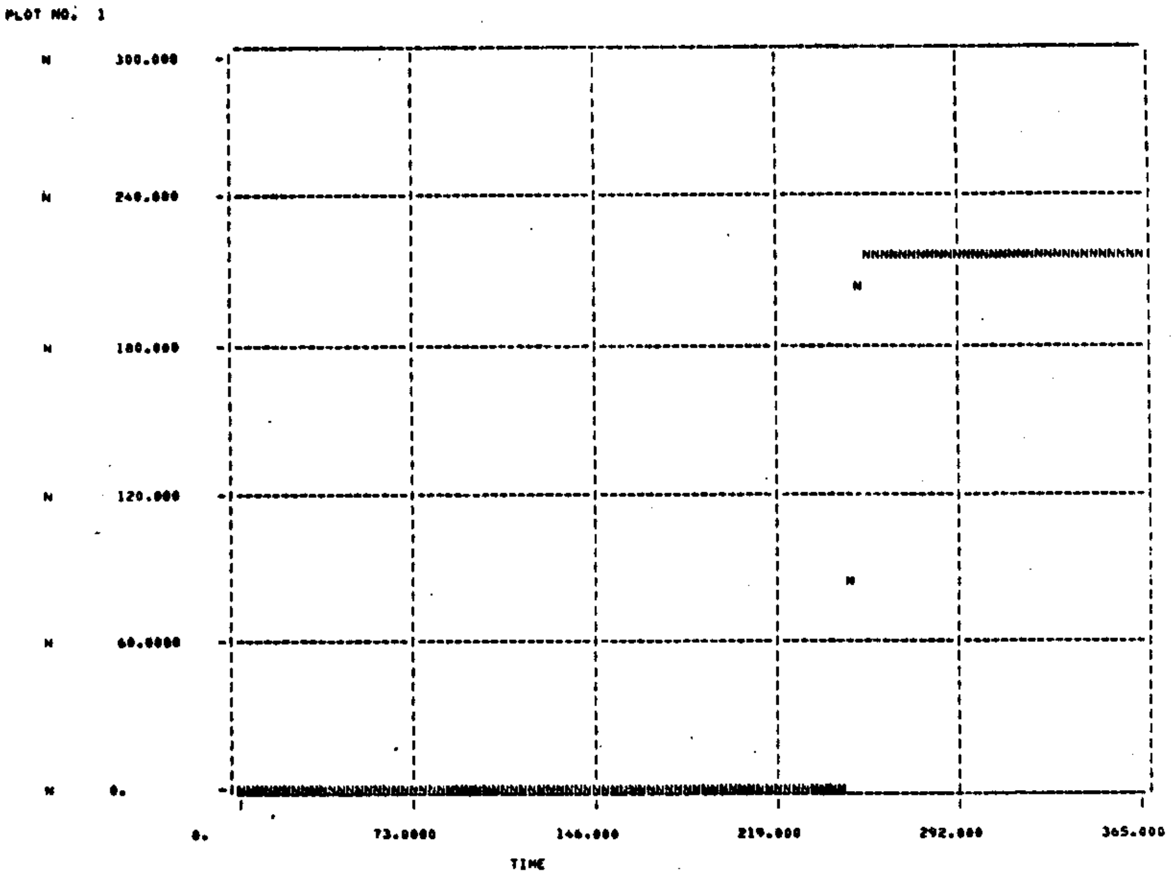
A description of each of the above distributions and the method used for their generation is given by Naylor et al. (1966).

Example 2.1.1-3. An event simulation using stochastic parameters.

```
STORAGE. RMIN,RMAX,NO,TEXP,FINAL
UNIFORM. SIZE
EXPONENT. TDELT
      EVENT MIGRT
      TNEXT=TIME+TDELT(TEXP)
      IF(TNEXT.GT.FINAL) RETURN
      NO=NO+SIZE(RMIN,RMAX)
      CALL EVENT(5HMIGRT,TNEXT,1)
      RETURN
      END
```

$^{7}8_{9}$ *end-of-record separator*

```
RMIN=10. $ RMAX=35. $ NO=0 $ TEXP=1. $ FINAL=255. $
TSTRT=0. $ TEND=365. $
EVENT. MIGRT,245.,1
PLOT. (NO=N)
```



This example might simulate the immigration of a species of animal during the time interval from day 245 to day 255. The time interval between arrivals of groups of animals was assumed to be exponentially distributed with an expected value of one. The number of animals per group was assumed to be a uniformly distributed random variable in the range from 10 through 35. The total number of animals which have arrived is contained in the variable NO. Refer to section 2.1.3 for a description of event definitions.

2.1.2 Flow definitions

Flows or material transfers between compartments, named X(j) where $1 \le j \le 999$, in a subsystem are computationally defined in flow definitions. A flow definition is comprised of a flow definition label followed by a series of one or more FORTRAN statements which compute the flow rate. The reserved variable FLOW should be set to the computed value of the flow rate. Flow definitions in general take the following form:

```
     (phrase - phrase).
          executable FORTRAN statements
```

In the above, the terms "phrase" are each one of the following forms:

(1) n where n is an integer constant.

(2) $v = n_1, n_2$ where v is a simple integer variable and n_1 and n_2 are integer constants.

(3) $v = n_1, n_2, n_3$ where v is a simple integer variable and the n_i are integer constants, $i = 1, 2, 3$.

(4) $v_1 = n_1 * v_2 \pm n_2$ where the v_i are simple integer variables and the n_i are integer constants, $i = 1, 2$.

The above phrases specify the indices of the source and destination state variable compartments between which a flow occurs. The system allows for a maximum of 999 compartments (i.e., X(1) through X(999)). The maximum number of flows which may be defined is 9999 or is limited by the amount of central memory core storage available. A flow definition label may begin in any column and must be completed in or before column 72. Any nonblank characters following the period in or before column 72 are assumed to be an executable FORTRAN statement.

Constant phrases.

If either of the phrases in a flow definition label are of the form (1) above, n must be an integer constant in the range $1 \le n \le 999$. The following flow label would define a flow from compartment X(3) to compartment X(239).

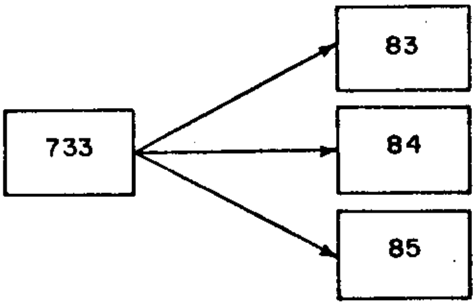
(3-239).

executable FORTRAN statements

Iterative phrases.

Phrases of the form (2) and (3) define flows iteratively. If the mathematical form of a series of flows is identical, perhaps differing only in the values of parameters used in the computation, the flows can be economically written. The phrases of forms (2) and (3) correspond in operation to the iteration phrase of a FORTRAN DO-loop. Phrases of the form (2) would indicate a series of compartments n_1, $n_1 + 1$, $n_1 + 2, \ldots, n_2$. These are the values that the integer variable v takes on. Admissable values of the constants n_1 and n_2 must satisfy $1 \le n_1 \le n_2 \le 999$. Phrases of the form (3) would indicate a series of compartments $n_1, n_1 + n_3, n_1 + 2 * n_3, n_1 + 3 * n_3, \ldots, n_1 + m * n_3$ where m is the smallest value such that $n_1 + m * n_3 \ge n_2$. Admissable values of the constants n_1, n_2 and n_3 must satisfy $1 \le n_1 \le n_2 \le 999$ and $n_1 + m * n_3 \le 999$. Source and destination compartment phrases may contain any combination of forms (1), (2), and (3). The integer valued variable v must be a simple integer variable containing from 1 to 5 characters. The following flow declarations illustrate some of the possible combinations. A flow diagram of the flows defined by each declaration is included with each case.

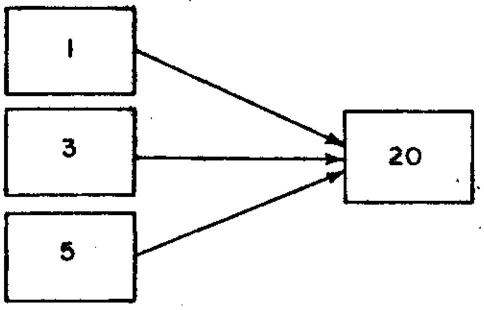
Case 1.



(733 - I = 83, 85).

executable FORTRAN statements

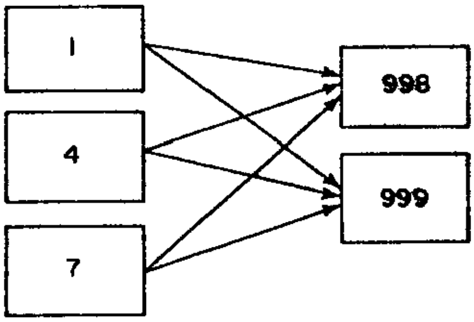
Case 2.



(KK = 1, 5, 2 - 20).

executable FORTRAN statements

Case 3.



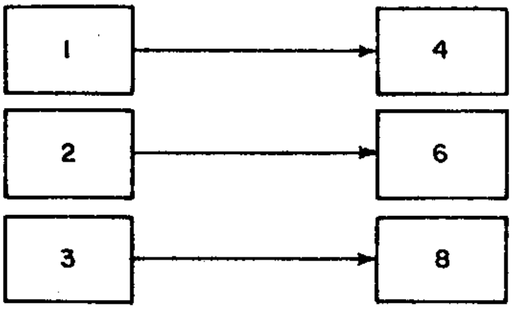
(IFROM = 1, 7, 3 - ITO = 998, 999).

executable FORTRAN statements

Computational phrases.

Phrases of the form (4) must be used in conjunction with iteration phrases of the forms (2) and (3). The variables v_1 and v_2 must be simple integer variables containing five or fewer alphanumeric characters. The variable v_2 must be the same variable used in the other half of the flow definition. The constants n_1 and n_2 must be simple integer constants. If either of the constants n_1 and n_2 are chosen to have the value zero, the zero must be written. That is, a computational phrase must appear exactly as specified in form (4). The values of the constants n_1 and n_2 must be chosen such that the values of v_1 satisfy $1 \leq v_1 \leq 999$. The following declarations illustrate the usage of computational phrases of form (4). A flow diagram of the flows defined follows each case.

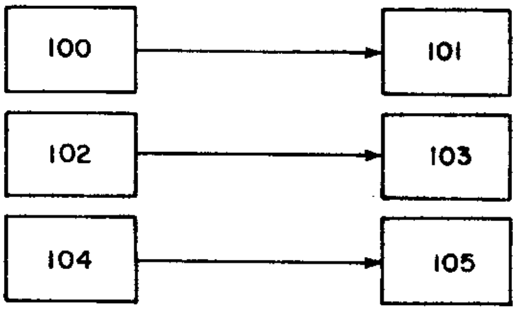
Case 1.



```
(I = 1, 3 - J = 2 * I + 2).
```

executable FORTRAN statements

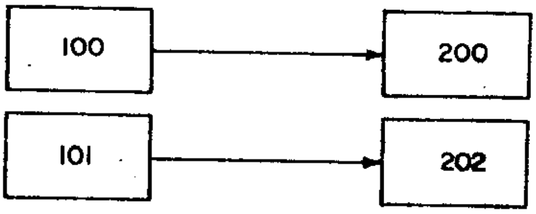
Case 2.



(M = 1 * N - 1 - N = 101, 105, 2).

executable FORTRAN statements

Case 3.



(I1 = 100, 101 - I2 = 2 * I1 + 0).

executable FORTRAN statements

The statements which follow a flow definition label can be any executable FORTRAN statements with the following restrictions.

(1) FORTRAN statement labels containing five numeric characters beginning with "9" should not be used. Any FORTRAN statement label which is not of the form 9DDDD, where the D's are any digits, may be used.

(2) FORTRAN transfer of control statements (i.e., conditional or unconditional jumps) should not transfer control to statements not contained within the range of the current flow definition label. The range of a flow definition label is defined as all executable FORTRAN statements following the flow definition label prior to encountering (i) another flow definition label, (ii) a parameter declaration statement (refer to section 2.1.1), (iii) a SUBROUTINE, FUNCTION, or EVENT statement (refer to sections 2.1.3 and 2.1.4), or (iv) the end of the source program.

(3) The reserved system variable FLOW is set equal to the computed value for the flow rate. The value of FLOW does not have to be set via an arithmetic replacement statement. FLOW may be passed as a formed parameter to a subprogram where its value is set. If within a flow definition FLOW is not assigned a value, its value is flagged as INDEFINITE and a fatal error will occur (refer to section 2.3).

Construction of flow simulations.

The design and construction of flow-oriented continuous variable simulations might be described by the following steps.

(1) Construct a flow diagram of the system to be simulated.

(2) Develop mathematical equations which will compute the values for each of the flow rates.

(3) Program the flow definition statements.

(4) Execute, debug, and evaluate the output of the simulation.

The following soil water model, taken from Smith (1971), is presented as an example of a compartmental flow model. The model is designed to simulate the following processes:

(1) infiltration of surface water.

(2) surface water runoff.

(3) transfer of soil water from unsaturated to saturated storage elements.

(4) soil water drainage.

Evapotranspiration is not considered in the model. This model is presented primarily to illustrate the implementation of the model in SIMCOMP. A complete discussion of the theory and performance of the model is presented in Smith (1971). A flow chart of the model is presented in Fig. 2.1.2-1. The following verbal description of the operation of the model is excerpted from the above mentioned report.

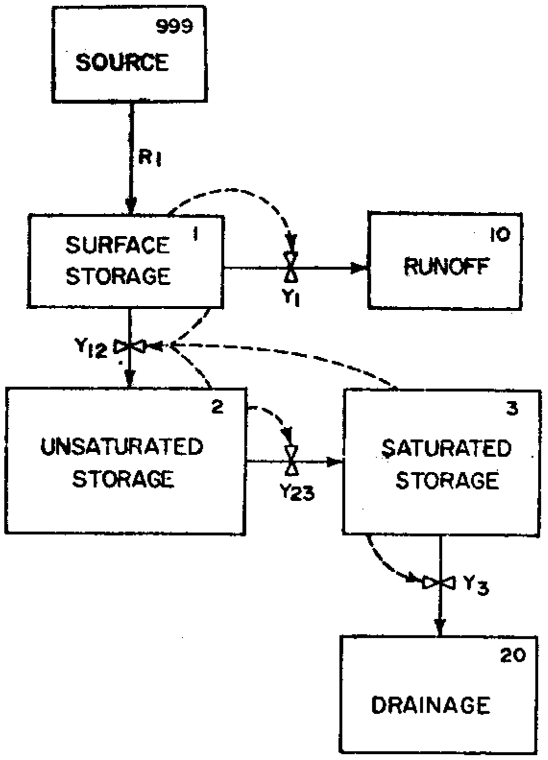


Fig. 2.1.2-1. Flow chart of the volumetric threshold infiltration model.

The compartments X(1), X(2), and X(3) represent the volumes for depression storage, unsaturated soil water storage, and the total volume between the unsaturated storage and saturation. The input to the system is the rain rate R1. The outputs of the system are the runoff rate Y1 and the drainage rate Y3. Ordinarily, evapotranspiration would draw water from compartment X(2). The flow between X(1) and X(2) is the actual infiltration rate Y12 and the flow between X(2) and X(3) is also the actual infiltration rate Y23.

If the rain rate is greater than the potential infiltration rate FP or if there is ponded water at the surface, i.e., X(1) is greater than zero, Y12 is equal to the potential infiltration rate. If the rain rate is nonzero and is less than the potential infiltration rate and X(1) is zero, Y12 is equal to the rain rate.

If X(1) is less than the surface storage capacity K1, the runoff rate Y1 is zero. When X(1) attempts to exceed K1, the runoff rate is equal to the difference between the rain rate and the infiltration rate.

When X(2) is less than the capacity of unsaturated storage K2, Y23 is zero. When X(2) approaches K2, steady state is reached for that storage, i.e., input equals output, and Y23 is equal to Y12. When X(3) is greater than zero, the drainage rate Y3 is equal to the saturated hydraulic conductivity (the final infiltration rate, FC).

A listing of the source and data sections of the model and the output produced during a 9-hour simulation comprised of two rainfall events is presented in the following example. The graphs presented were reproduced from microfilm which was generated by SIMCOMP.

Example 2.1.2-1. A sample simulation illustrating flows.

```
STORAGE. RAIN(2,24)
STORAGE. A,FC
REAL. N,K1,K2,K3
STORAGE. R1,Y1,Y12,Y23,Y3,FP
C....COMPARTMENT DEFINITIONS.
C
C      X(1)            DEPRESSION STORAGE.
C      X(2)            UNSATURATED STORAGE.
C      X(3)            SATURATED STORAGE.
C      X(10)           TOTAL RUNOFF.
C      X(20)           DEEP STORAGE.
C      X(999)          SOURCE OF RAINFALL.
C
C....VARIABLE DEFINITIONS.
C
C      K1              VOLUME OF DEPRESSION STORAGE.
C      K2              VOLUME OF UNSATURATED STORAGE.
C      K3              VOLUME OF SATURATED STORAGE.
C      FC              SATURATED HYDRALIC CONDUCTIVITY
C                           (FINAL INFILTRATION RATE).
C      N               POTENTIAL INFILTRATION EXPONENT.
C      A               POTENTIAL INFILTRATION COEFFICIENT.
C      R1              RAIN RATE.
C      FP              POTENTIAL INFILTRATION RATE.
C      Y1              RUNOFF RATE.
C      Y12             ACTUAL INFILTRATION RATE (X(1) TO X(2)).
C      Y23             ACTUAL INFILTRATION RATE (X(2) TO X(3)).
C      Y3              DRAINAGE RATE.
C      RAIN            RAIN RATE DATA RECORD.
C
C...THE RAIN RATE IS LINEARLY INTERPOLATED FROM DATA.
(999-1). R1=ALINT2(TIME,IFCK,RAIN)
      FLOW=R1
C...INFILTRATION TO UNSATURATED STORAGE.
(1-2). FP=A*(K2+K3-X(2)-X(3))**N+FC
      Y12=FP
      IF (X(1).LE.0.) Y12=AMIN1(R1,FP)
      IF (Y12*DT.GT.X(1)) Y12=X(1)/DT
      FLOW=Y12
C...RUNOFF WHEN THE CAPACITY OF DEPRESSION STORAGE IS EXCEEDED.
(1-10). Y1=0.
      IF (X(1).GT.K1) Y1=AMAX1(R1-Y12,0.)
      IF ((Y12+Y1)*DT.GT.X(1)) Y1=(X(1)-Y12*DT)/DT
      FLOW=Y1
C...INFILTRATION TO SATURATED STORAGE.
(2-3). Y23=0.
      IF (X(2).GT.K2) Y23=Y12
      IF (Y23*DT.GT.X(2)) Y23=X(2)/DT
      FLOW=Y23
C...DRAINAGE WHEN THE SOIL IS SATURATED.
(3-20). Y3=AMIN1(Y23,FC)
      IF (X(3).GT.0.) Y3=FC
      IF (Y3*DT.GT.X(3)) Y3=X(3)/DT
      FLOW=Y3
```

$^{7}8_{9}$ *end-of record separator*

```
K1=0.1 $ K2=1.60 $ K3=0.15 $ A=0.65 $ N=1.19 $ FC=0.83 $
X=3*0. $ X(10)=0. $ X(20)=0. $ X(999)=1000. $
TSTRT=0. $ TEND=9. $ DT=0.02 $ DTPR=0.1 $
R1=0. $ Y1=0. $ Y3=0. $ Y12=0. $ Y23=0. $ FP=0. $
RAIN=0.,0.,0.1,2.6,0.2,5.6,0.3,6.0,0.4,5.8,0.5,4.6,0.6,3.4,0.7,2.0,0.8,1.0,
    0.9,0.4,1.0,0.,5.,0.,5.05,1.3,5.1,2.8,5.15,3.0,5.2,2.9,5.25,2.3,5.3,1.7,
    5.35,1.0,5.4,0.5,5.45,0.2,5.5,0. $
PRINT.
PRINT. R1,FP,Y1,Y12,Y23,Y3
TITLE.          RAINFALL RATE GENERATED FROM DATA.
PLOT. (R1)
TITLE.   DEPRESSION STORAGE = 1,  RUNOFF RATE = Y,  ACTUAL INFILTRATION RATE = A
PLOT. (Y1=Y),(X(1)=1),(Y12=A)
TITLE.   UNSATURATED = 2,  SATURATED = 3,  POTENTIAL INFILTRATION RATE = P
PLOT. (X(2)=2),(X(3)=3),(FP=P)
TITLE.   DEPRESSION STORAGE = 1,  RUNOFF RATE = Y,  ACTUAL INFILTRATION RATE = A
PLOT. (Y1=Y),(X(1)=1),(Y12=A)[0.,2.]
TITLE.   DEPRESSION STORAGE = 1,  RUNOFF RATE = Y,  ACTUAL INFILTRATION RATE = A
PLOT. (Y1=Y),(X(1)=1),(Y12=A)[4.5,6.]
TITLE.   UNSATURATED = 2,  SATURATED = 3,  POTENTIAL INFILTRATION RATE = P
PLOT. (X(2)=2),(X(3)=3),(FP=P)[0.,2.]
TITLE.   UNSATURATED = 2,  SATURATED = 3,  POTENTIAL INFILTRATION RATE = P
PLOT. (X(2)=2),(X(3)=3),(FP=P)[4.5,6.]
FILM.
```

SIMULATION RESULTS

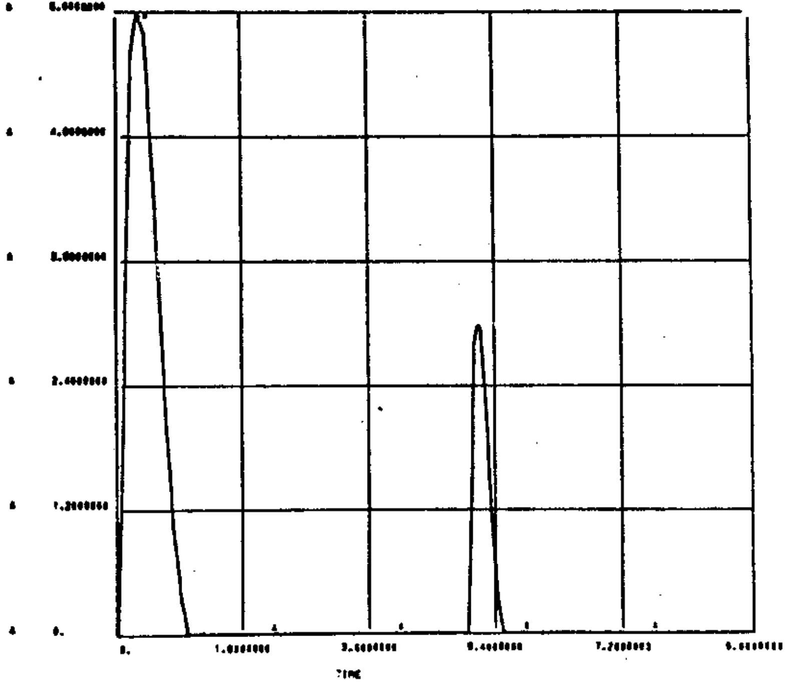
(partial listing)

```
TIME = 0.
        X(999) =  1000.00000         X(1) =              0         X(2) =              0         X(10) =              0
        X(3) =              0        X(20) =             0         R1 =               0          FP =                0
        Y1 =               0         Y12 =               0         Y23 =              0          Y3 =                0

TIME =  .100000000
        X(999) =  999.896000         X(1) =  .416000000E-01        X(2) =  .624000000E-01        X(10) =              0
        X(3) =              0        X(20) =             0         R1 =  2.00000000              FP =  2.06831609
        Y1 =               0         Y12 =  1.56000000             Y23 =              0          Y3 =                0

TIME =  .200000000
        X(999) =  999.404000         X(1) =  .113176426            X(2) =  .297248000            X(10) =  .185575574
        X(3) =              0        X(20) =             0         R1 =  5.60000000              FP =  1.87494158
        Y1 =  3.72508042             Y12 =  1.87494158             Y23 =              0          Y3 =                0

TIME =  .300000000
        X(999) =  998.820000         X(1) =  .120000000            X(2) =  .475640767            X(10) =  .584359233
        X(3) =              0        X(20) =             0         R1 =  6.00000000              FP =  1.72540424
        Y1 =  4.19459576             Y12 =  1.72540424             Y23 =              0          Y3 =                0

TIME =  .400000000
        X(999) =  998.232000         X(1) =  .120000000            X(2) =  .640008860            X(10) =  1.00799114
        X(3) =              0        X(20) =             0         R1 =  5.80000000              FP =  1.59111732
        Y1 =  4.20888268             Y12 =  1.59111732             Y23 =              0          Y3 =                0

TIME =  .500000000
        X(999) =  997.724000         X(1) =  .120000000            X(2) =  .791777277            X(10) =  1.36422272
        X(3) =              0        X(20) =             0         R1 =  4.60000000              FP =  1.47044433
        Y1 =  3.12955567             Y12 =  1.47044433             Y23 =              0          Y3 =                0

TIME =  .600000000
        X(999) =  997.336000         X(1) =  .120000000            X(2) =  .932220648            X(10) =  1.61177935
        X(3) =              0        X(20) =             0         R1 =  3.40000000              FP =  1.36196648
        Y1 =  2.03803352             Y12 =  1.36196648             Y23 =              0          Y3 =                0

TIME =  .700000000
        X(999) =  997.080000         X(1) =  .120000000            X(2) =  1.06240353            X(10) =  1.73751647
        X(3) =              0        X(20) =             0         R1 =  2.00000000              FP =  1.26445921
        Y1 =  .739540793             Y12 =  1.26445921             Y23 =              0          Y3 =                0

TIME =  .800000000
        X(999) =  996.940000         X(1) =  .116462491            X(2) =  1.18359088            X(10) =  1.75993871
        X(3) =              0        X(20) =             0         R1 =  1.00000000              FP =  1.17687545
        Y1 =               0         Y12 =  1.17687545             Y23 =              0          Y3 =                0

TIME =  .900000000
        X(999) =  996.876000         X(1) =  .675565142E-01        X(2) =  1.29650470            X(10) =  1.75993871
        X(3) =              0        X(20) =             0         R1 =  .400000000              FP =  1.09833736
        Y1 =               0         Y12 =  1.09833736             Y23 =              0          Y3 =                0

TIME = 1.00000000
        X(999) =  996.860000         X(1) =  .170002901E-14        X(2) =  1.38006129            X(10) =  1.75993871
        X(3) =              0        X(20) =             0         R1 =  .852651283E-13          FP =  1.03000669
        Y1 =               0         Y12 =  .800000000E-01         Y23 =              0          Y3 =                0

TIME =  1.10000000
        X(999) =  996.860000         X(1) =              0         X(2) =  1.38006129            X(10) =  1.75993871
        X(3) =              0        X(20) =             0         R1 =               0          FP =  1.02906174
        Y1 =               0         Y12 =               0         Y23 =              0          Y3 =                0
```

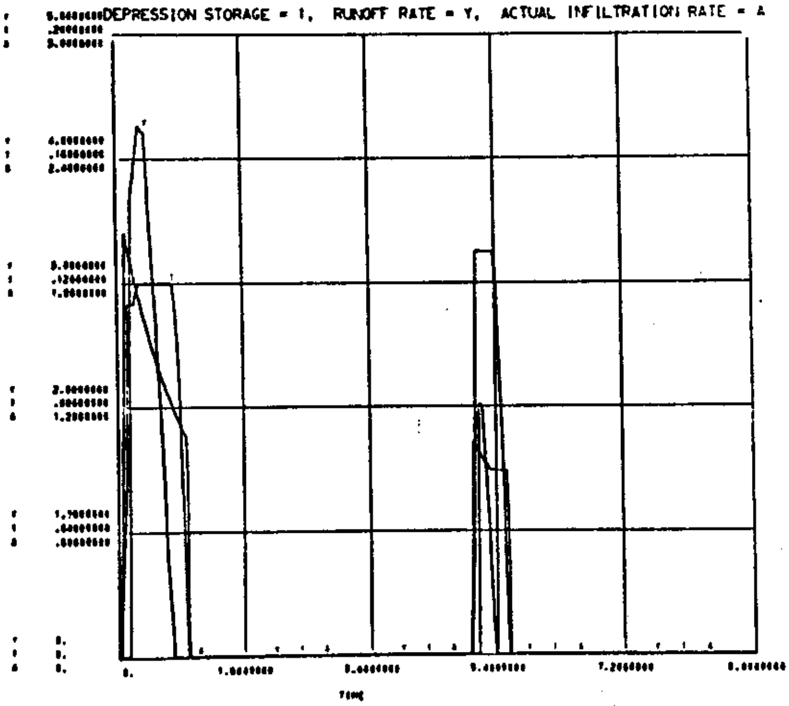
```
      GRAPHICAL SIMULATION RESULTS                    07/23/73                 19.11.52.

GRAPH   GROUP   GROUP RANGE DECLARATION   DEPENDENT VARIABLE(S)  PLOTTED     INDEPENDENT VARIABLE   INDEPENDENT VARIABLE
NO.                                                              CHARACTER                          RANGE DECLARATION
  ..........................................................................................................

  1       1                                 NJM                    M           YEAR
                                            NJF                    F
                                            NJ                     J
..........................................................................................................

  2       1                                 NAM                    M           YEAR
                                            NAF                    F
                                            NA                     A
..........................................................................................................

  3       1                                 NJ                     J           YEAR
                                            NA                     A
                                            N                      N
..........................................................................................................

  4       1                                 NJ                     J           YEAR
..........................................................................................................

  5       1                                 NA                     A           YEAR
..........................................................................................................

  6       1                                 N                      N           YEAR
```

PLOT NO. 1

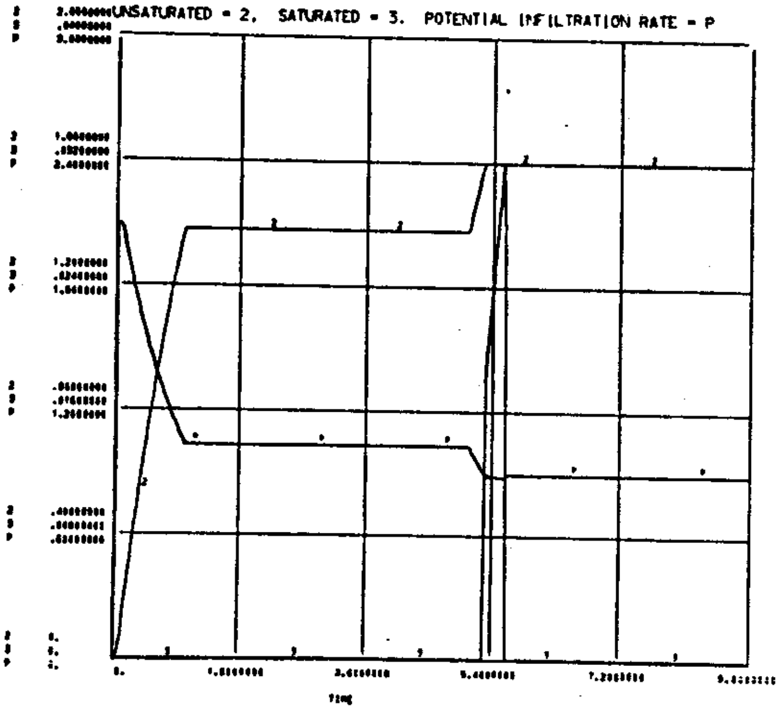
RAINFALL RATE GENERATED FROM DATA.



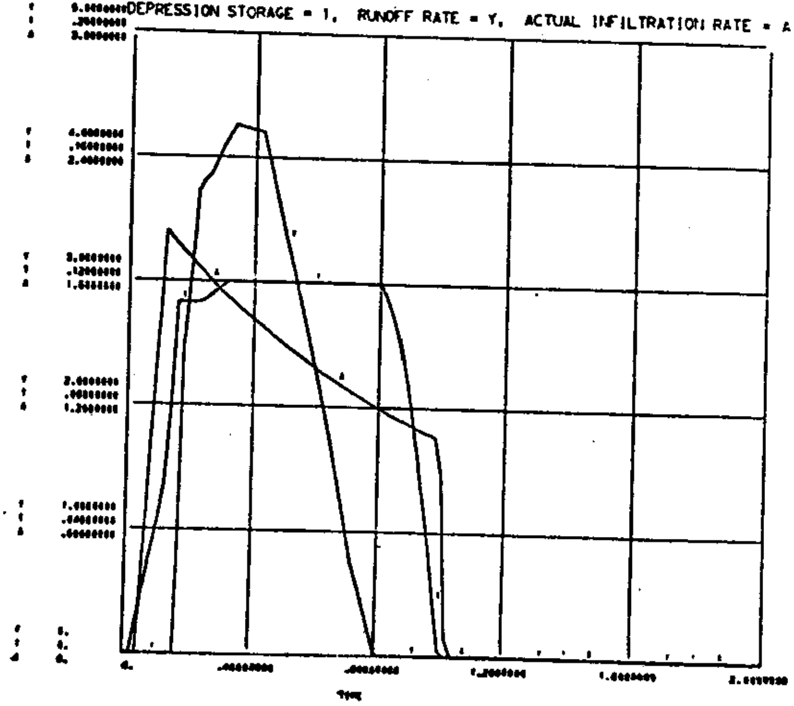
PLOT NO. 2



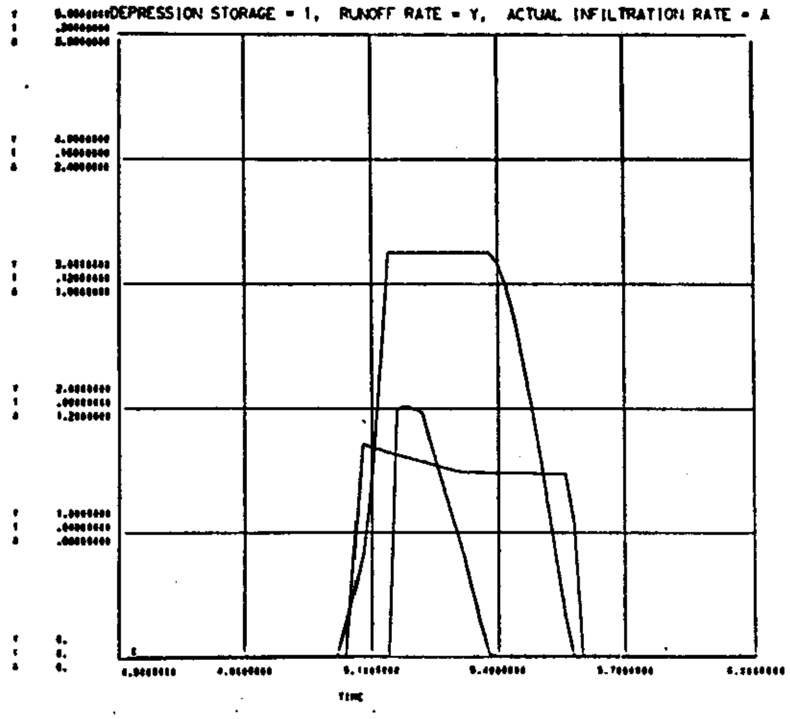
PLOT NO. 3



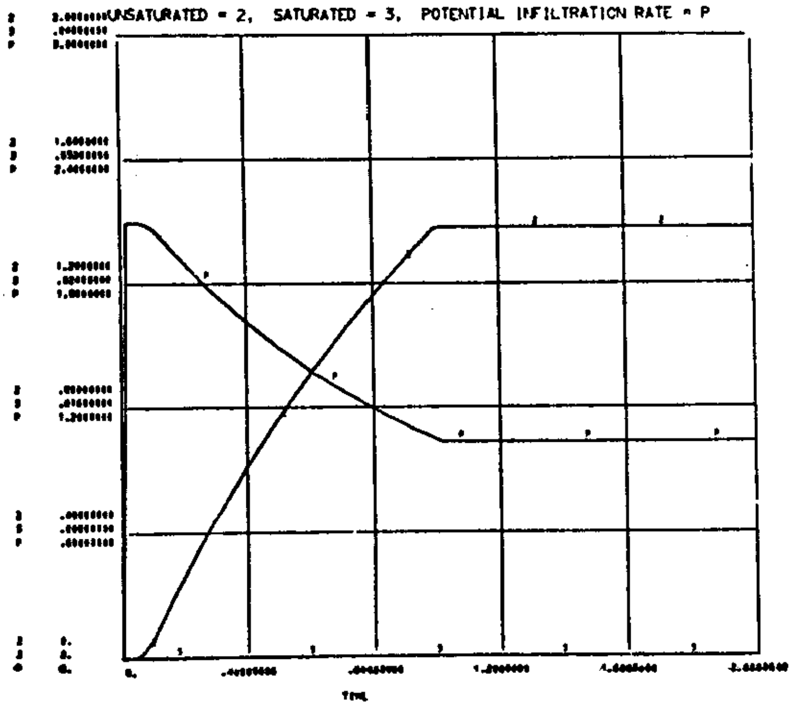
PLOT NO. 4

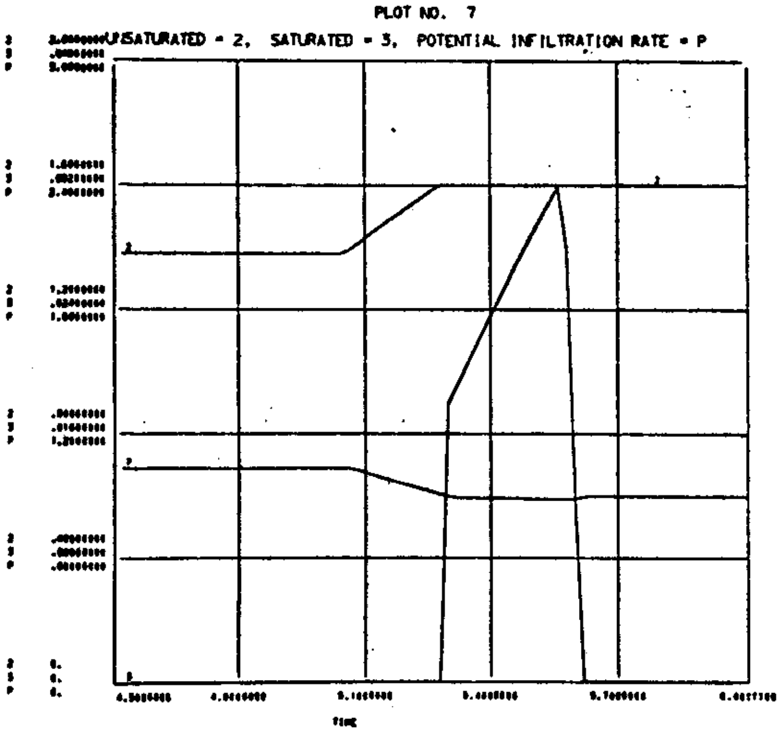


PLOT NO. 5



PLOT NO. 6





PLOT NO. 7

UNSATURATED = 2, SATURATED = 3, POTENTIAL INFILTRATION RATE = P

2.1.3 Event definitions.

An event in the SIMCOMP language is defined as a set of computations which may be scheduled for execution at any instant during simulated time. Event routines are essentially FORTRAN subroutines which are called by the executive routine at requested times. The format of an event routine is:

```
EVENT name
    :
  FORTRAN statements
    :
END
```

All statements must begin in or after column 7. Columns 72 through 80 are ignored. Column 6 may be used for statement continuation. The name of the event *name* must contain from one to five alphanumeric characters starting with a letter. Event names beginning with the character X should be avoided. The FORTRAN statements which represent the computations in the event must be followed by an END card. All variables which have been declared in parameter declaration statements (refer to section 2.1.1), in addition to the system reserved variables, may be considered present in the event and may be used in the computations. A maximum of 100 different events can be *defined* in a simulation.

Event scheduling.

SIMCOMP simulations are executed under the control of an executive routine. This executive routine has the responsibility of stepping the simulation through time. In addition to scheduling and passing control to a number of system-defined events such as printing output, saving values of variables for plotting, and updating the state variables if flows are included, the SIMCOMP executive keeps a dynamic list of all user-defined events scheduled to occur and their time of occurrence. This list is termed the event stack.

An event can be scheduled to occur either exogenously (externally) or endogenously (internally). Exogenous events are defined as those events which are scheduled prior to the start of simulated time. By including an exogenous event request card in the data (refer to section 2.2.3), an event, its time of occurrence, and a priority is entered into the event stack. Exogenous event request cards have the following format.

```
EVENT. name, time, priority
```

Endogenous events are defined as those events which are scheduled dynamically during the course of a simulation. An event is placed in the event stack by a FORTRAN call to the system event schedule in the following format,

```
CALL EVENT(mHname,time,priority)
```

where,

m	is a character count of the number of characters contained in the name of the event ($1 \leq m \leq 5$).
name	is the name of the event routine (left justified). The term mHname is a FORTRAN hollarith constant.
time	is a real-valued variable or constant containing the value of simulated time at which the event is to occur.
priority	is an integer variable or constant in the range 1 through 512.

After a call of the above form is made and the current simulated time, TIME, becomes equal to the scheduled time of occurrence of the event, the event is called and executed. When an event is called by the executive routine, the corresponding entry in the event stack is purged. The priority of an event is used as a tie breaker if more than one event is scheduled to occur at the same time. A priority of 1 is highest (first to occur) and 512 is lowest (last to occur). If the value of a priority is outside the range 1 through 512, a priority of 512 is assumed. If two or more events of the same priority are scheduled to occur at the same time, the first to have been scheduled is the first to occur. Additionally, if the same event is scheduled to occur more than once at identical times, the second and subsequent requests are ignored. The maximum number of events that can be scheduled at any one time is limited by the amount of core available to the job. If an attempt to schedule a nonexistent event is made, a diagnostic is issued and the simulation is terminated.

Once an event has been put into the event stack, the event may be canceled at any time prior to the time of occurrence. This is accomplished with a FORTRAN call of the following form,

```
CALL CANCEL(mHname,dummy,status)
```

where,

m	is a character count of the number of characters contained in the name of the event ($1 \leq n \leq 5$).
name	is the name of the event routine (left justified).
dummy	is a dummy argument which is not used, but must be included for compatibility with calls to EVENT.
status	is an integer variable which is used to signal the status of the cancellation operation to the user.

Upon return from the cancellation routine status contains,

0	if the routine was found in the event stack and was successfully canceled.

1 if the routine was not found in the stack and no action was taken.

2 if the event stack was empty.

If an event is scheduled to occur more than once and a call to CANCEL is made, the entry which was first to occur is removed from the event stack.

System-defined events.

A number of events are defined and scheduled by the system. The user should be made aware of these events for the following reason. In some simulations various system actives are sometimes scheduled to occur at the same time as user-defined events. Most notable of these is the system routine which produces printed output. If a user's routine was scheduled to occur at the same time as the system's printing routine but at a lower priority, then the printed output would not reflect the state of the system after events scheduled at that time have occurred. Table 2.1.3-1 contains a list of the system routines, their scheduling priority, and the system-defined variable which controls their time of occurrence. Some of the routines listed are user-defined special purpose subroutines described in section 2.1.4.

Table 2.1.3-1. System-defined events.

Routine Name	Priority	Controlling Variable	Action
START[a]	100	TSTRT	User-supplied.
XPRNT	200	DTPR	Prints tabular output.
XPLOT	200	DTPL	Saves values of variables for plotting.
XCSIM	300	DT	CYCL1 is called if included by the user, the flows are computed, the state variables are updated, and CYCL2 is called if included by the user.
FINIS[a]	500	TEND	User-supplied.
HALT	512	TEND	Halts execution.

[a] User-supplied routines, scheduled by the system.

The system-defined routine HALT can be scheduled by the user any time he desires the simulation terminated. A choice of priorities for scheduling events should be made with the above table in mind. In most situations the user will desire to schedule his events at a higher priority than the system events (i.e., less than 100). As indicated in Table 2.1.3-1, a number of system-defined events are scheduled according to the values given the reserved system control variables. If values for these variables are needed by the system, but have not been set by the user, default values will be supplied. A complete discussion of the system-control variables and their use in controlling simulations is contained in section 2.2.1.

Construction of event simulations.

The construction and operation of event-oriented simulations is illustrated by considering a simple-event simulation of a hypothetical population. The simulation is not intended to be biologically realistic. The processes to be considered are (i) births, (ii) recruitment from the juvenile age class to the adult age class, and (iii) deaths. The following variables are the variables of interest in the simulation.

NJM - No. of juvenile males.

NJF - No. of juvenile females.

NJ - Total no. of juveniles.

NAM - No. of adult males.

NAF - No. of adult females.

NA - Total no. of adults.

N - Total population.

Each of the processes or events in the simulation consist of two sets of computations. These computations are (i) computations which reflect changes in the variables of interest due to the processes being simulated and (ii) computations which determine the time at which an event will occur. The particular equations used to compute these two quantities embody the assumptions about the processes involved in the population. For the sake of clarity in this example we will assume a very simple description for the processes influencing the population dynamics.

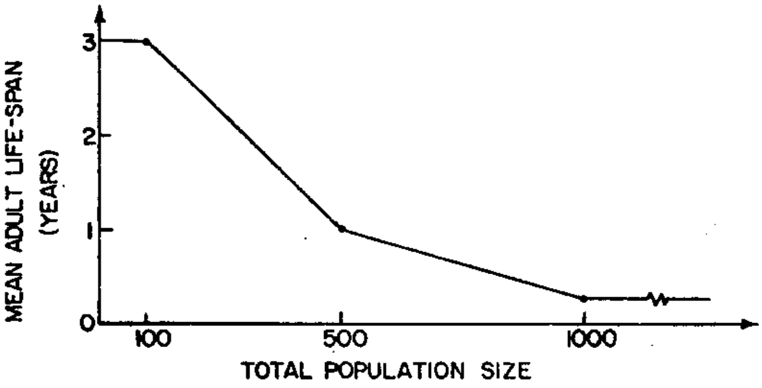
(1) Births are assumed only to occur from the 90th to the 120th day of each year. We assume that 80% of all female adults have offspring during this time interval and that the number of offspring per female occurs in the following proportions.

No. of offspring/birth	Percent occurrence
1	5%
2	80%
3	10%
4	5%

We further assume that males are born as often as females. Therefore during the 30 days of natality the number of birth events which occur on the average is 0.8 * NAF. Hence the average time between births is 30./(0.8 * NAF). The standard deviation of the time between births is assumed to be 10% of the mean.

(2) Recruitment from the juvenile age class to the adult age class is based on the assumption that the mean time required for a juvenile to mature is 365 days with a standard deviation of 20 days.

(3) Deaths are assumed to occur according to the following graph of mean adult lifespan vs. total population size.



The standard deviation of the mean adult lifespan is assumed to be 10% of the mean. Table 2.1.3-2 lists the events required for this simulation and the actions performed by each event.

Table 2.1.3-2. Population simulation events.

Event	Name	Computations
(1) Birth	BIRTH	(a) No. of offspring/birth (b) Sex of each offspring (c) Adjust population size (d) Schedule time of maturation (e) Schedule time of next birth
(2) Male/female maturation	RCRTM/ RCRTF	(a) Adjust age class sizes. (b) Schedule time of death
(3) Male/female death	DETHM/ DETHF	(a) Adjust population size

A complete listing of the simulation and the results are contained in example 2.1.3-1. Since the simulation contains stochastic elements, the results shown in the output represent only one of the many relaizations of the simulation which would be required for an exhaustive analysis of the model. Whenever a variable in a simulation is defined stochastically, the value of the variable at any point in the simulation is obtained by the random sampling of a value from the indicated distribution function. Therefore any one run of the simulation represents only one possible realization of the system which is being modeled. If the statistical properties of the variables of interest are desired, a number of runs using different random number sequences would be required.

Example 2.1.3-1. A sample simulation illustrating events.

```
STORAGE. NJM,NJF,NJ,NAM,NAF,NA,N,YEAR
UNIFORM, FRCT
NORMAL. TSMP
      EVENT BIRTH
      YEAR=TIME/365.
C...BIRTH EVENT.  DETERMINE THE NUMBER OF OFFSPRING.
      F=FRCT(0.,1.)
      NB=1
      IF(F.LE.0.05) GO TO 5
      NB=2
      IF(F.LE.0.85) GO TO 5
      NB=3
      IF(F.LE.0.95) GO TO 5
      NB=4
C...INCREMENT THE POPULATION VARIABLES AND SCHEDULE RECRUITMENT.
    5 DO 20 I=1,NB
C...SAMPLE THE TIME OF RECRUITMENT OF THE OFFSPRING.
   10 TRC=TSMP(365.,20.)
      IF(TRC.LE.0.) GO TO 10
C...INCREMENT THE TOTAL POPULATION SIZE AND NO. OF JUVENILES.
      NJ=NJ+1
      N=N+1
C...DETERMINE THE SEX OF THE OFFSPRING.
      R=FRCT(0.,1.)
      IF(R.GT.0.5) GO TO 15
      NJM=NJM+1
      CALL EVENT(5HRCRTM,TIME+TRC,20)
      GO TO 20
   15 NJF=NJF+1
      CALL EVENT(5HRCRTF,TIME+TRC,20)
   20 CONTINUE
C...SCHEDULE THE TIME TO THE NEXT BIRTH.
      TMB=30./(0.8*NAF)
      TSB=0.1*TMB
   25 TB=TSMP(TMB,TSB)
      IF (TB.LE.0.) GO TO 25
      TY=AMOD(TIME,365.)
      IF(TY+TB.GT.120.) TB=TB+335.
      CALL EVENT(5HBIRTH,TIME+TB,20)
      RETURN
      END
      EVENT RCRTM
      YEAR=TIME/365.
C...EVENT OF THE RECRUITMENT OF A MALE JUVENILE.
      NJM=NJM-1
      NAM=NAM+1
      NJ=NJ-1
      NA=NA+1
C...SAMPLE THE LIFESPAN OF THE ADULT AND SCHEDULE THE DEATH.
      TML=ALINT2(N,IFLG,100,1095,500,365,1000,36)
      TSL=0.1*TML
    5 TD=TSMP(TML,TSL)
      IF (TD.LE.0.) GO TO 5
      CALL EVENT(5HDETHM,TIME+TD,20)
      RETURN
      END
      EVENT RCRTF
      YEAR=TIME/365.
C...EVENT OF THE RECRUITMENT OF A FEMALE JUVENILE.
      NJF=NJF-1
      NAF=NAF+1
      NJ=NJ-1
      NA=NA+1
C...SAMPLE THE LIFESPAN OF THE ADULT AND SCHEDULE THE DEATH.
      TML=ALINT2(N,IFLG,100,1095,500,365,1000,36)
    5 CALL EVENT(4HHALT,TIME,1)
      RETURN
      END
      EVENT DETHF
      YEAR=TIME/365.
C...DEATH EVENT.  IF POPULATION GOES TO ZERO THE SIMULATION IS HALTED.
      IF(N.LT.1) GO TO 5
      N=N-1
      NA=NA-1
      NAF=NAF-1
      RETURN
    5 CALL EVENT(4HHALT,TIME,1)
      RETURN
      END
      SUBROUTINE START
```

```
         TSL=0.1*TML
       5 TD=TSMP(TML,TSL)
         IF (TD.LE.0.) GO TO 5
         CALL EVENT(5HDETHF,TIME+TD,20)
         RETURN
         END
         EVENT DETHM
         YEAR=TIME/365.
C...DEATH EVENT.  IF POPULATION GOES TO ZERO THE SIMULATION IS HALTED.
         IF(N.LT.1) GO TO 5
         N=N-1
         NA=NA-1
         NAM=NAM-1
         RETURN
C...ASSUMING AN INITIAL POPULATION OF YOUNG ADULTS ONLY, SCHEDULE THEIR
C      DEATHS.
         DO 5 I=1,NAM
         TML=ALINT2(N,IFLG,100,1095,500,365,1000,36)
         TSL=0.1*TML
       4 TD=TSMP(TML,TSL)
         IF (TD.LE.0.) GO TO 4
       5 CALL EVENT(5HDETHM,TIME+TD,20)
         DO 10 I=1,NAF
         TML=ALINT2(N,IFLG,100,1095,500,365,1000,36)
         TSL=0.1*TML
       9 TD=TSMP(TML,TSL)
         IF (TD.LE.0.) GO TO 9
      10 CALL EVENT(5HDETHF,TIME+TD,20)
C...SCHEDULE THE FIRST BIRTH.
         CALL EVENT(5HBIRTH,90.,20)
         RETURN
         END
```

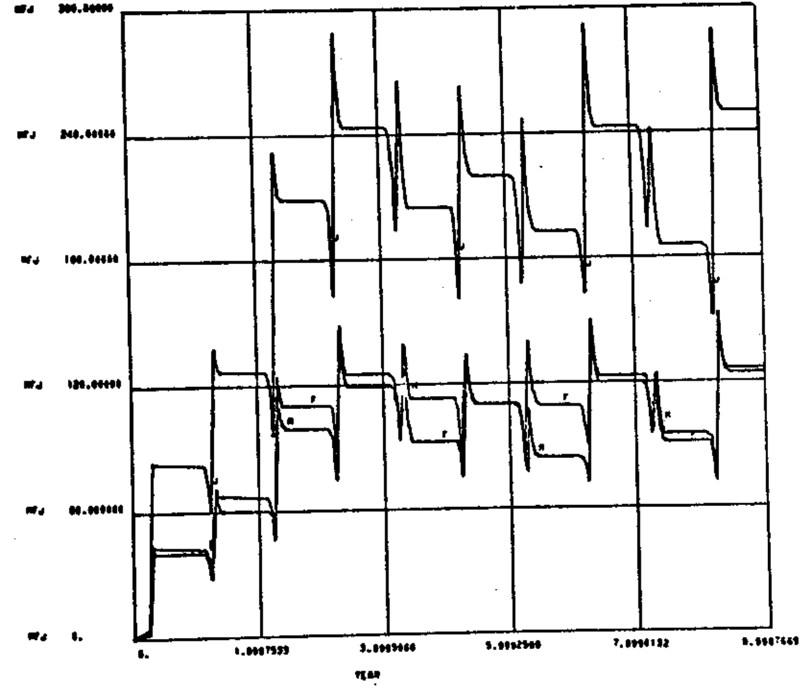
$^{7}8_{9}$ *end-of-record separator*

```
NJM=0 $ NJF=0 $ NJ=0 $ NAM=50 $ NAF=50 $ NA=100 $ N=100 $
YEAR=0. $ TSTRT=0. $ TEND=3650. $ DTPL=4.5625 $
TITLE. JUVENILE POPULATION -- TOTAL = J, MALES = M, FEMALES = F.
PLOT. (NJM=M,NJF=F,NJ=J)/YEAR
TITLE. ADULT POPULATION -- TOTAL = A, MALES = M, FEMALES = F.
PLOT. (NAM=M,NAF=F,NA=A)/YEAR
TITLE. TOTAL POPULATION -- ADULTS + JUVENILES = N, ADULTS = A, JUVENILES = J.
PLOT. (NJ=J,NA=A,N=N)/YEAR
TITLE.                          JUVENILE POPULATION.
PLOT. (NJ=J)/YEAR
TITLE.                          ADULT POPULATION.
PLOT. (NA=A)/YEAR
TITLE.                          TOTAL POPULATION.
PLOT. (N=N)/YEAR
FILM.
```

```
    GRAPHICAL SIMULATION RESULTS                07/23/73               10.11.52.

GRAPH  GROUP  GROUP RANGE DECLARATION   DEPENDENT VARIABLE(S)  PLOTTED    INDEPENDENT VARIABLE   INDEPENDENT VARIABLE
NO.                                                            CHARACTER                         RANGE DECLARATION

  ..............................................................................................................

  1      1                              NJM                     M          YEAR
                                        NJF                     F
                                        NJ                      J

.................................................................................................................

  2      1                              NAM                     M          YEAR
                                        NAF                     F
                                        NA                      A

.................................................................................................................

  3      1                              NJ                      J          YEAR
                                        NA                      A
                                        N                       N

.................................................................................................................

  4      1                              NJ                      J          YEAR

.................................................................................................................

  5      1                              NA                      A          YEAR

.................................................................................................................

  6      1                              N                       N          YEAR
```

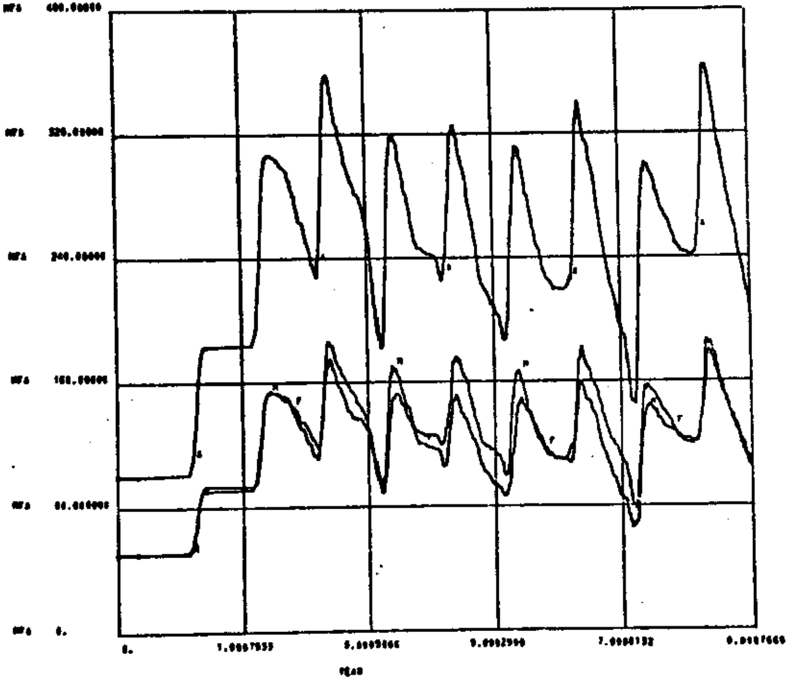
PLOT NO. 1

JUVENILE POPULATION -- TOTAL = J, MALES = M, FEMALES = F.



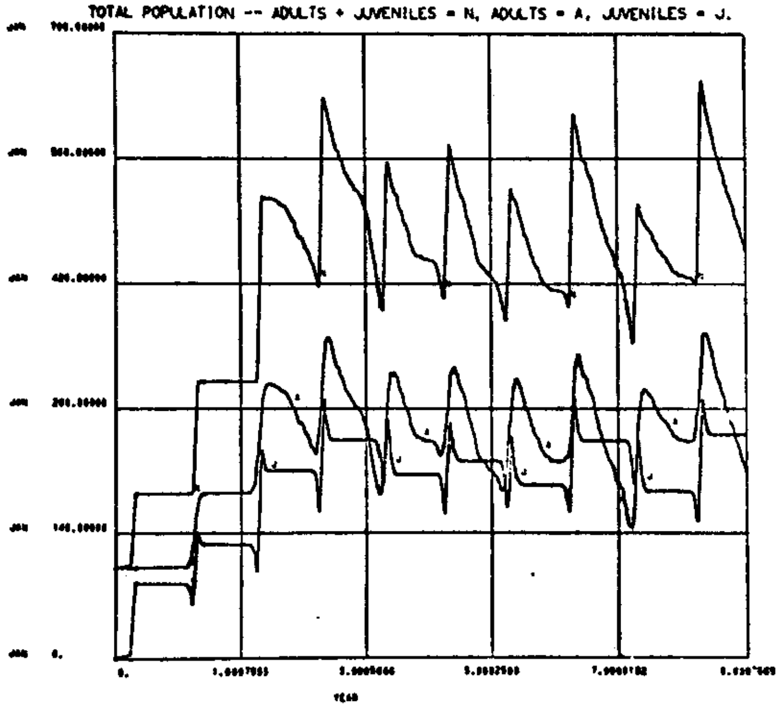
PLOT NO. 2

ADULT POPULATION -- TOTAL = A, MALES = M, FEMALES = F.



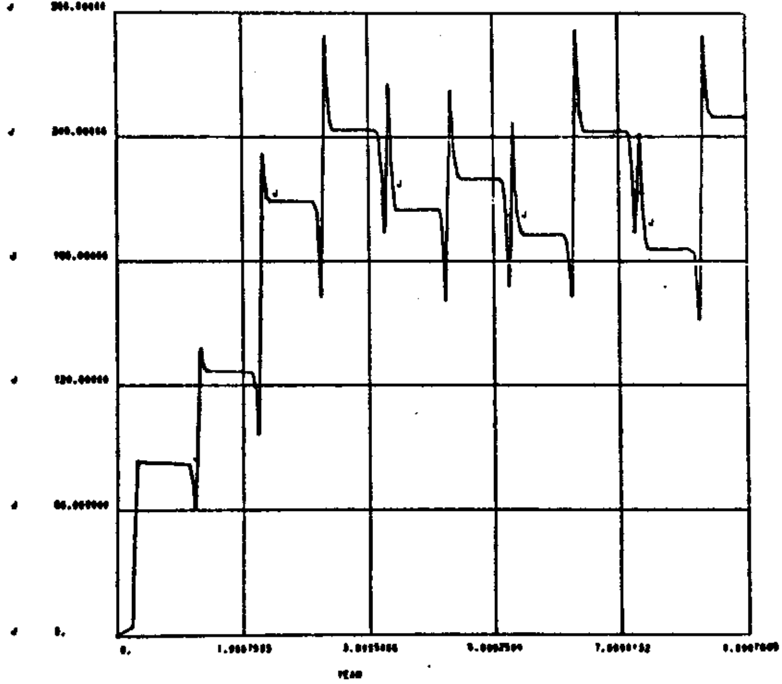
PLOT NO. 3

TOTAL POPULATION -- ADULTS + JUVENILES = N, ADULTS = A, JUVENILES = J.



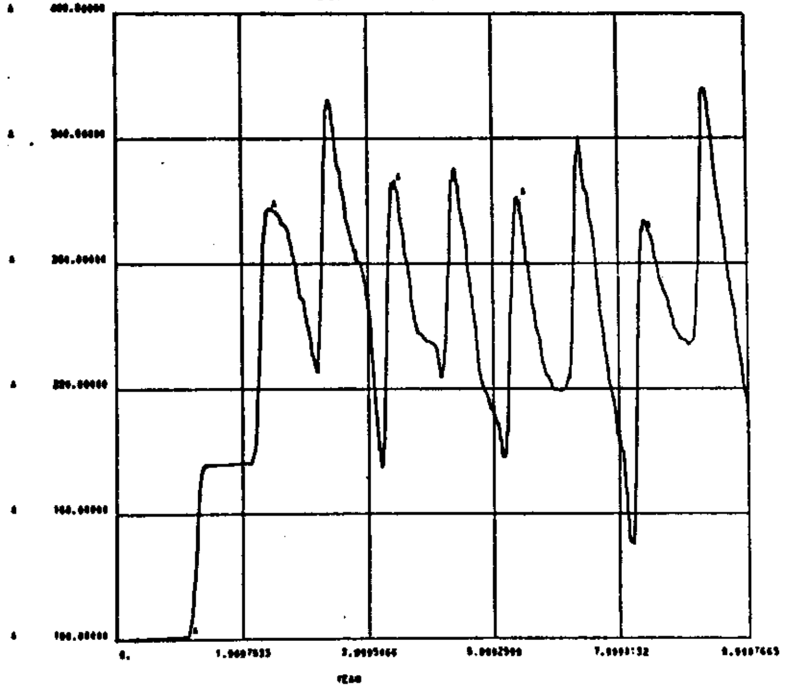
PLOT NO. 4

JUVENILE POPULATION.



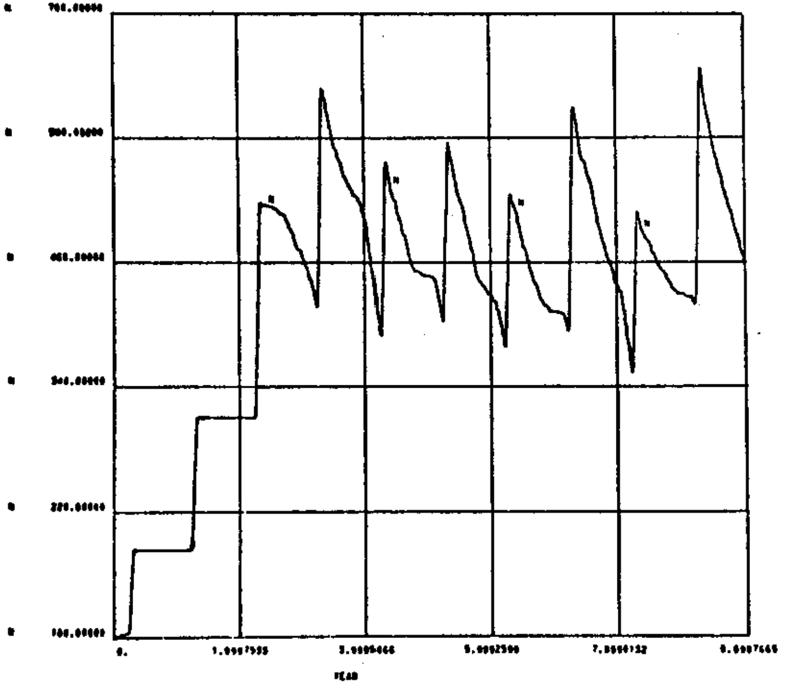
PLOT NO. 5

ADULT POPULATION.



PLOT NO. 6

TOTAL POPULATION.



2.1.4 Subprograms.

FORTRAN subroutines and functions may be supplied by the programmer. Subroutines and functions may appear anywhere in the source section provided they do not appeear within the intended range of a flow definition label. Parameter declaration statements may not be included within the text of a subprogram. All reserved system control variables (cf. Table 2.1-1) and all user-defined variables and stochastic functions declared in parameter declaration statements should be considered globally defined and are accessible within every user-supplied subprogram. *These variables are in a common block inserted by SIMCOMP into each of these routines.* The format of user-supplied subprograms conforms to FORTRAN specifications for subroutines and functions. A user-supplied subprogram can be called from within any flow definition or other subprogram or event.

Certain special purpose subroutines can be supplied by the user which will be called by the SIMCOMP executive timing routine at predetermined times in the simulation. These reserved routine names are listed in Table 2.1.4-1. Computations which are to be performed at the specified times are included in a FORTRAN subroutine appropriately named. Since special-purpose subroutines are called by the executive routine, argument lists are not allowed. A flow chart of the execution sequence in a simulation containing flow definitions is given in Fig. 2.1.4-1.

Table 2.1.4-1. Reserved subroutine names

Subroutine Name	Use
START	Called after parameter values have been set by data assignment statements in the data section just prior to the start of execution, TIME = TSTRT.
CYCL1[a]	Called just prior to the computation of flows at each time step.
CYCL2[a]	Called after the flows have been computed and the state variables have been updated, but prior to any printing or storing of values for plotting at each time step.
FINIS	Called at the end of simulation, TIME = TEND.

[a] Called only if flow definitions are present.

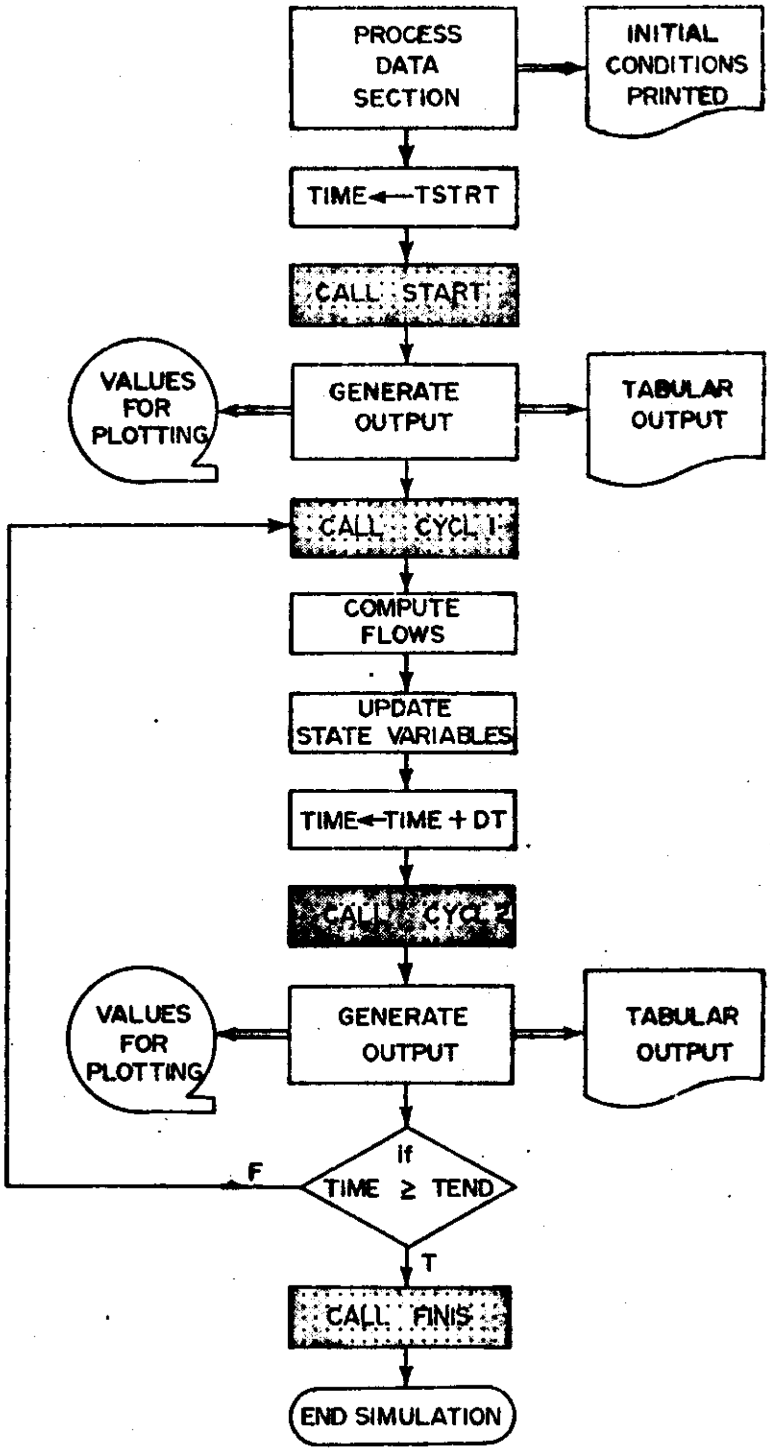


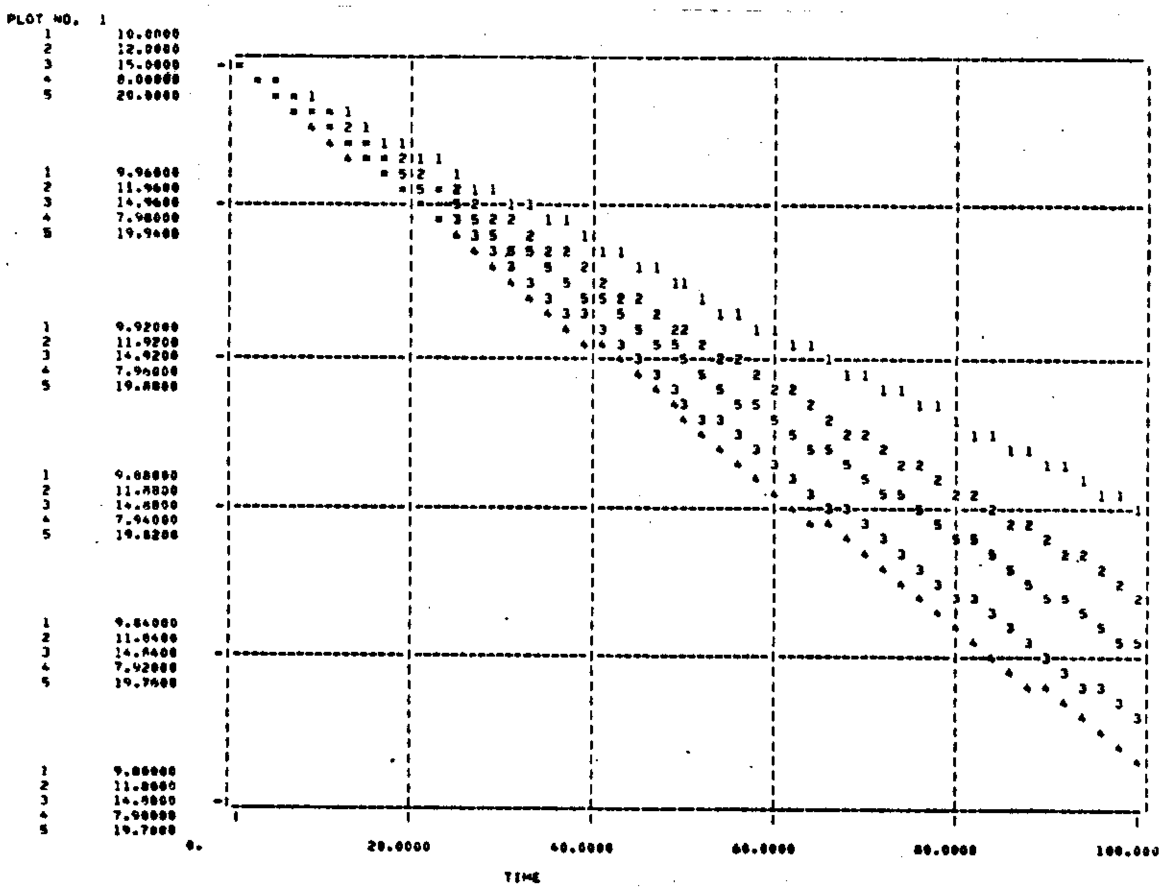
Fig. 2.1.4-1. Flow execution sequence.

Example 2.1.4-1. An example of a user-defined subroutine. The graph was reproduced from a printer plot generated by SIMCOMP.

```
STORAGE. V(5).P1.P2
(I=101.105-263).
     CALL VCALC
     J=I-100
     FLOW=V(J)*P1/DT
     SUBROUTINE VCALC
     DO 10 I=1.5
     J=I+100
  10 V(I)=X(J)*P2
     RETURN
     END
```

$^7 8_9$ *end-of-record separator*

```
X(101)=10.,12.,15.,8.,20. $ X(263)=1000. $ P1=0.0001 $ P2=2.363 $
TSTRT=0. $ TEND=100. $ DT=2. $
PLOT. (X(101)=1),(X(102)=2),(X(103)=3),(X(104)=4),(X(105)=5)
```



Note that the variables in the STORAGE. statement are globally defined and are available for use in the subroutine and in the flows.

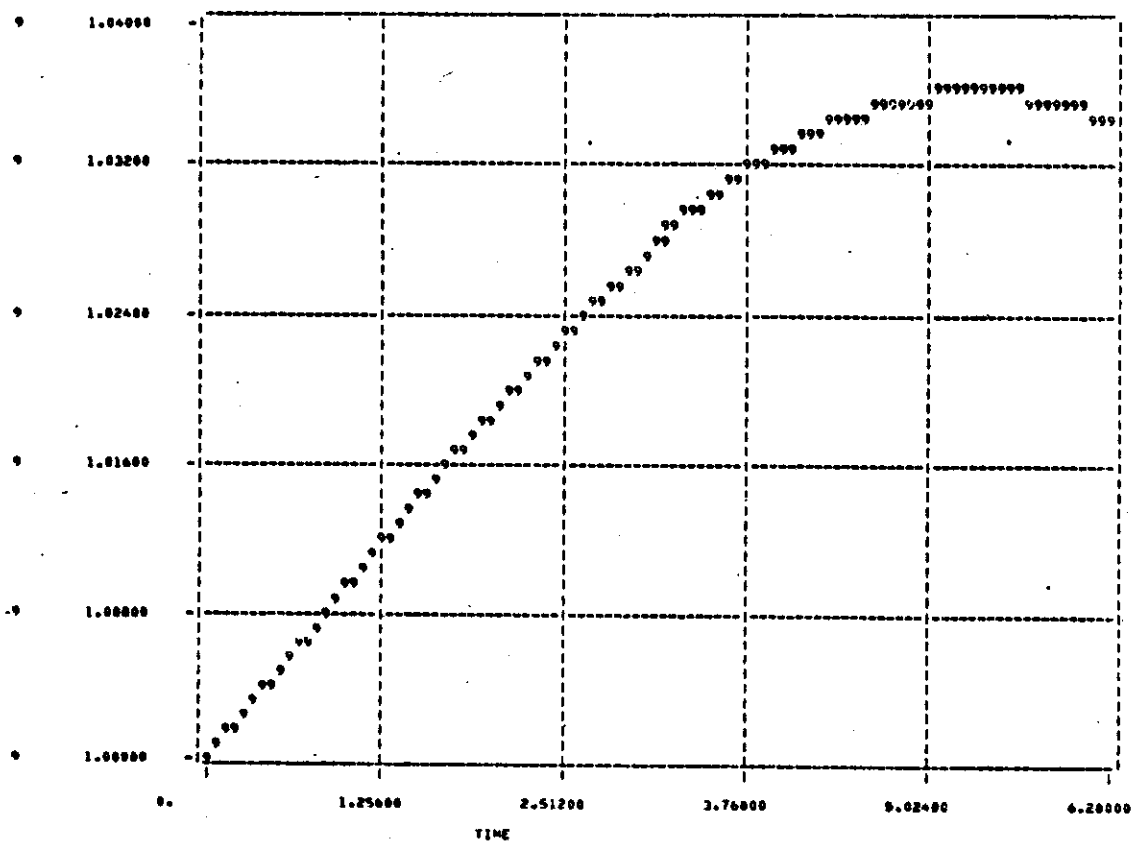
Example 2.1.4-2. An example of a user-supplied function.

```
STORAGE. A,T1,T2,PHI,C1,C2
      FUNCTION COSX(T)
      COSX=A*COS((T-T2)/3.14+PHI)
      RETURN
      END
      SUBROUTINE CYCL1
      C1=X(9)*COSX(TIME-T1)
      C2=0.1*C1
      RETURN
      END
(1-9). FLOW=C1
(2-9). FLOW=C2/X(2)
```

$^{7}8_{9}$ *end-of-record separator*

```
A=0.01 $ T1=20. $ T2=10. $ PHI=3.14 $ C1=0. $ C2=0. $
TSTRT=0. $ TEND=6.28 $ DT=0.0628 $
X(1)=100. $ X(2)=100. $ X(9)=1. $
PLOT. (X(9)=9)
```

PLOT NO. 1



2.1.5 Utility routines.

A library of subroutines and functions is available which is accessed and made available to the user by SIMCOMP. Whenever the user includes a call to one or more of the utility routines and does not supply a subprogram by the same name, the utility library is accessed and the called routine or routines are loaded. The routines currently available include the functions reported by Parton and Innis (1972). The calling sequences for their functions are compatible with the FORTRAN listings provided therein. The utility routines listed in Table 2.1.5-1 are also available, and a description of their use follows.

Table 2.1.5-1. Utility routines.

Name	Purpose
ALINT1	Linear interpolation of data whose independent variable is regularly spaced.
ALINT2	Same as ALINT1 for unevenly spaced data points.
FLOWV	Returns the most recently computed value for a particular flow.
XSTATS	Statistical sampling package
PUNCHD	Produces a punched deck of data in SIMCOMP acceptable format.

Linear interpolation.

Many times the most desirable way to specify a function is by a table which is linearly interpolated. The FORTRAN callable functions ALINT1 and ALINT2 will interpolate a table of values producing a value of the dependent variable for any given value of the independent variable.

Mathematically, the operation of ALINT1 and ALINT2 is described.

Given:

x[3/] - the value of the independent variable at which point a linearly interpolated value is to be computed.

n - the number of pairs of values in the interpolation table.

x_j - the j^{th} value of the independent variable in the table to be interpolated.

y_j - the j^{th} value of the dependent variable in the table corresponding to x_j.

In order for the linear interpolation routine ALINT2 to operate efficiently, the values of the independent variable must be in ascending order,

$$x_j \le x_{j+1} \quad \text{for} \quad j = 1, 2, \ldots, n-1.$$

The functions ALINT1 and ALINT2 compute the linearly interpolated value as follows:

[3/] The use of lower case x in this section should not be confused with the state variables X.

if $x \leq x_1$ then ALINT = y_1

if $x \geq x_n$ then ALINT = y_n

if $x_j \leq x < x_{j+1}$ for $j = 1, 2, \ldots, n - 1$

then

$$\text{ALINT} = y_j + \frac{y_{j+1} - y_j}{x_{j+1} - x_j} \cdot (x - x_j).$$

<u>Equal interval data</u>.

The utility function ALINT1 can be used whenever the values of the independent variable in the interpolation table are equally spaced. An example of such a table and its graph is presented in Fig. 2.1.5-1. The linearly interpolated value for $x = 0.25$ is $y(x) = 0.375$. Note in the graph that for values of the independent variable outside the range of definition of the table, that is less than 0.0 and greater than 0.5, the value of the dependent variable is assumed to be equal to the value of the function at its tabular end point. Therefore $x = 0.55$ produces $y(x) = 0.05$. Whenever extrapolation occurs, this is the action taken by ALINT1 and ALINT2.

X_j	Y_j
0.0	.1
0.1	.3
0.2	.4
0.3	.35
0.4	.25
0.5	.05

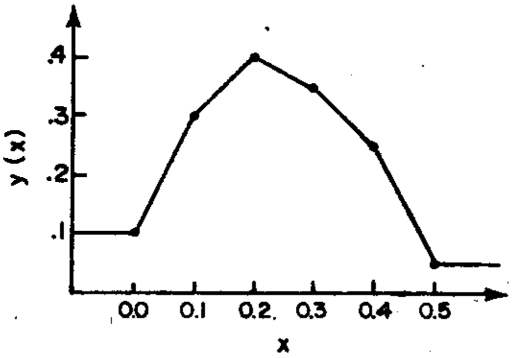


Fig. 2.1.5-1. Sample tabular function and graph containing equal interval data.

The second calling sequence to ALINT1 is,

V = ALINT1(R,IFLG,RS,RD,Y1,Y2, ... , YN)

where R, IFLG, RS, and RD are as defined in the first type of call. The constants or variables Y1 through YN correspond to the values in the array RT. The quantities Y1 through YN correspond to the values y_1, y_2, ... , y_n in the mathematical description of the interpolation algorithm. These quantities may be either integer or real valued, but again it is recommended that real values be used. There must be at least two or more entries of the dependent variable in the call(i.e., n must be greater than or equal to two). Using the second method of calling, the previous example would be programmed as follows.

Example 2.1.5-2. Use of ALINT1, type 2 call.

```
(1-2). FLOW=ALINT1(X(2),IDUM,0.,0.1,0.1,0.3,0.4,0.35,0.25,0.05)
⁷8₉    end-of-record separator
X=100.,0. $ TSTRT=0. $ TEND=10. $ DT=0.1 $
PLOT. (X(2)=2)
```

<u>Unequally spaced data</u>.

The utility function ALINT2 must be used whenever the values of the independent variable in the interpolation table are unequally spaced. In this case the values of the independent variable at each of the tabular points must be supplied in the function call. An example of an interpolation table which contains unequally spaced data is presented in Fig. 2.1.5-2.

The first type of FORTRAN calling sequence to ALINT1 is,

V = ALINT1(R,IFLG,RS,RD,RT)

where, R is the value of the independent variable at which point a linearly interpolated value is desired.

RS is the starting value of the independent variable in the table.

RD is the increment between successive values of the independent variable in the table.

RT is an array containing the tabular values of the dependent variable.

The above four quantities can be either integer- or real-valued variables or constants. It is recommended that only real values are used so that internal conversion is not required. The array RT must be declared in STORAGE., and the declared size of the array must be one word longer than the number of values in the array. The last location in the array must not be set to any value either in the data section or within the source section. By use of this convention, ALINT1 is able to determine n, the number of values in the table. Upon return from the function,

ALINT1 is the interpolated value.

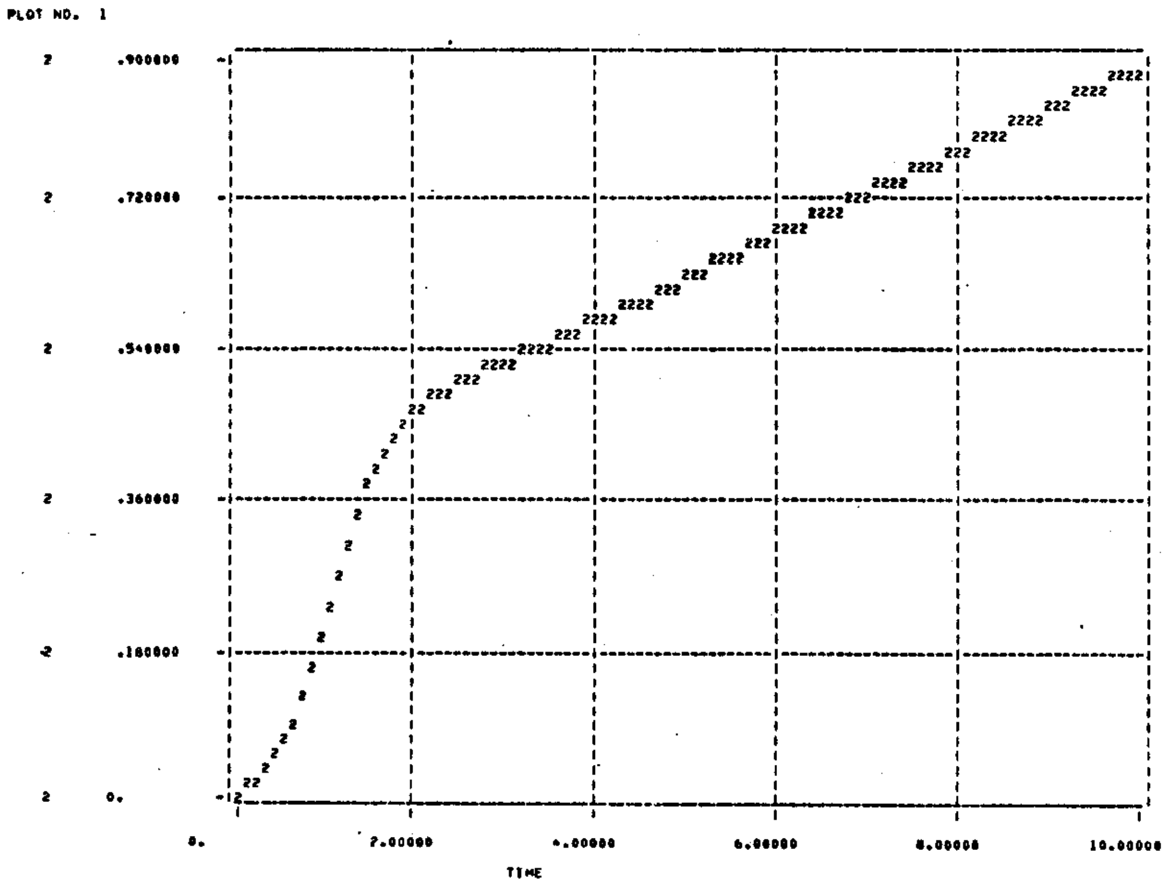
IFLG is an integer variable set by the function as a flag to the user, IFLG = 0 for normal interpolation.

IFLG = 1 if extrapolation occurred.

For the sample data in Fig. 2.1.5-1 the above calling sequence is illustrated.

Example 2.1.5-1. Use of ALINT1, type 1 call.

```
STORAGE. Y(7)
(1-2). FLOW=ALINT1(X(2),IDUM,0.,0.1,Y)
789     end-of-record separator
Y=0.1,0.3,0.4,0.35,0.25,0.05 $ X=100.,0. $ TSTRT=0. $ TEND=10. $ DT=0.1 $
PLOT. (X(2)=2)
```



X_i	Y_i
0	1
10	.7
20	.5
35	.35
50	.25
70	.2

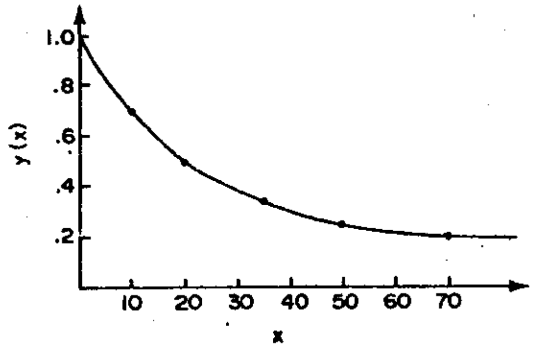


Fig. 2.1.5-2. Sample tabular function and graph containing unequally spaced data.

The first type of FORTRAN calling sequence to ALINT2 is,

V = ALINT2(R,IFLG,TABLE)

where R and IFLG are as described in the calls to ALINT1. The variable TABLE must be a two-dimensional array declared in STORAGE. and is dimensioned TABLE (2,n+1) where n is the number of pairs of values in the interpolation table. The entries TABLE (1,n+1) and TABLE (2,n+1) must not be given any values in the data section or in the source program. The array TABLE can be either an integer-valued or real-valued array, but it is recommended that a real-valued array be used. The location of the values of the interpolation table in the array TABLE follows,

TABLE (1,1) = x_1, TABLE (2,1) = y_1
TABLE (1,2) = x_2, TABLE (2,2) = y_2
$\vdots$ $\vdots$
TABLE (1,n) = x_n, TABLE (2,n) = y_n,

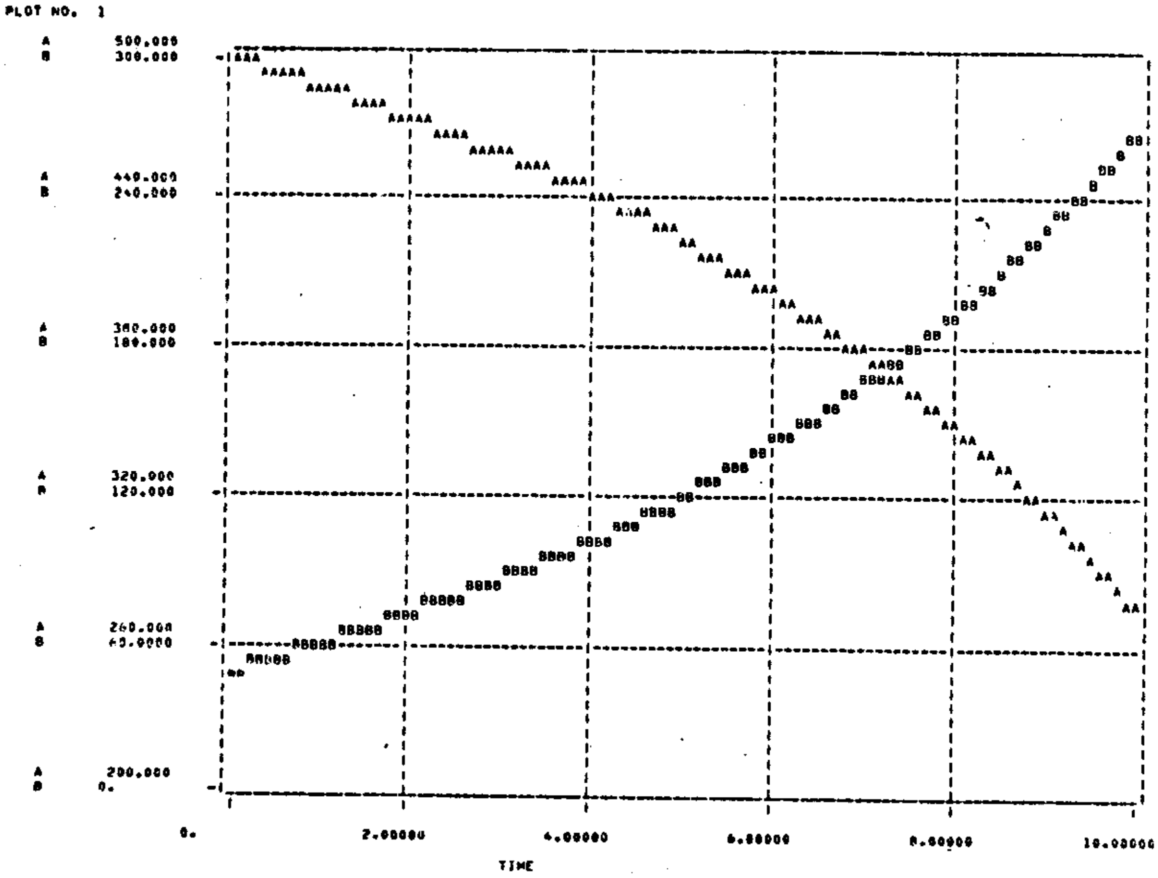
with TABLE (1,n+1) and TABLE (2,n+1) not given any values. The following example is derived from the data given in Fig. 2.1.5-2.

Example 2.1.5-3. Use of ALINT2, type 1 call.

```
STORAGE. TT(2,7)
(8-80). TV=0.8*X(80)
      TF=ALINT2(TV,IFL,TT)
      FLOW=TV*TF
      IF(TV.LT.0.) FLOW=0.
```

$^{7}8_{9}$ *end-of-record separator*

```
TT=0.,1.,10.,0.7,20.,0.5,35.,0.35,50.,0.25,70.,0.2 $
TSTRT=0. $ TEND=10. $ DT=0.1 $ X(8)=500. $ X(80)=50. $
PLOT. (X(8)),(X(80))
```



The second type of calling sequence to ALINT2 is,

V = ALINT2(R,IFLG,R1,Y1,R2,Y2, ... ,RN,YN)

where R and IFLG are as described in the calls to ALINT2 ALINT1. The constants or variables R1, Y1 through RN, YN correspond to the entries in TABLE in the first type of call to ALINT2. Referring to the mathematical description of linear interpolation the arguments R1, Y1 through RN, YN correspond to,

$$
\begin{aligned}
R1 &= x_1, & Y1 &= y_1\\
R2 &= x_2, & Y2 &= y_2\\
&\vdots & &\vdots\\
RN &= x_n, & YN &= y_n.
\end{aligned}
$$

Note that in both types of calls to ALINT2 the values of the independent variable in the interpolation table must be specified in ascending order. Using the second method of calling ALINT2, the sample data contained in Fig. 2.1.5-2 would be linearly interpolated as in the following example.

Example 2.1.5-4. Use of ALINT2, type 2 call.

```
(8-80). TV=0.8*X(80)
      TF=ALINT2(TV,IFL,0,1,10,0.7,20,0.5,35,0.35,50,0.25,70,0.2)
      FLOW=TV*TF
      IF(TV.LT.0.) FLOW=0.
789     end-of-record separator
TSTRT=0. $ TEND=10. $ DT=0.1 $ X(8)=500. $ X(80)=50. $
PLOT. (X(8)),(X(80))
```

Step functions.

The special utility function ALINT2 can be used to generate step functions. A step function f(x) is in general defined by,

$$f(x) = \begin{matrix} a & \text{if } x \le x_s \\ b & \text{if } x > x_s. \end{matrix}$$

By allowing two consecutive entries of the independent variable in the interpolation table to assume the same value, ALINT2 will generate a step. Fig. 2.1.5-3 contains the equation of a step function, its graph, and the interpolation table used for its generation. Using the second form of call to ALINT2, the step function illustrated in Fig. 2.1.5-3 would be programmed as in the following definition of a flow.

$$f(x) = \begin{cases} 0 & \text{if } x \le 3 \\ 1 & \text{if } x > 3 \end{cases}$$

x	f(x)
3	0
3	1

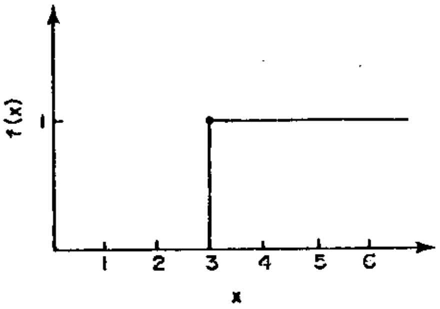


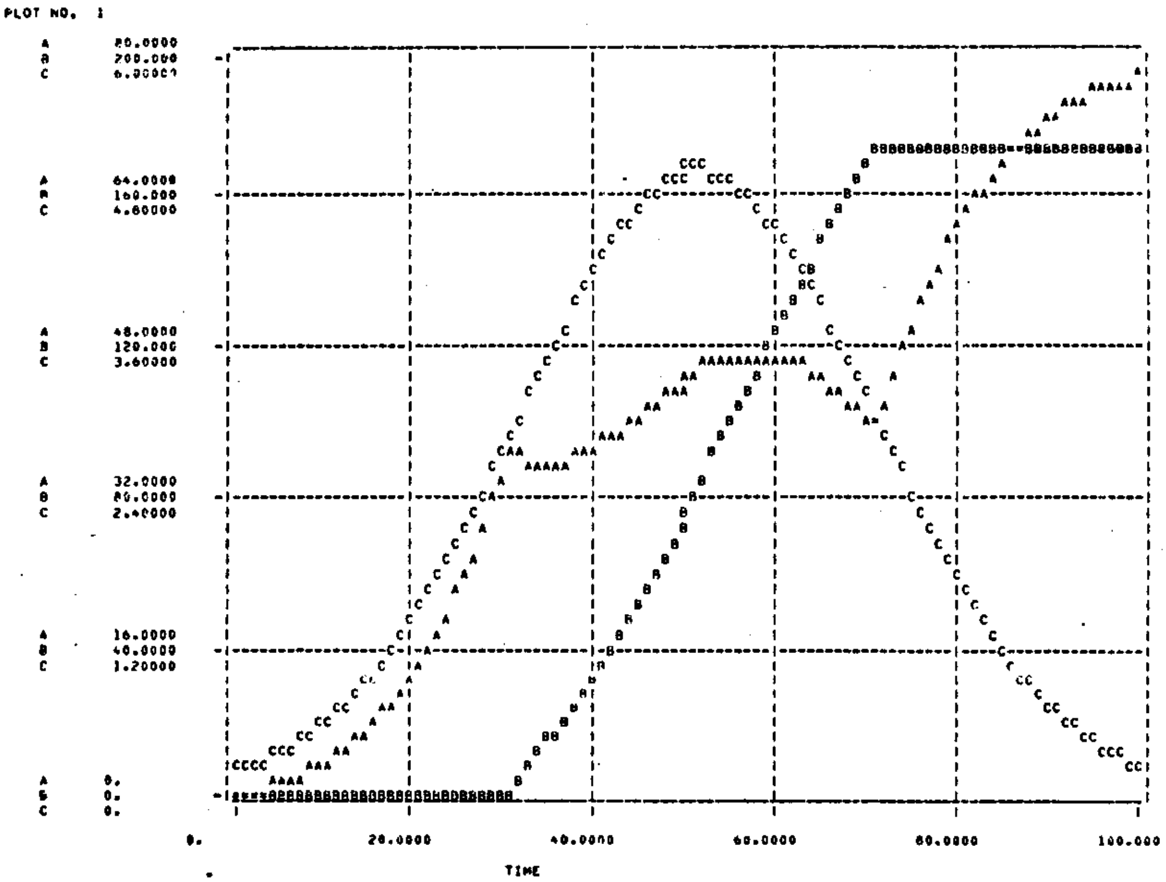
Fig. 2.1.5-3. Sample step function.

Example 2.1.5-5. Use of ALINT2 as a step function.

```
STORAGE. RAIN
(3-1). RAIN=5.106*EXP(-0.00125*(TIME-50.)**2)
      FLOW=RAIN
(1-2). FLOW=0.1*X(1)*ALINT2(RAIN,IF,3.,0.,3.,1.)
```

$^{7}8_{9}$ *end-of-record separator*

```
X=0.,0.,100. $ TSTRT=0. $ TEND=100. $ RAIN=0.22434 $ DT=1. $
PLOT. (X(1)),(X(2)),(RAIN)
```



Retrieving values of flows.

The value which is computed for a flow is not directly available to any part of a source program except within the range of the particular flow. Sometimes it is desirable to acquire the value of a particular flow or flows for use in the computation at some later time. The usual situation is when the value of one flow depends upon the value of some previously computed flow. While this situation is physically impossible (Innis 1972), it may be a very useful procedure. The special utility subroutine

FLOWV is used to access the value of a previously computed flow given the source and destination compartment indices of the desired flow. The FORTRAN calling sequence to the routine FLOWV is,

```
CALL FLOWV(I,J,VALUE,IFLAG)
```

where I is the source compartment index.

J is the destination compartment index.

VALUE is the most recently computed value of the flow.

IFLAG is a flag which signals various conditions to the user.

Table 2.1.5-2 summarizes the values which IFLAG can attain and their meanings. Any attempt to subsequently use the variable VALUE in the computation while IFLAG returning any of the values 1 through 5 will result in an arithmetic-mode error and the simulation will be abnormally terminated. By checking the value of IFLAG prior to using VALUE in the computation, an abnormal termination can be avoided. Refer to section 2.3 to determine what arithmetic operations can produce any of the conditions 2 through 4. The quantities I and J may be integer constants or variables. VALUE must be a real-valued variable. The following flow chart and example illustrate the use of FLOWV.

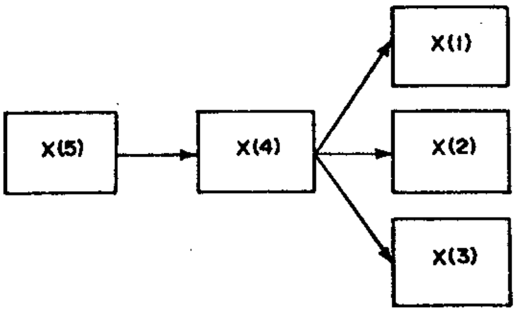


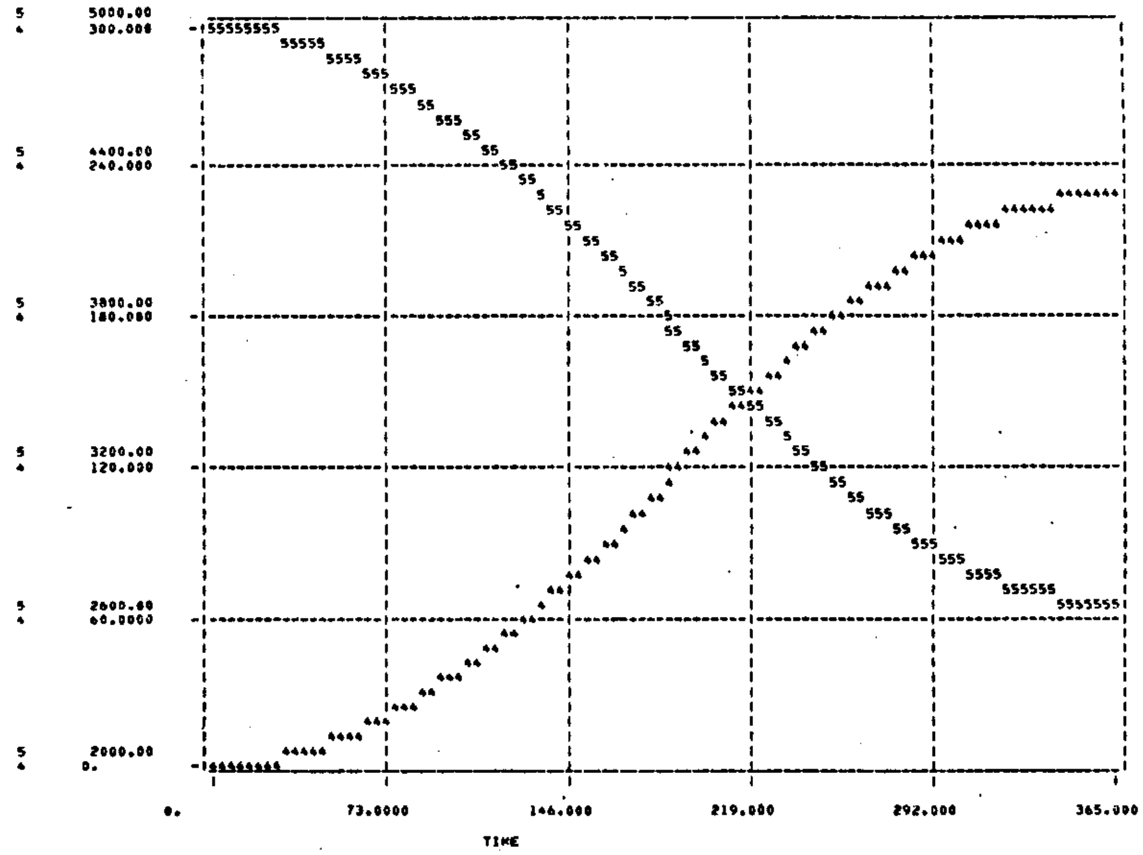
Table 2.1.5-2. Values for IFLAG on return from FLOWV.

IFLAG	VALUE	Description
0	--	Current value returned OK.
1	Indefinite	Flow exists, but was not assigned a value by the user.
2	+Infinite	Flow exists, value is positive infinite.
3	-Infinite	Flow exists, value is negative infinite.
4	Indefinite	Flow exists, value is indefinite.
5	Indefinite	Flow was not defined in the simulation.

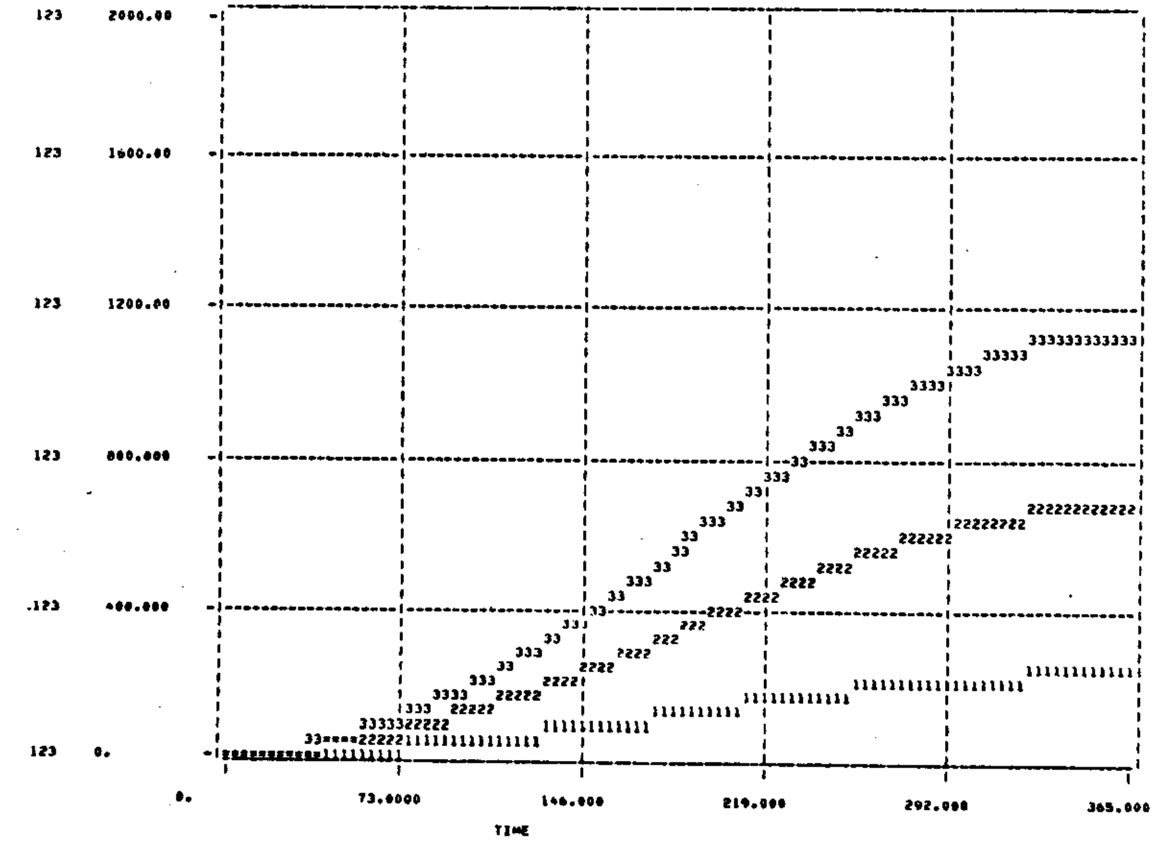
Example 2.1.5-6. Simulation containing a call to FLOWV.

```
STORAGE. R(3)
(5-4). FLOW=10.*SIN(TIME*3.14159/365.)
(4-I=1,3). CALL FLOWV(5,4,FL,ICHK)
       FLOW=0.
       IF(ICHK.NE.0) GO TO 10
       FLOW=R(I)*FL
    10 CONTINUE
789      end-of-record separator
X=4*0.,5000. $ R=0.1,0.3,0.5 $
TSTRT=0. $ TEND=365. $ DT=1. $
PLOT. (X(5)=5),(X(4)=4)
PLOT. (X(1)=1,X(2)=2,X(3)=3)
```

PLOT NO. 1



PLOT NO. 2



Statistical-sampling package.

Often the outputs of a simulation experiment are statistical measurements. Statistical measures of simulated variables through time are often the principle variables of interest in event simulations. Such quantities as the yearly average population size, the mean time to maturation, and the probability of reaching a given age are typical measures of performance of an event-oriented simulation of population dynamics. Normally, statements must be scattered throughout the program to gather such statistics. Writing the statements necessary for gathering such quantities as sums and sums of squares is a task to be avoided because it clutters the logic of simulation with statements whose only function is the collection of output information.

The statistical sampling package XSTATS provides a number of subroutines which simplify the gathering and reporting of such information. The statistical sampling routines allow the use of either of two sampling strategies. These are (i) discrete sampling of variables and (ii) time-weighted sampling of variables. The mathematical description of each of these methods is presented in Table 2.1.5-3 given the following definitions.

n - the number of samples.

x_i[4/] - the value of the i^{th} sample.

t_i - the time at which the value changed from x_i to x_{i+1}.

t_0 - the time at which sampling started.

$\Delta t_i = t_i - t_{i-1}$ - the length of time which the i^{th} sample had the value x_i.

Table 2.1.5-3. Statistical sampling computational methods.

Statistic	Discrete Sampling	Time-weighted Sampling
Number or total time	n	$\Sigma \Delta t_i$
Sum	Σx_i	$\Sigma x_i \cdot \Delta t_i$
Sum of squares	Σx_i^2	$\Sigma x_i^2 \cdot \Delta t_i$
Mean	$\frac{\Sigma x_i}{n}$	$\frac{\Sigma x_i \cdot \Delta t_i}{\Sigma \Delta t_i}$
Mean square	$\frac{\Sigma x_i^2}{n}$	$\frac{\Sigma x_i^2 \cdot \Delta t_i}{\Sigma \Delta t_i}$
Variance	$\frac{\Sigma x_i^2}{n} - \left(\frac{\Sigma x_i}{n}\right)^2$	$\frac{\Sigma x_i^2 \cdot \Delta t_i}{\Sigma \Delta t_i} - \left(\frac{\Sigma x_i \cdot \Delta t_i}{\Sigma \Delta t_i}\right)^2$
Standard deviation	$\sqrt{\text{variance}}$	$\sqrt{\text{variance}}$
Maximum	largest x_i	largest x_i
Minimum	smallest x_i	smallest x_i

Discrete sampling.

As an example of discrete sampling consider the following table of values which might have been the

---

[4/] The use of lower case x in this section should not be confused with the state variables X.

times to maturation of individuals in a simulated population.

individual	1	2	3	4	5	6
time to maturity	1.8	1.7	2.1	0.9	1.6	1.7

Table 2.1.5-4 displays the values of each of the statistics using discrete sampling.

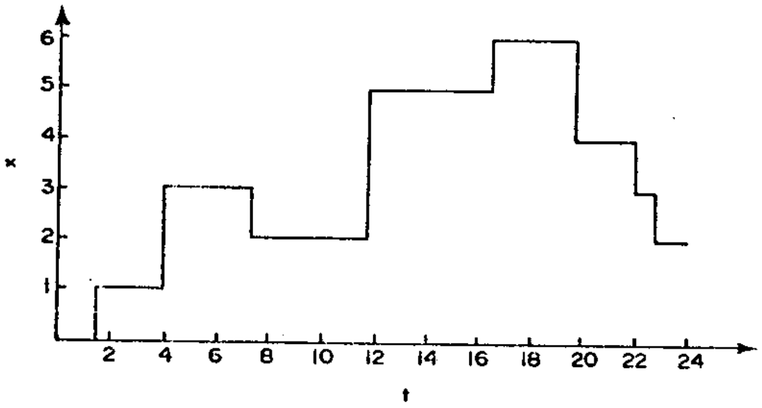
Table 2.1.5-4. Example of discrete sampling.

Statistic	Value
Number	6.0
Sum	9.8
Sum of squares	16.8
Mean	1.63
Mean square	2.8
Variance	0.132
Standard deviation	0.364
Maximum	2.1
Minimum	0.9

Time-weighted sampling.

As an example of time-weighted sampling consider the following table of values and their graph through time. The graph might be the graph of the size of a population through time. Remember that the value t_i is the time at which the sampled variable changed from x_i to x_{i+1}.

t_i	1.4	4	7.4	11.9	16.5	19.6	22	22.8	24
Δt_i	1.4	2.6	3.4	4.5	4.6	3.1	2.4	0.8	1.2
x_i	0	1	3	2	5	6	4	3	2



Using time-weighted statistics, we are not just interested in quantities such as the average of each of the sizes the population takes on, but in the more meaningful quantities such as the time-weighted average. That is, the average over the 24-unit time interval is computed by averaging the sizes the population assumes weighted by the time spent in each population size. Table 2.1.5-5 displays the values of each of the statistics using time-weighted sampling for the above example.

Table 2.1.5-5. Example of time-weighted sampling.

Statistic	Value
Total time	24.0
Sum	77.8
Sum of squares	328.2
Mean	3.242
Mean square	13.675
Variance	3.167
Standard deviation	1.779
Maximum	6.0
Minimum	0.0

Statistical package calling sequences.

The statistical sampling package XSTATS contains two entry points for sampling values of variables. The FORTRAN-calling sequences for sampling the value of a variable are

```
     STORAGE.  VAR
          :
          CALL SAMPLE(VAR)
     or   CALL SAMPLE(VAR,TIME).
```

The first argument in the call must be a real-valued variable and should be declared in STORAGE. The first form of the call is used for discrete sampling. Each time a new value for the sampled variable is computed, a call to sample should be made. The second form of call is used for time-weighted sampling. A call to sample should be made for each value the variable assumes when TIME is equal to the final end point of the interval over which the value holds. Calls to SAMPLE for a particular variable can not be mixed between the two forms of call. If an attempt is made to sample a variable in both discrete and time-weighted modes, a diagnostic is issued and subsequent calls referencing this variable are ignored. A maximum of 30 different variables can be sampled within a simulation. Upon attempting to sample more than 30 variables, a diagnostic is issued and attempts to sample in excess of the first 30 variables are ignored.

The sampling sequence can be reinitialized to begin anew at a point in the simulation by calls of the following form,

```
                CALL  RESET(VAR)
     or         CALL  RESET(VAR,TIME).
```

When RESET is called, each of the statistics for the named variable are reset to the appropriate initial values. In the discrete sampling case quantities such as n, Σx and Σx^2 are initialized to zero. For time-weighted sampling the initial time of sampling t_0 is set to the current value of time in addition to the other required initializations. If RESET is not called prior to any time-weighted calls to SAMPLE, the initial value assumed for t_0 is TSTRT, or the time of the first event if TSTRT is not given a value in an event-only simulation.

The statistics gathered by calls to sample will be printed by executing a call of the following form,

```
CALL  REPORT(VAR)
```

A report of all statistics will be printed in the output as illustrated in the example at the end of this section automatically. If the variable VAR is declared in STORAGE., the output will be labeled with the name of the variable. Otherwise the output is labeled by an integer enclosed in asterisks (*) which represents the actual core location of the variable.

The values of any of the statistics for any sampled variable can be assessed during the simulation by the FORTRAN function calls presented in Table 2.1.5-6. If the requested variable has not been sampled prior to the function call, an indefinite result is returned.

Table 2.1.5-6. Statistical function calls.

Sample Calling Sequence	Value Returned
V = COUNT(VAR)	n or $\Sigma\Delta t_i$
V = SUM(VAR)	sum
V = SUMSQ(VAR)	sum of squares
V = AVERAGE(VAR)	mean
V = RMEANSQ(VAR)	mean - square
V = VARIANC(VAR)	variance
V = STDEV(VAR)	standard deviation
V = RMAX(VAR)	largest value
V = RMIN(VAR)	smallest value

Example 2.1.5-7. Illustration of calls to the statistical package.

```
STORAGE. POP,BDEL,DDEL,PMEAN
NORMAL. PNORM
      EVENT BIRTH
C...EACH TIME THIS EVENT IS CALLED, A BIRTH OCCURES.  THE TIME BETWEEN
C     BIRTHS IS ASSUMED TO BE NORMALLY DISTRIBUTED WITH THE MEAN AND
C     STANDARD DEVIATION A FUNCTION OF TIME GIVEN BY AN INTERPOLATION
C     TABLE.
C
      IF(POP.LE.0.) RETURN
C...SAMPLE THE POPULATION SIZE.
      CALL SAMPLE(POP,TIME)
C...INCREMFNT THE POPULATION SIZE.
      POP=POP+1.
C...SCHFDULE THE TIME OF NEXT BIRTH.
      PMEAN=ALINT2(TIME,ICHK,0.,4.,45.,2.5,70.,1.,90.,0.57,120.,0.5,
     -     180.,0.73,210.,1.25,270.,3.3,365.,4.)
      PSTDV=0.1*PMEAN
    5 BDEL=PNORM(PMEAN,PSTDV)
C...THE NORMAL DISTRIBUTION IS TRUNCATED AT ZERO.
      IF(BDEL.LE.0.) GO TO 5
      CALL EVENT(5HBIRTH,TIME+BDEL,1)
C...SAMPLE THE TIME BETWEEN BIRTHS.
      CALL SAMPLE(BDEL)
      RETURN
      END
      EVENT DEATH
C...EACH TIME THIS EVENT IS CALLED A DEATH OCCURES.  THE TIME BETWEEN
C     DEATHS IS ASSUMED TO BE NORMALLY DISTRIBUTED WITH A MEAN OF 1.0 AND
C       STANDARD DEVIATION OF 0.1.
C
```

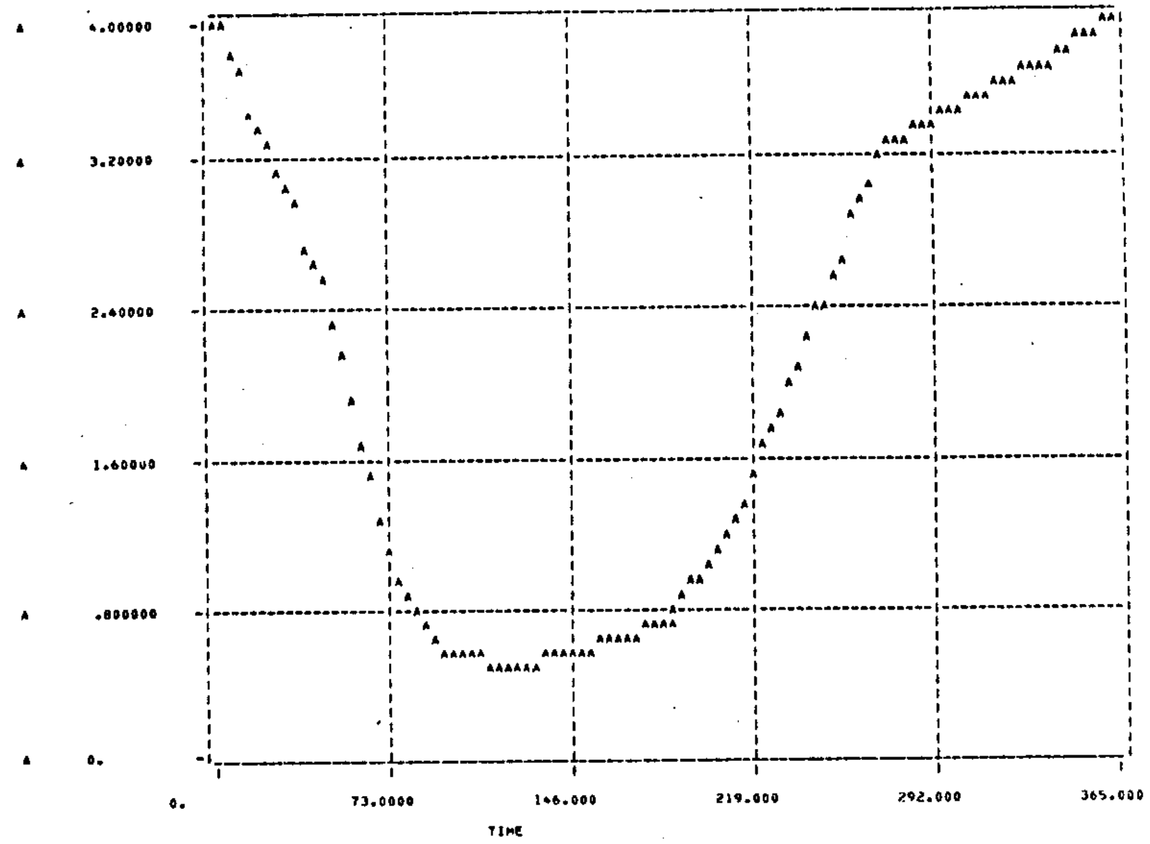
```
      IF(POP.LE.0.) RETURN
C...SAMPLE THE POPULATION SIZE.
      CALL SAMPLE(POP,TIME)
C...DECREMENT THE POPULATION SIZE.
      POP=POP-1.
C...SCHEDULE THE TIME OF NEXT DEATH.
    5 DDEL=PNORM(1.,0.1)
C...THE NORMAL DISTRIBUTION IS TRUNCATED AT ZERO.
      IF(DDEL.LE.0.) GO TO 5
      CALL EVENT(5HDEATH,TIME+DDEL,1)
C...SAMPLE THE TIME BETWEEN DEATHS.
      CALL SAMPLE(DDEL)
      RETURN
      END
      EVENT STOP
C...REPORT STATISTICS.
      CALL REPORT(POP)
      CALL REPORT(BDEL)
      CALL REPORT(DDEL)
      RETURN
      END
```

$^{7}8_{9}$ *end-of-record separator*

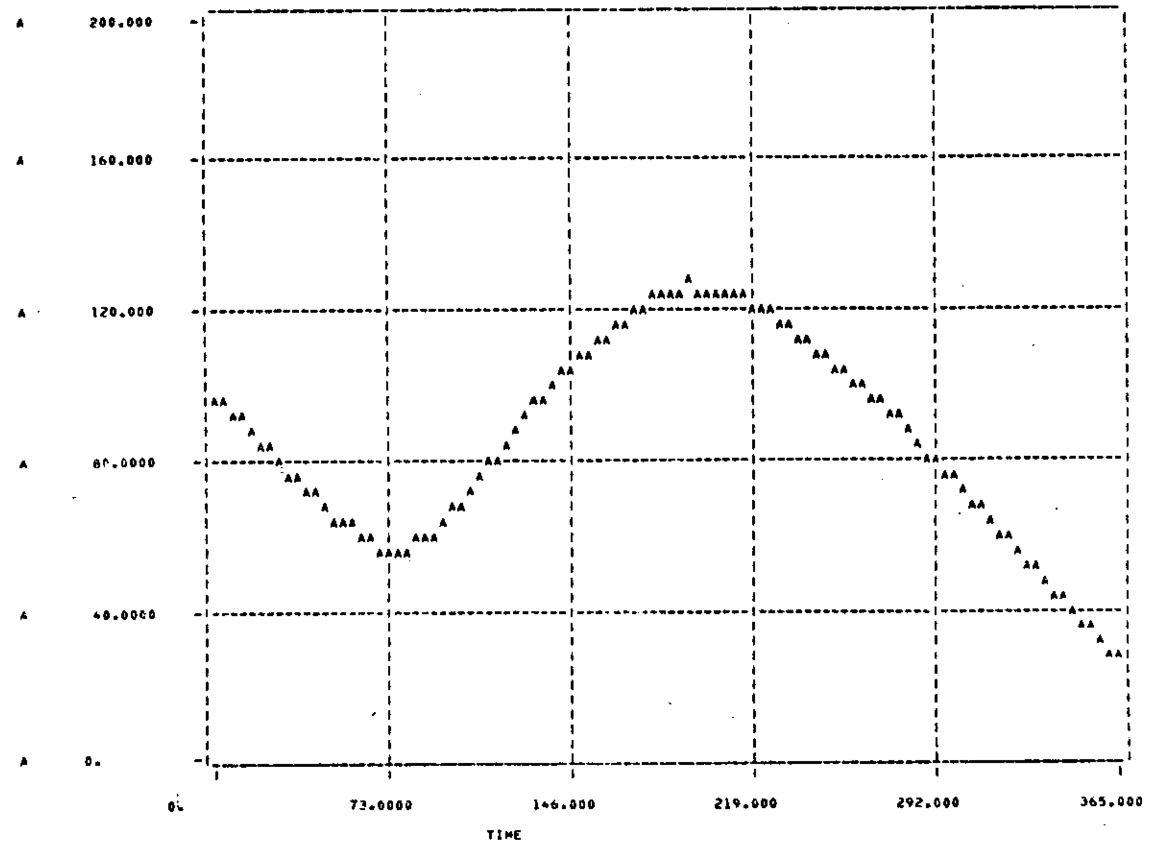
```
TSTRT=0. $ TEND=365. $ POP=100. $ PMEAN=4. $
EVENT. BIRTH,0.,1
EVENT. DEATH,0.,1
EVENT. STOP,365.,1
PLOT. (PMEAN)
PLOT. (POP)
```

```
STATISTICAL REPORT FOR POP                 TIME  =   365.000000

  TOTAL TIME     364.391079                AVERAGE     87.9510232                    ST. DEV.     27.1806966
     MAXIMUM     129.000000                MINIMUM     27.0000000                    VARIANCE     738.790267
         SUM     32048.6776                SUM SQ.     3087932.19                    MEAN SQ.     8474.22553


STATISTICAL REPORT FOR BDEL                TIME  =   365.000000

      NUMBER     296.000000                AVERAGE     1.24070477                    ST. DEV.     1.02023854
     MAXIMUM     4.55162573                MINIMUM     .385895681                    VARIANCE     1.04088668
         SUM     367.248612                SUM SQ.     763.749563                    MEAN SQ.     2.58023501


STATISTICAL REPORT FOR DDEL                TIME  =   365.000000

      NUMBER     369.000000                AVERAGE     .990517629                    ST. DEV.   9.937122086E-02
     MAXIMUM     1.28477325                MINIMUM     .693412332                    VARIANCE   9.874639535E-03
         SUM     365.501005                SUM SQ.     365.678931                    MEAN SQ.     .990999813
```

PLOT NO. 1



PLOT NO. 2



Punching data decks.

The special utility subroutine PUNCHD provides the capability of obtaining a punched deck of the system-declared simulation control variables (refer to Table 2.1-1, with the exeception of FLOW) and all variables declared in STORAGE.. The data deck is punched in a format consistent for input to a SIMCOMP simulation (refer to section 2.2.1). Any variables which have not been assigned a value are ignored. The call to PUNCHD contains no parameters and is of the following form.

```
CALL PUNCHD
```

The call to PUNCHD can be placed at any point in the simulation, but will be executed only the first time it is called. The call in most cases should be placed in routines that are executed only once, such as START or FINIS. A call to routine PUNCHD is most useful when data is generated in one simulation and is to be used as input in another simulation. The punched deck produced by PUNCHD will have the same job number as the job that generated the punched deck.

2.1.6 Listing controls.

SIMCOMP normally produces a listing of the source section during compilation. The SIMCOMP compiler directives LIST. and NOLIST. may appear at any place in the source section. If a NOLIST. directive is encountered, the printing of all source statements from that point on is suppressed. The printing of source statements is reinitiated by the LIST. directive. LIST. and NOLIST. can appear anywhere in columns 1 through 72 and blanks are ignored.

2.1.7. Execution controls.

At CSU using the SCOPE 3.3 operating system, the SIMCOMP compiler remains on the system as a permanent file and is called up and executed by means of the job control cards listed in Appendix C. SIMCOMP simulations can be executed in any one of three different modes, selected by the inclusion or absence of execution-control directives in the source section. The three modes of execution correspond to (i) the absence of any execution directives, (ii) the inclusion of a DEBUG. execution directive, and (iii) the inclusion of a NOGO. execution directive. Either DEBUG. or NOGO. is key-punched anywhere in columns 1 through 72 and blanks are ignored. If both a DEBUG. and a NOGO. card appear in the source section, then NOGO. is assumed.

The first mode of execution, that is, the default action of the compiler, should be used during the early stages of the development of a simulation. Any FORTRAN compilation errors which are detected will be printed in the output along with the offending statements. The second mode of execution, selected by a DEBUG. directive, should be used after all compilation errors have been eliminated. If compilation errors are encountered while DEBUG. has been selected, the only indication is a message entered in the dayfile; the run is terminated. The printed output will not contain any diagnostics explaining the nature of the compilation error. The use of DEBUG. is intended primarily for the detection and reporting of execution errors. If DEBUG. is selected and an arithmetic-mode error occurs during the course of the simulation, a short explanation of the error along with a dump of all variables and their values is provided. A complete explanation of the use of the DEBUG. facility is contained in section 2.3. The third mode of execution, selected by the NOGO. directive, suppresses compiler generation of the job control cards and therefore requires the user to supply the desired job control cards. When the default or DEBUG. mode for execution is selected, a standard set of job control cards are generated by the compiler which will automatically execute the simulation. By selecting NOGO. the user must supply his own control cards if anything more than a SIMCOMP compilation is desired. The job control card sequences generated in the default and DEBUG. modes are listed in Appendix B.

2.1.8 Comments

Comments conform to the format for FORTRAN comments and can appear anywhere in the source section. Any statement with the letter C in column 1 will be taken as a comment. The commentary information can be any string of blanks and characters in columns 2 through 80. Each comment statement must begin with a C in column 1 and may not be continued by means of a non-blank character in column 6 on subsequent cards.

2.2 Data Section

When a SIMCOMP source program has been compiled successfully, the first phase of execution of the simulation begins by reading and processing the data section. The data section is comprised of three types of statements. These are (i) data value assignment statements, (ii) output requests, and (iii) exogenous event requests. All statements in the data section are free form in columns 1 through 80 and blanks are ignored. Statements within the data section can appear in any order. Illegally formatted statements will produce a diagnostic, but in a great majority of cases the errors are not fatal. An attempt will be made to execute the simulation by assuming default values for critical parameters. As a result the output should be examined for data section diagnostics if correct results are to be realized. The physical location of the data section within the job deck is illustrated in Appendix B.

2.2.1 Parameter input data.

The values of variables or arrays declared in storage-allocation statements (refer to section 2.1.1) and the values of the simulation-control variables (refer to Table 2.1-1, with the exception of FLOW) may be set by data-value assignment statements. Data-value assignment is specified by statements of the following form:

$$\text{var} = v \ \$$$

or

$$\text{var} = v_1, v_2, \ldots , v_n \ \$$$

The variable name or array element "var" must have been declared in a storage-allocation statement or is a reserved simulation-control variable. The value of the variable or the values of the array "v" may be either integer- or real-valued constants. The mode of the value should correspond to the mode of the variable. If the mode of the value and the variable differ the mode of the value is converted to the mode of the variable. If an attempt is made to assign a real value to an integer variable, the value is truncated to an integer, the assignment is made, and a diagnostic is issued. Each expression is followed by a dollar sign. The expressions are free form in columns 1 through 80 and blanks are ignored. More than one expression may appear on a single data card, in addition to being run on from one card to the next. If any variable declared in a storage-allocation statement or state variable is not given a value in the data section, the variable is flagged as indefinite, and subsequent use of the variable prior to assigning the variable a value in the simulation will cause an arithmetic-mode error. Refer to section 2.3 on debugging for an explanation of the resulting diagnostic.

If the term "var" is an array element and a series of values are to be assigned to the array, the values are stored by columns in ascending order. This means that successive storage locations in a multiple-dimensioned array can be located by visualizing the left-most subscripts to vary the fastest. The array element "var" must be the location in the array where the storing of values begins. In an array declared as B(3,2,2) the order of the elements of the array is as follows:

```
B(1,1,1)
B(2,1,1)
B(3,1,1)
B(1,2,1)
B(2,2,1)
B(3,2,1)
B(1,1,2)
B(2,1,2)
B(3,1,2)
B(1,2,2)
B(2,2,2)
B(3,2,2)
```

For example, if we desired to set the last six locations of the above array to the values 1., 2., 3., 2., 4., and 6., respectively, we write:

```
B(1,1,2) = 1., 2., 3., 2., 4., 6. $
```

If an array name is not followed by a subscript, the first element is assumed. That is,

```
B = 0., 1.22E2, 1.22E-2 $
```

would result in

```
          B(1,1,1) = 0.
          B(2,1,1) = 1.22E2
and       B(3,1,1) = 1.22E-2.
```

If a series of locations are to contain the same value, an integer-repetition factor may be used. If the last six elements of the array B were to have the values 1., 1., 1., 3., 3., and 9., respectively, then we write:

```
B(1,1,2) = 3*1., 2*3., 9. $
```

The entire array could be set to zero with the single statement:

```
B = 12*0. $
```

If a particular variable or array element is set more than once, the last assignment is assumed in effect. If the entire array above is to be set to zero with the exception of element B(2,2,1) which has the value 9.3×10^{-3}, we write:

```
B = 12*0. $ B(2,2,1) = 9.3E-3 $
```

Example 2.2.1-1. Illustration of data-value assignment.

```
STORAGE. S(3,3),S2(4),INDX
STORAGE. VA,VB,VC,TOP,PX,KVAL(2,3)
REAL. I(10),J(10)
INTEGER. T1,T2
```

$^{7}8_{9}$ *end-of-record separator*

```
S=9*1. $ VB=2. $ VC=3.14 $ VA=3.69385E-6 $ TOP=10.3 $ S(1,1)=0.2 $ S(2,2)=0.3 $
S(3,3)=0.5 $ INDX=1 $ KVAL=1,90,3*0,500 $ PX=9.9E+10 $ I=3*0,1 $ I(5)=5*2 $ J=1,
3,5,3*6,4*9 $ T1=20 $ T2=10 $ PX=999.999 $ S2(2)=39. $ S(4)=41. $ TSTRT=0. $ TEN
D=100. $ DT=0.01 $ DTPR=10. $ DTFL=20. $ X(1)=0. $ X(23)=3*50. $
```

Simulation control variables.

When a simulation has been defined containing flows, the reserved system control variables TSTRT, TEND, and DT should be given values in the data section. These variables must be given values in order for the simulation to execute; if the user fails to set the value of any or all of these three variables, default values will be assumed. Whenever the system assumes a default value for a control variable not specified by the user, a warning message is issued. Table 2.2.1-1 describes the default values assumed by the system. When the user supplies values for TSTRT, TEND, and DT the following conditions must hold:

$$\text{TSTRT} \leq \text{TEND}$$

$$\text{DT} > 0$$

If either or both of these conditions are not satisfied, a diagnostic is printed and the values in the last line of Table 2.2.1-1 are assumed.

Table 2.2.1-1. Default values of simulation control variables for simulations containing flows.

TSTRT	TEND	DT
given	given	(TEND-TSTRT)/10.
given	TSTRT + 10.*DT	given
TEND - 10.*DT	given	given
given	TSTRT + 10.	1.
TEND - 10.	given	1.
0.	10.*DT	given
0.	10.	1.

If the simulation does not contain any flows, the simulation is assumed to contain only events. In this case the reserved simulation control variable DT has no meaning and does not have to be given a value. In order for an event-only simulation to execute, the chain of events has to be initiated in either of two ways. If TSTRT is given a value in the data section and subroutine START has been supplied by the user, then execution time calls to the event schedule (refer to section 2.1.3) can be included in subroutine START to initiate the event sequence. If TSTRT is not given a value, then the only way an event sequence can be initiated is by including exogenous-event requests (refer to sections 2.1.3 and 2.2.3) in the data section. In this case if TSTRT has not been given a value, TSTRT is assumed to be equal to the time of the first exogenously scheduled event. If TEND is not given a value, no action is taken and the user has the responsibility of scheduling the system-defined event HALT at the appropriate time. If TEND is given a value, the simulation will be terminated automatically at time TEND. If TEND is less than or equal to the value of TSTRT, the value of TEND is assumed indefinite.

The reserved simulation control variables DTPR, DTPL, and DTFL are used by the SIMCOMP executive routine to determine the frequency of tabular print-outs, storage of values for plotting, and the printing of values of flows respectively. If one of the above output actions is requested (refer to section 2.2.2) and the corresponding simulation control variable has not been given a value in the data section or was given an illegal value, the system will issue a diagnostic and a default value will be chosen. The following default values (Table 2.2.1-2) are those chosen if at least one flow has been defined in the simulation.

Table 2.2.1-2. Default values of output control variables for continuous simulations (containing flow definitions).

Condition		Default Action
PRINT. request(s) present and		
	(1) DTPR $\leq$ 0. or indefinite.	DTPR = maximum of (TEND-TSTRT)/10 and DT.
	(2) DTPR < DT.	DTPR = DT.
PLOT. request(s) present and		
	(1) DTPL $\leq$ 0. or indefinite; and TIME is the independent variable of a least one plot.	DTPL is set to a value which provides good readability while minimizing execution time.
	(2) DTPL $\leq$ 0. or indefinite; and TIME is not the independent variable of any plot.	DTPL = DT.
	(3) DTPL < DT.	DTPL = DT.
FLOW. request(s) present and		
	(1) DTFL $\leq$ 0. or indefinite.	DTFL = maximum of (TEND-TSTRT)/10 and DT.
	(2) DTFL < DT.	DTFL = DT.

If no flows have been defined in the simulation and only events have been defined, the following table (Table 2.2.1-3) describes the default values chosen for the output simulation control variables.

Table 2.2.1-3. Default values of output control variables for simulations containing only events.

Condition	Default Action
PRINT. request(s) present and	
(1) DTPR ≤ 0 or indefinite, and TEND is defined.	DTPR = (TEND-TSTRT)/10.
(2) DTPR ≤ 0 or indefinite, and TEND is undefined.	DTPR = 1.
PLOT. request(s) present and	
(1) DTPL ≤ 0 or indefinite; TEND is defined; and TIME is the independent variable of a least one plot.	DTPL is se to a value which provides good readability while minimizing execution time.
(2) DTPL ≤ 0 or indefinite; TEND is defined; and TIME is not the indefinite variable of any plot.	DTPL = 1.
(3) TEND is undefined.	DTPL = 1.

2.2.2 Output requests.

The results of a simulation can be requested as printed tables of values through time in addition to printed or microfilm plots. Output requests also allow a measure of control over the printing of the initial values of variables declared in STORAGE.. An execution trace facility is also provided which is especially useful during the debugging stages of simulations containing events. The format and usage of these commands are described in the following pages. In general, an output request is comprised of a request verb followed by a period with the remainder of the card containing the necessary information. The output requests are free form in columns 1 through 80 with blanks ignored. All output requests are contained in the data section.

PRINT. requests.

Printed tabular output requests are specified by the statement form:

$$\text{PRINT. } v_1, v_2, \ldots , v_m$$

Each of the variables v_i to be printed must be a variable or array location which appears in a storage allocation statement (i.e., STORAGE., INTEGER., or REAL. statement) in the source section. The values of the state variables may also be requested for printing by specifying "X(i)" where i is the compartment number.

If a card with the word PRINT. without a list of variables is included, then all state variables will be printed. As many PRINT. cards as needed may be included.

The variable which controls the frequency of tabular output through simulated time is DTPR. The value for DTPR should be set by means of a data-assignment expression. If DTPR is not given a value, a diagnostic will be issued and a default value assumed. Tables 2.2.1-2 and 2.2.1-3 describe the values chosen.

The following example illustrates some legal PRINT. requests. Presumably the state variables requested in the second request would have been defined in the simulation. An example of the output produced by PRINT. requests is presented in example 2.2.2-1.

Example 2.2.2-1. Illustration of PRINT. requests.

```
STORAGE. RSET,QVAL(3),TQ(4,3,2)
REAL. NPOP,IST(2,3)
INTEGER. DAY,MONTH,YEAR
789      end-of-record separator
PRINT. DAY,MONTH,YEAR,RSET,IST(1,1),IST(2,1),QVAL(1),QVAL(2),QVAL(3),TQ(1,1,1)
PRINT. NPOP,X(3),X(4),X(66),X(976)
PRINT.
```

FLOW. requests.

The printing of computed values for the flows over each integration step is specified by the statement form:

FLOW. (i,j), (k,l), ... , (m,n)

Each parenthesized pair of numbers refers to a flow defined in the source section. If a card with the word FLOW. is included without a list of particular flows, then all flows defined in the simulation will be printed. If a requested flow (i,j) has not been defined, a diagnostic is issued and the illegal request is ignored.

The variable which controls the frequency with which flows are printed is DTFL. The value for DTFL should be set by means of a data assignment expression. If DTFL is not given a value, a diagnostic will be issued and a default value assumed. Tables 2.2.1-2 and 2.2.1-3 describe the values chosen. When evaluating the performance of a simulation, care must be taken to associate the values of the flows with the correct time step. The following simple example illustrates the relationship between the times at which the state variables are printed via a PRINT. request and the times at which the flows are printed via a FLOW. request.

Example 2.2.2-2. Illustration of FLOW. requests.

```
(100-200). FLOW=0.01*X(200)
(I=1,3-500). FI=I*2
      FLOW=0.1*X(I)/FI
789     end-of-record separator
X(1)=3*100. $ X(100)=1000. $ X(200)=1. $ X(500)=0. $
TSTRT=0. $ TEND=10. $ DT=1. $ DTFL=1. $ DTPR=1. $
PRINT.
FLOW.
```

```
SIMULATION RESULTS
                              (partial listing)

TIME = 0.
         X(100) =  1000.00000            X(200) =  1.00000000          X(1) =  100.000000            X(500) =           0
         X(2) =  100.080000              X(3) =  100.000000

VALUES OF FLOWS. TIME = 0.                  TO  1.00000000
         FLOW(100,200) =  .10000E-01  FLOW(  1,500) =  5.00000000  FLOW(  2,500) =  2.50000000  FLOW(  3,500) =  1.66666667

TIME =  1.00000000
         X(100) =  999.990000            X(200) =  1.01000000          X(1) =  95.0000000            X(500) =  9.16666667
         X(2) =  97.5000000              X(3) =  98.3333333

VALUES OF FLOWS. TIME =  1.00000000      TO  2.00000000
         FLOW(100,200) =  .10100E-01  FLOW(  1,500) =  4.75000000  FLOW(  2,500) =  2.43750000  FLOW(  3,500) =  1.63888889

TIME =  2.00000000
         X(100) =  999.979900            X(200) =  1.02010000          X(1) =  90.2500000            X(500) =  17.9930556
         X(2) =  95.0625000              X(3) =  96.6944444

VALUES OF FLOWS. TIME =  2.00000000      TO  3.00000000
         FLOW(100,200) =  .10201E-01  FLOW(  1,500) =  4.51250000  FLOW(  2,500) =  2.37656250  FLOW(  3,500) =  1.61157407

TIME =  3.00000000
         X(100) =  999.969699            X(200) =  1.03030100          X(1) =  85.7375000            X(500) =  26.4936921
         X(2) =  92.6859375              X(3) =  95.0828704

VALUES OF FLOWS. TIME =  3.00000000      TO  4.00000000
         FLOW(100,200) =  .10303E-01  FLOW(  1,500) =  4.28687500  FLOW(  2,500) =  2.31714844  FLOW(  3,500) =  1.58471451

TIME =  4.00000000
         X(100) =  999.959396            X(200) =  1.04060401          X(1) =  81.4506250            X(500) =  34.6824301
         X(2) =  90.3687891              X(3) =  93.4981559

VALUES OF FLOWS. TIME =  4.00000000      TO  5.00000000
         FLOW(100,200) =  .10406E-01  FLOW(  1,500) =  4.07253125  FLOW(  2,500) =  2.25921973  FLOW(  3,500) =  1.55830260

TIME =  5.00000000
         X(100) =  999.948990            X(200) =  1.05101005          X(1) =  77.3780937            X(500) =  42.5724836
         X(2) =  88.1095693              X(3) =  91.9398533

VALUES OF FLOWS. TIME =  5.00000000      TO  6.00000000
         FLOW(100,200) =  .10510E-01  FLOW(  1,500) =  3.86890469  FLOW(  2,500) =  2.20273923  FLOW(  3,500) =  1.53233089

TIME =  6.00000000
         X(100) =  999.938480            X(200) =  1.06152015          X(1) =  73.5091891            X(500) =  50.1764585
         X(2) =  85.9068301              X(3) =  90.4075224

VALUES OF FLOWS. TIME =  6.00000000      TO  7.00000000
         FLOW(100,200) =  .10615E-01  FLOW(  1,500) =  3.67545945  FLOW(  2,500) =  2.14767075  FLOW(  3,500) =  1.50679204

TIME =  7.00000000
         X(100) =  999.927865            X(200) =  1.07213535          X(1) =  69.8337296            X(500) =  57.5063807
         X(2) =  83.7591593              X(3) =  88.9007303

VALUES OF FLOWS. TIME =  7.00000000      TO  8.00000000
         FLOW(100,200) =  .10721E-01  FLOW(  1,500) =  3.49168648  FLOW(  2,500) =  2.09397898  FLOW(  3,500) =  1.48167884

TIME =  8.00000000
         X(100) =  999.917143            X(200) =  1.08285671          X(1) =  66.3420431            X(500) =  64.5737250
         X(2) =  81.6651804              X(3) =  87.4190515

VALUES OF FLOWS. TIME =  8.00000000      TO  9.00000000
         FLOW(100,200) =  .10828E-01  FLOW(  1,500) =  3.31710216  FLOW(  2,500) =  2.04162951  FLOW(  3,500) =  1.45698419
```

PLOT. requests.

Plotted output requests are specified by the following statement forms:

PLOT. (group$_1$), ... , (group$_n$)

or

PLOT. (group$_1$), ... , (group$_n$) phrase

Each plot card will generate a single plot (on the line printer unless a FILM. card is included--refer to page 2.2.2-12) of the variables listed in the groups on the plot card. Each "(group$_i$)" is an expression of the following form:

$$(u_1, \dots , u_m)$$

or

$$(u_1, \dots , u_m \text{ [min,max]})$$[5]

Each of the terms "u$_i$" is an expression specifying a dependent variable and takes on one of the following forms:

var

or

var=c

or

var.LOG

or

var.LOG=c

Each term "var" is a dependent variable to be plotted and must be a simple variable or location in an array which was declared in a storage-allocation statement in the source section, or is a state variable of the form "X(i)" where i is a compartment number. The logorithm to the base 10 of a variable is plotted by specifying ".LOG" immediately following the variable

---

[5] The characters "[" and "]" are represented on a key punch by the multipunches 8.7 card 0-8-2, respectively.

name and its subscripts if present. In this case the variable is plotted on a log scale, but the true values of the variable are printed on the dependent axis. If one variable in a group is requested to be plotted on a log scale, all variables in the group will be so plotted. The character used in the plot to identify the particular variable is normally chosen by the plotting routine. An index of the variables plotted and the characters used is printed out. A specific character for a variable can be selected by appending the expression "=c," where "c" is the character to be plotted.

Each group of variables (group_i) is scaled on the plot independently of the other groups. One through five groups per plot card with one through five variables per group are allowed. If the expression in brackets "[min,max]" is included in a group, the minimum and maximum specified in the brackets are used as the extremes of the dependent variable(s) in the group. The terms "min" and "max" must be integer- or real-valued constants. If the extreme values for a group are not specified, the minimum and maximum values for all variables in the group, appropriately rounded for readability, are used as the extreme values of the group.

The optional modifying expression "phrase" is of the following forms:

/ var

or

/ var [min,max]

or

[min,max]

The term "/ var" specifies the independent variable for the plot. Only one independent variable is allowed per plot. If the term "/ var" is omitted, "/ TIME" is assumed. The term "var" can be any variable declared in a storage-allocation statement or a state variable of the form X(i). The bracketed expression "[min,max]" specifies the extreme values to be used in the plot for the independent variable. If the bracketed expression is omitted, the minimum and maximum values for the independent variable are used. The quantities "min" and "max" must be integer- or real-values constants.

When PLOT. cards have been included in the data section, the values of the variables named on PLOT. cards are saved on a mass storage device during the simulation at intervals of DTPL. These values are later retrieved and used to produce the plots. As long as TIME is the independent variable for at least one plot, the value of DTPL does not have to be set by the user via a data-assignment statement. A value for DTPL which reduces the amount of data to be saved while preserving the ultimate readability of the plot is chosen automatically. Refer to Tables 2.2.1-2 and 2.2.1-3 for a complete description of the choices of values for DTPL.

PLOT. requests are free form in columns 1 through 80 and blanks are ignored. Each plot request must be completed on one data card. The following examples illustrate some of the possibilities of formatting plot requests. All of the printer plots were generated in the same run and are reproduced following the examples of the plot requests.

Example 2.2.2-3. Illustration of PLOT. requests with printer-plotted output.

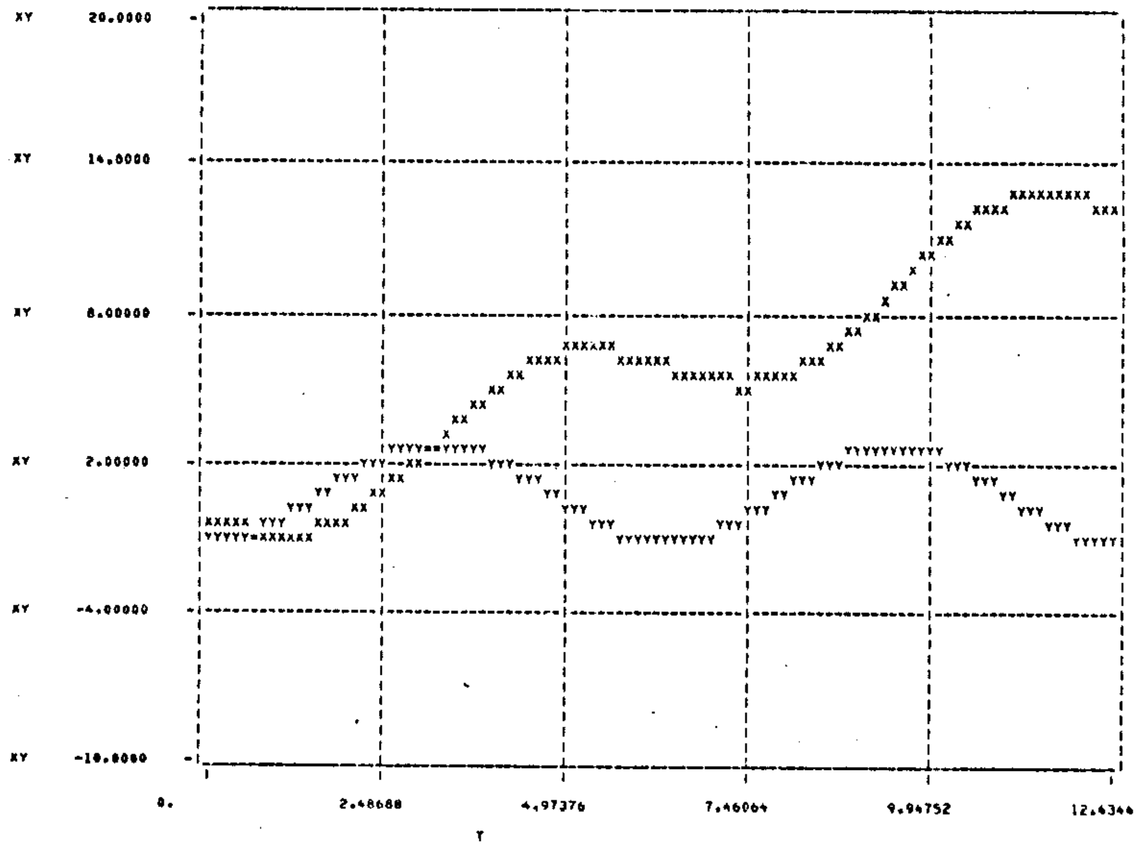
```
STORAGE. T,A,A1,RX,RY,TSIN
STORAGE. U,V
(1-2).
      T=TIME*6.28/50.
      RX=A*T-A1*SIN(T)
      RY=A-A1*COS(T)
      TSIN=SIN(T)
      FLOW=?.*TSIN
      V=TIME/10.
      U=EXP(V*SIN(V))
```

$^{7}8_{9}$ *end-of-record separator*

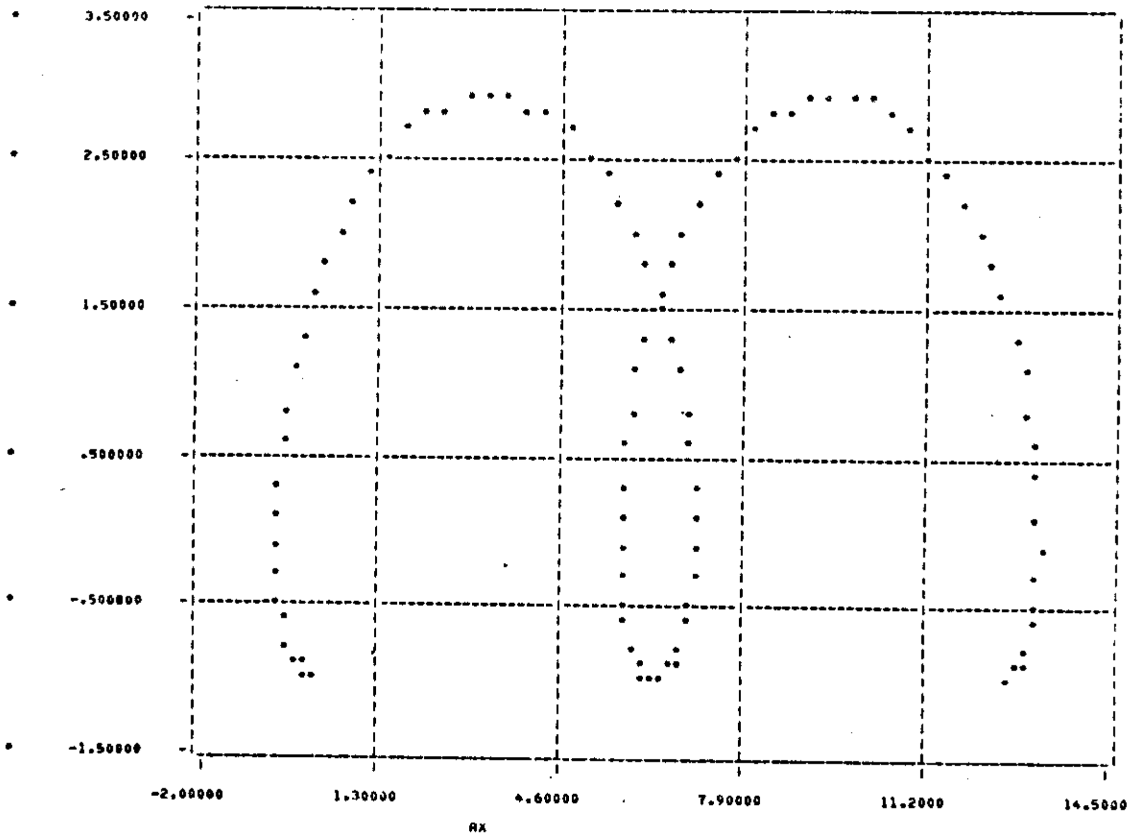
```
TSTRT=0. $ TEND=100. $ DT=1. $ X=2*0. $
T=0. $ A=1. $ A1=2. $ RX=0. $ RY=-1. $ TSIN=0. $
V=0. $ U=1. $
PLOT. (RX=X,RY=Y)/T
PLOT. (RY=*[-1.5,3.5])/RX[-2,14.5]
PLOT. (TSIN),(X(1))
PLOT. (TSIN=+[-2,2])/X(1)[-40,+40]
PLOT. (U)/V
PLOT. (U.LOG)/V
```

```
     GRAPHICAL SIMULATION RESULTS                       07/23/73                    10.48.56.

                                                                                            INDEPENDENT VARIABLE
GRAPH   GROUP   GROUP RANGE DECLARATION   DEPENDENT VARIABLE(S)   PLOTTED     INDEPENDENT VARIABLE   RANGE DECLARATION
NO.                                                               CHARACTER
----------------------------------------------------------------------------------------------------------------------

                                          RX                      X           T
  1       1                               RY                      Y

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

  2       1     -1.50      TO   3.50      RY                      *           RX                     -2.00    TO   14.5

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

  3       1                               TSIN                    A           TIME
          2                               X(1)                    B

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

  4       1     -2.00      TO   2.00      TSIN                    +           X(1)                   -40.0    TO   40.0

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

  5       1                               U                       A           V

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

  6       1                               U               LOG     A           V
```

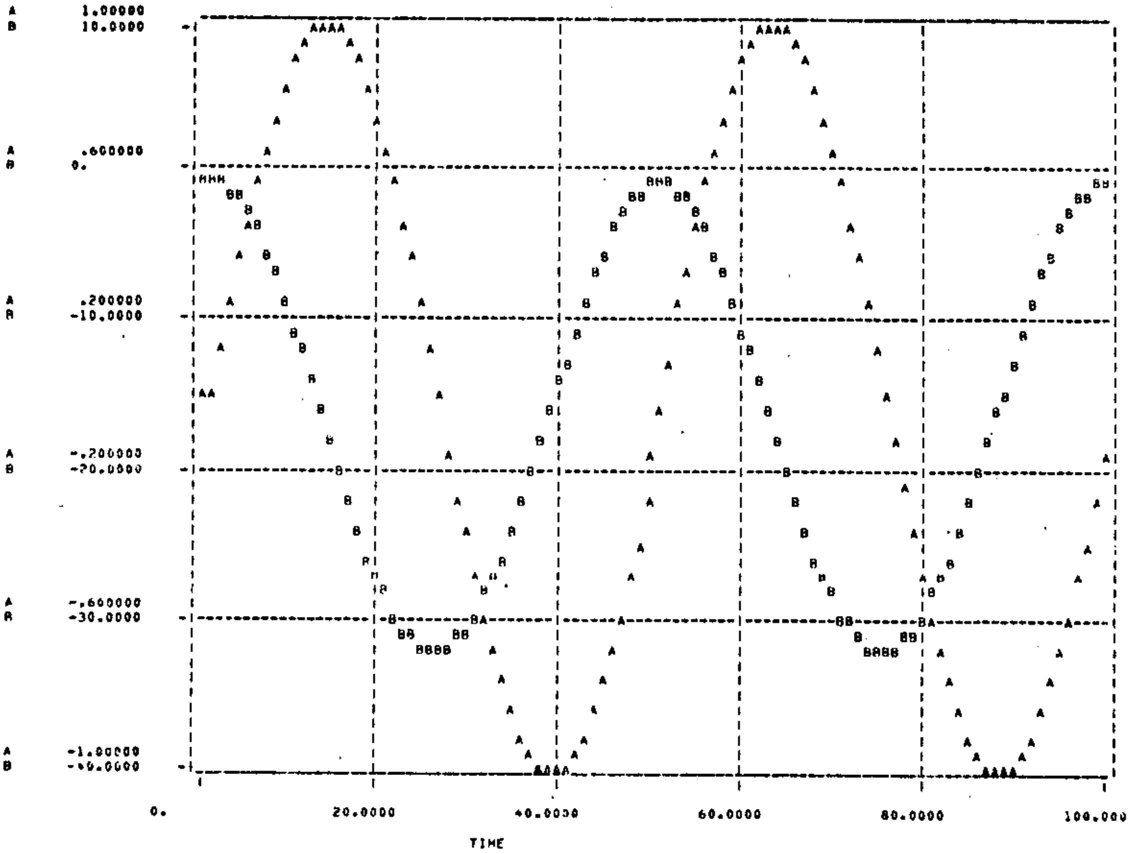
PLOT NO. 1



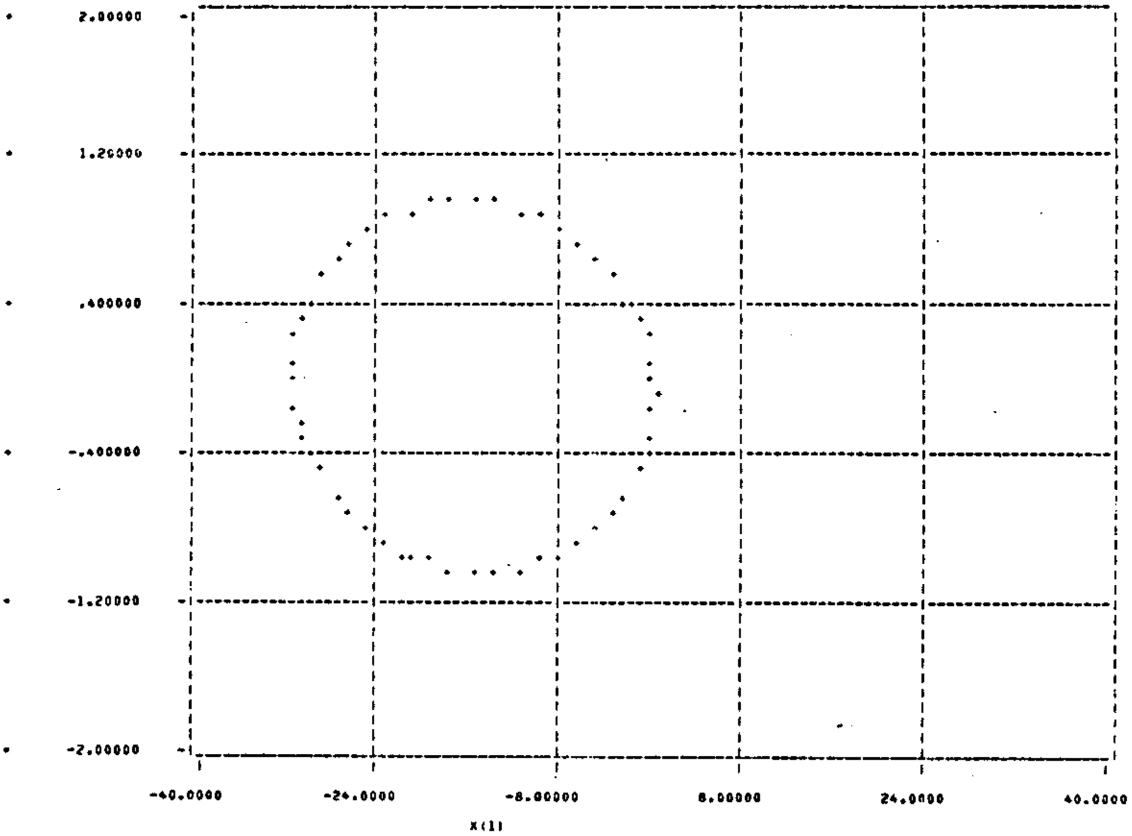
PLOT NO. 2



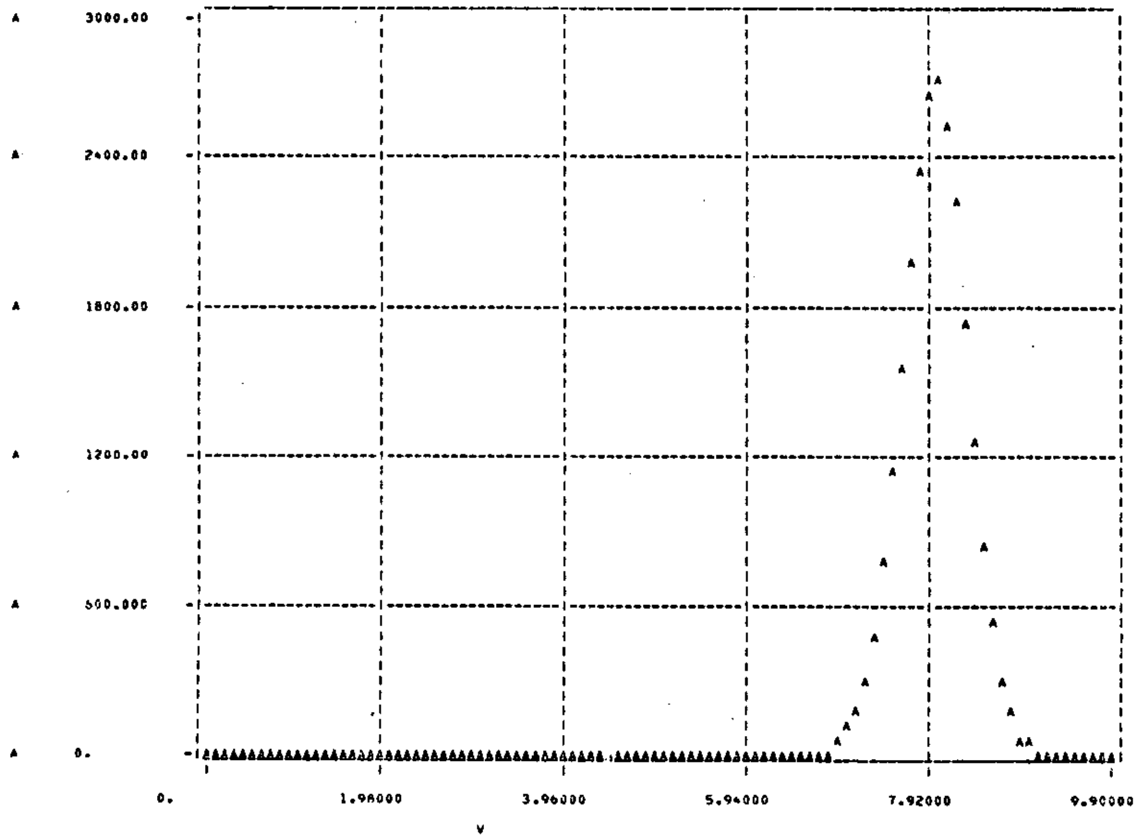
PLOT NO. 3



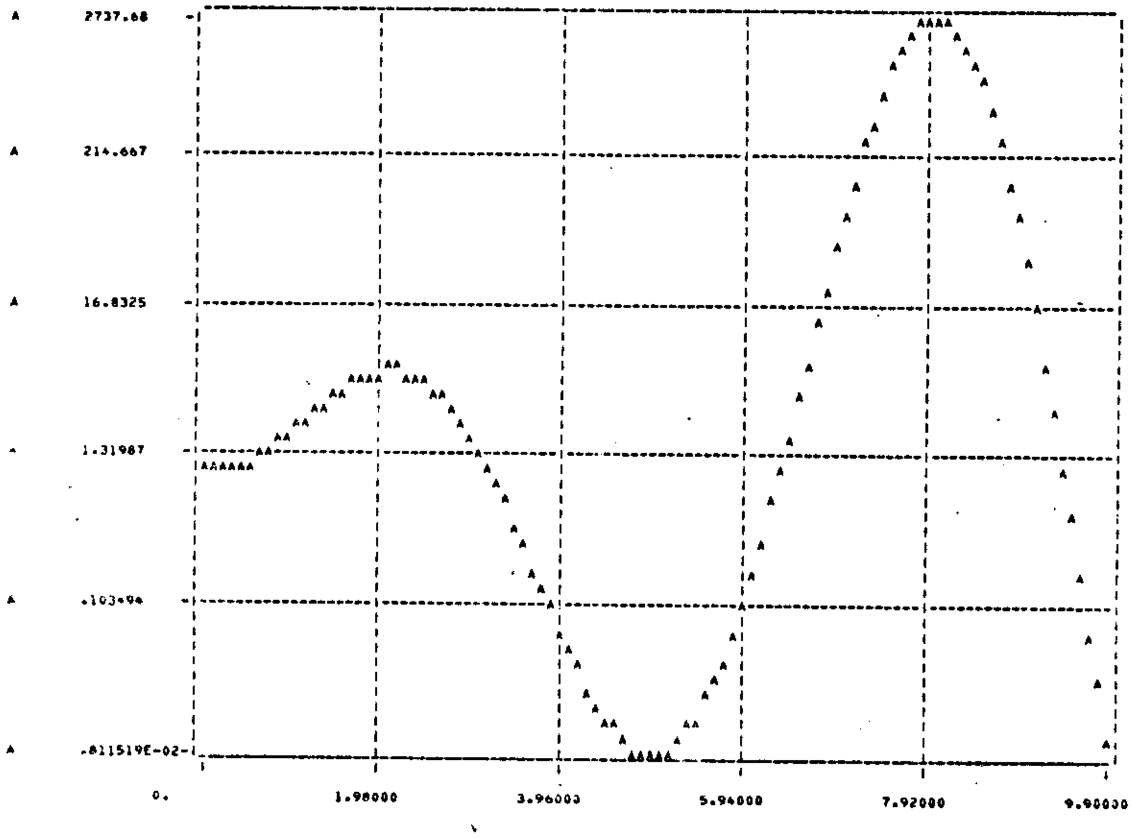
PLOT NO. 4



PLOT NO. 5



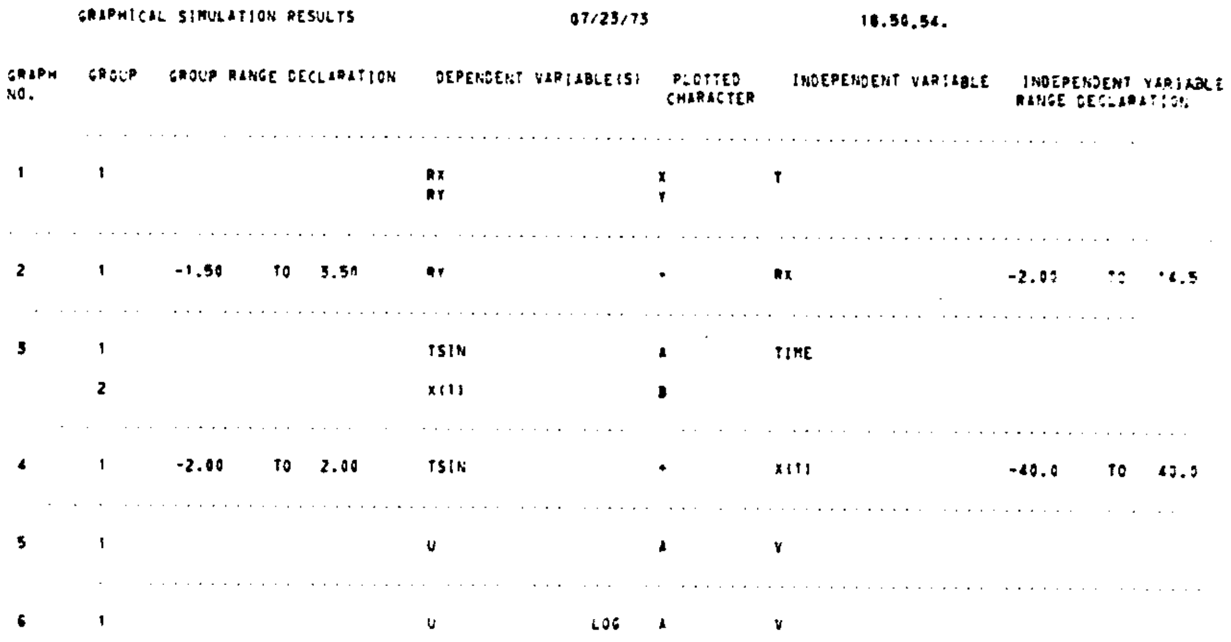
PLOT NO. 6



<u>FILM</u>. <u>requests</u>.

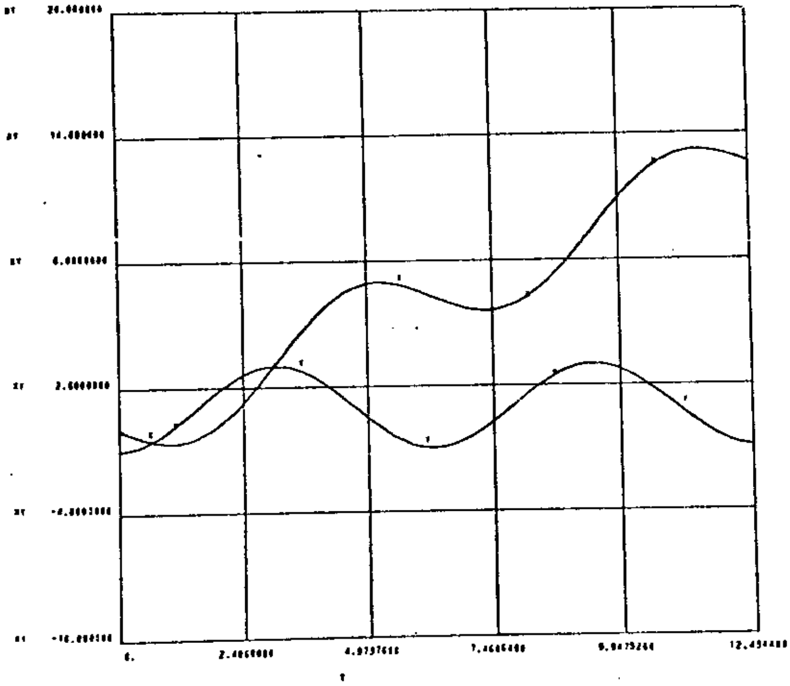
PLOT. requests will cause the graphs to be produced on the line printer and will accompany the output. A FILM. card included in the data section, all plots requested will be generated on microfilm. Plots generated on microfilm will generally have a higher degree of resolution than those produced by the line printer. The FILM. card is free form in columns 1 through 80 and blanks are ignored. The plots which were illustrated in the preceding section are reproduced in the following from microfilm.

Example 2.2.2-4. A sample of microfilm output.

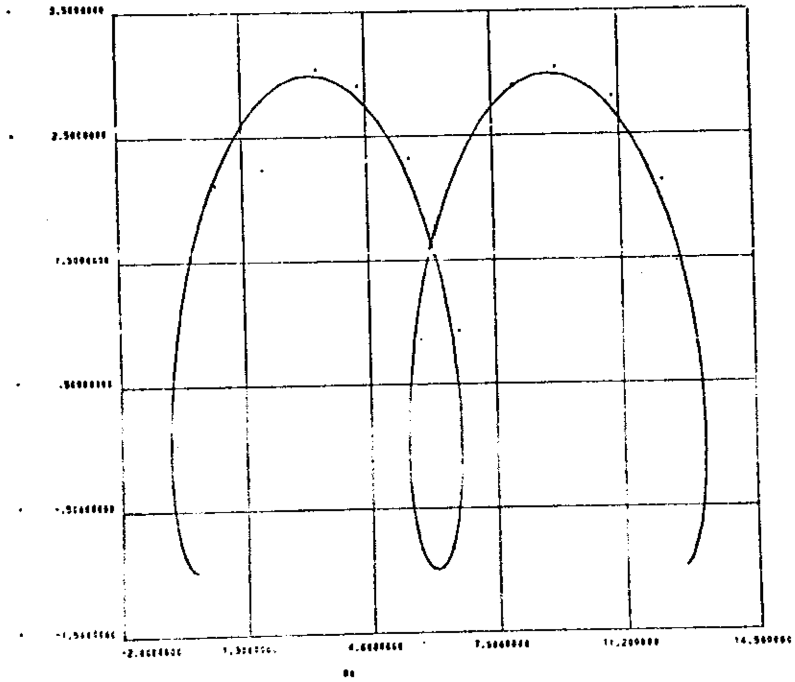


```
      GRAPHICAL SIMULATION RESULTS               07/23/73            10.50.54.

GRAPH  GROUP  GROUP RANGE DECLARATION   DEPENDENT VARIABLE(S)  PLOTTED    INDEPENDENT VARIABLE   INDEPENDENT VARIABLE
NO.                                                            CHARACTER                         RANGE DECLARATION
      ...........................................................................................................
  1      1                              RX                       X        T
                                        RY                       Y
 .................................................................................................................
  2      1     -1.50     TO   3.50      RY                       *        RX                    -2.00     TO   14.5
   ..............................................................................................................
  3      1                              TSIN                     A        TIME
         2                              X(1)                     B
    .............................................................................................................
  4      1     -2.00     TO   2.00      TSIN                     *        X(1)                  -40.0     TO   43.0
    .............................................................................................................
  5      1                              U                        A        V
         .........................................................................................................
  6      1                              U                  LOG   A        V
```

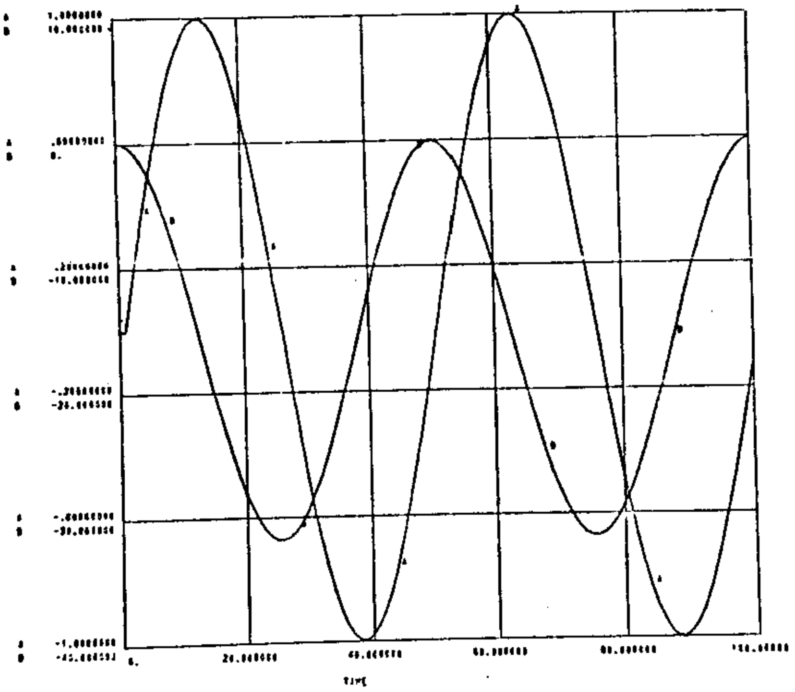
PLOT NO. 1



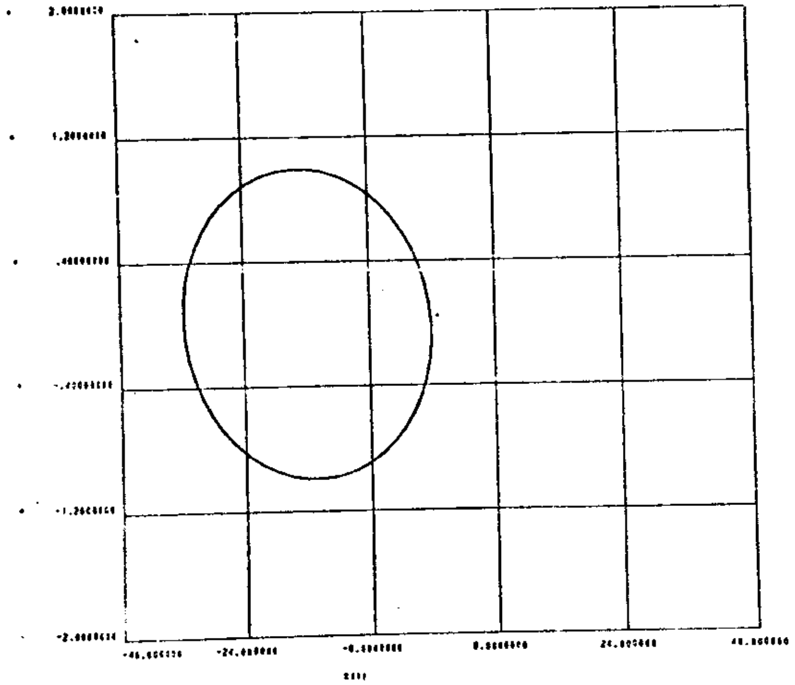
PLOT NO. 2



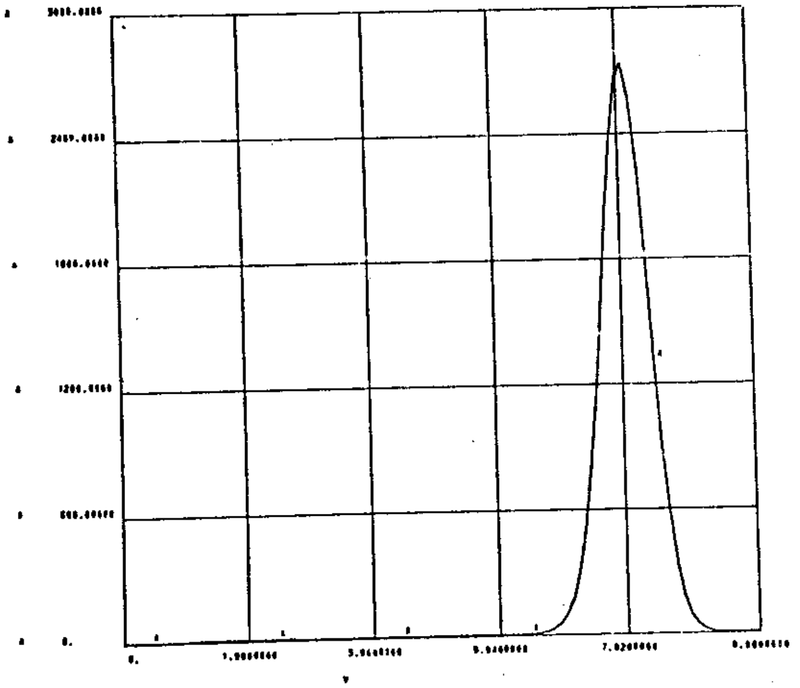
PLOT NO. 3



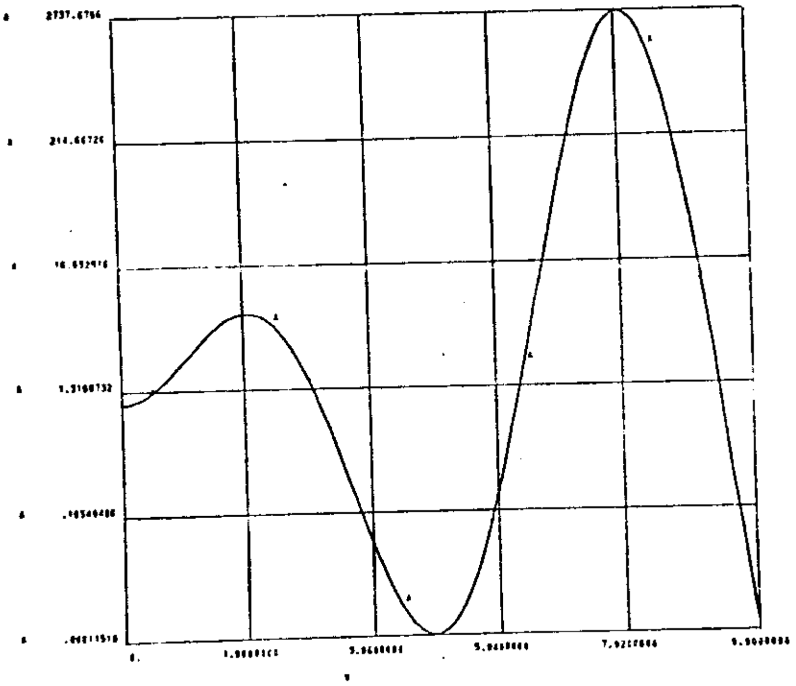
PLOT NO. 4



PLOT NO. 5



PLOT NO. 6



TITLE. requests.

A one-line title for each plot can be requested by a statement of the following form:

TITLE. text

A TITLE. card is free format in columns 1 through 80. The title for each plot is indicated by placing a TITLE. card before the corresponding PLOT. card. If a TITLE. card appears after all PLOT. requests or if more than one TITLE. card precedes a PLOT. request, a diagnostic is issued and the offending TITLE. card or cards will be ignored. The TITLE. card immediately preceding a PLOT. card will be used as the title of the plot. Any nonblank characters following the period on a title card is assumed part of the title and will be reproduced at the top of the corresponding plot. The use of TITLE. requests is illustrated in example 2.1.3-1.

Initial-conditions listing controls.

After the data section has been processed but prior to the start of execution of the simulation, the initial values of the simulation control variables, state variables, and primary user-declared variables are printed. Primary user-declared variables are those variables declared in storage-allocation statements which are not prefaced by an asterisk (refer to section 2.1.1). The initial values of secondary user-declared variables (i.e., those variables prefaced by an asterisk in storage-allocation statements), in addition to the initial values normally printed, are requested by the inclusion of an ALL. card in the data section. The characters ALL. are free form in columns 1 through 80, and the period must be included. Similarly, the printing of *all* initial conditions is suppressed by the inclusion of a NONE. card. The characters NONE. are free form in columns 1 through 80, and the trailing period must be included.

Both of the examples presented are followed by the initial-conditions output which was requested.

Example 2.2.2-5. An example of normal initial-conditions output.

```
STORAGE. S(3,3),S2(4),INDX,*LARGE(25)
STORAGE. VA,VB,VC,TOP,PX,KVAL(2,3)
REAL. I(10),J(10)
INTEGER. T1,T2,*TEMP1,*TEMP2
```

$^{7}8_{9}$ *end-of-record separator*

```
S=9*1. $ VB=2. $ VC=3.14 $ VA=3.69385E-6 $ TOP=10.3 $ S(1,1)=0.2 $ S(2,2)=0.3 $
S(3,3)=0.5 $ INDX=1 $ KVAL=1,90,3*0,500 $ PX=9.9E+10 $ I=3*0,1 $ I(5)=5*2 $ J=1,
3,5,3*6,4*9 $ T1=20 $ T2=10 $ PX=999.999 $ S2(2)=39. $ S(4)=41. $ TSTRT=0. $ TEN
D=100. $ DT=0.01 $ DTPR=10. $ DTFL=20. $ X(1)=0. $ X(23)=3*50. $
LARGE=2,85,63,45,72,15,38,92,100,12,3,4,50,5*10 $ LARGE(21)=5*1 $
```

```
SIMCOMP VERSION 3.0          PARAMETER VALUES

                                  - SIMULATION CONTROL PARAMETERS -

                                     TSTRT =            0
                                     TEND  =   100.000000
                                     DT    =   .100000000E-01
                                     DTPR  =   10.0000000
                                     DTFL  =   20.0000000

                                 - PRIMARY USER DEFINED VARIABLES -

      I(1-3) =             0       I(4) =  1.00000000           I(5-9) =  2.00000000           I(10) = INDEFINITE
      INDX =             1         J(1) =  1.00000000           J(2) =  3.00000000             J(3) =  5.00000000
      J(4-6) =  6.00000000         J(7-10) =  9.00000000        KVAL(1,1) =              1     KVAL(2,1) =             90
      KVAL(1,2-1,3) =            0, KVAL(2,3) =            500  PX =  999.999000               S(1,1) =  .200000000
      S(2,1-3,1) =  1.00000000     S(1,2) =  41.0000000         S(2,2) =  .300000000           S(3,2-2,3) =  1.00000000
      S(3,3) =  .500000000         S2(1) = INDEFINITE           S2(2) =  39.0000000            S2(3-4) = INDEFINITE
      TOP =  10.3000000            T1 =            20           T2 =            10             VA =  .369385000E-05
      VR =  2.00000000             VC =  3.14000000
```

Example 2.2.2-6. An example of initial-conditions output using ALL.

```
STORAGE. S(3,3),S2(4),INDX,*LARGE(25)
STORAGE. VA,VB,VC,TOP,PX,KVAL(2,3)
REAL. I(10),J(10)
INTEGER. T1,T2,*TEMP1,*TEMP2
```

$^{7}8_{9}$ *end-of-record separator*

```
S=9*1. $ VB=2. $ VC=3.14 $ VA=3.69385E-6 $ TOP=10.3 $ S(1,1)=0.2 $ S(2,2)=0.3 $
S(3,3)=0.5 $ INDX=1 $ KVAL=1,90,3*0,500 $ PX=9.9E+10 $ I=3*0,1 $ I(5)=5*2 $ J=1,
3,5,3*6,4*9 $ T1=20 $ T2=10 $ PX=999.999 $ S2(2)=39. $ S(4)=41. $ TSTRT=0. $ TEN
D=100. $ DT=0.01 $ DTPR=10. $ DTFL=20. $ X(1)=0. $ X(23)=3*50. $
LARGE=2,85,63,45,72,15,38,92,100,12,3,4,50,5*10 $ LARGE(21)=5*1 $
ALL.
```

```
SIMCOMP VFRSION 3.0           PARAMETER VALUES

                                      - SIMULATION CONTROL PARAMETERS -

                                         TSTRT =            0
                                         TEND  =  100.000000
                                         DT    =  .100000000E-01
                                         DTPR  =  10.0000000
                                         DTFL  =  20.0000000

                                     - PRIMARY USER DEFINED VARIABLES -

         I(1-3) =            0         I(4) =  1.00000000            I(5-9) =  2.00000000          I(10) = INDEFINITE
         INDX =            1           J(1) =  1.00000000            J(2) =  3.00000000            J(3) =  5.00000000
         J(4-6) =  6.00000000          J(7-10) =  9.00000000         KVAL(1,1) =            1      KVAL(2,1) =           90
         KVAL(1,2-1,3) =            0  KVAL(2,3) =          500      PX =  999.999000              S(1,1) =  .200000000
         S(2,1-3,1) =  1.00000000      S(1,2) =  41.0000000          S(2,2) =  .300000000          S(3,2-2,3) =  1.00000000
         S(3,3) =  .500000000          S2(1) = INDEFINITE            S2(2) =  39.0000000           S2(3-4) = INDEFINITE
         TOP =  10.3000000             T1 =          20              T2 =           10             VA =  .369385000E-05
         VB =  2.00000000              VC =  3.14000000

                                    - SECONDARY USER DEFINED VARIABLES -

         LARGE(1) =             2      LARGE(2) =           85       LARGE(3) =           63       LARGE(4) =           45
         LARGE(5) =            72      LARGE(6) =           15       LARGE(7) =           38       LARGE(8) =           92
         LARGE(9) =           100      LARGE(10) =           12      LARGE(11) =            3      LARGE(12) =             4
         LARGE(13) =           50      LARGE(14-18) =           10   LARGE(19-20) = INDEFINITE     LARGE(21-25) =             1
         TEMP1 = INDEFINITE            TEMP2 = INDEFINITE
```

Event execution trace.

Simulations containing a large number of events, in which the logical structure for scheduling and rescheduling the events is complicated, are sometimes difficult to debug. It is important in such cases to determine that the events are being scheduled and executed in the proper sequence. The event-execution trace facility is provided to aid in debugging this type of simulation. The trace facility is envoked by including in the data section a card of the following form:

TRACE.

The command is free form in columns 1 through 80. Blanks are ignored.

The trace facility will print in the output the contents of the event stack, including event names, scheduled times of occurrence, and priorities, each time an event is executed. The current value of simulated time is also printed. Care should be taken in using the trace feature since in some simulations a very large amount of output can be produced.

2.2.3. Exogenous event requests.

Events can be scheduled externally prior to the start of simulation by statements of the following form:

EVENT. name, time, priority

The event name must be the name of an event defined in the source section or one of the system-defined events included in Table 2.1.3-1. The event HALT is the most commonly used system-defined event. The simulated time of occurrence, "time," must be either an integer- or real-valued constant. Integer-valued constants are converted to real-valued constants internally since TIME is a real-valued variable. The priority of the event, "priority," must be an integer- or real-valued constant in the range 1 to 512. Real-valued priorities specified on an event card are truncated to an integer. Priorities outside the range 1 through 512 are assumed to be 512. If the priority is not specified, a priority of one is assumed.

A maximum of 20 exogenous event requests can be included in the data section. If more than 20 requests are needed, endogenous event requests of any number could be included in subroutine START (refer to section 2.1.3). An example of an exogenous event request is contained in example 2.1.1-2.

2.3 Debugging.

During the course of programming a simulation, three phases of development occur. These are (i) the detection and correction of compilation errors, (ii) the detection and correction of execution errors, and (iii) the evaluation and refinement of the results of the simulation in preparation for production runs. Section 2.1.7 described the use of special execution controls. The default mode of execution should be used during the first phase of development. Once compilation errors have been eliminated, the DEBUG. mode of execution should be selected. The inclusion of a DEBUG. statement in the source section enables the simulation executive routine to detect arithmetic mode errors and produce a report. Arithmetic mode errors occur when illegal values are used in an arithmetic operation. Other types of errors can occur which will not produce a report, but are usually self explanatory such as an exceeded time limit.

Types of errors reported.

As stated above, arithmetic mode errors are detected when an illegal value is used in a computation. Table 2.3-1 can be the result of an operation, but will not be detected and reported until the resulting illegal value is used as an operand in a computation. Table 2.3-1 summarizes the illegal conditions which are detected and reported.

Table 2.3-1. Summary of error modes.

Error Mode	Condition
1	Address out of range - an attempt was made to reference central memory outside of established limits.
2	Operand out of range - the floating point arithmetic unit attempted to use an infinite operand.
3	Combined errors 1 and 2.
4	Indefinite operand - floating point arithmetic unit attempted to use an indefinite operand.
5	Combined errors 1 and 4.
6	Combined errors 2 and 4.
7	Combined errors 1, 2, and 4.

Error number one, address out of range, usually occurs when an index of a subscripted variable gets too large. Error numbers two and four occur when infinite or indefinite operands are used in the computation. The possible ways in which infinite or indefinite operands can be generated by division are illustrated in Table 2.3-2, using the following definitions of floating point (real) values. In Table 2.3-3, "X" represents any octal digit.

Table 2.3-2. Illegal results produced by division (A/B).

		B			
		+N	-N	+0	-0
A	+N	-	-	+∞	-∞
	-N	-	-	-∞	+∞
	+0	0	0	+IND	+IND
	-0	0	0	+IND	+IND

Table 2.3-3. Definition of floating point operands and results.

Mnemonic	Octal(internal) Representation	Meaning
+0	0000 X ... X	positive zero
-0	7777 X ... X	negative zero
+∞	3777 X ... X	positive infinite
-∞	4000 X ... X	negative infinite
+IND	1777 X ... X	positive indefinite
-IND	6000 X ... X	negative indefinite
N	--	any value with the exception of ±∞, ±IND, or ±0.

Positive or negative infinite results can be generated whenever a computation yields a floating point value whose absolute magnitude is outside the range 10^{-293} to 10^{322}. Such a condition can occur in iterative computations where the value of a variable grows exponentially. This case can happen quite easily in simulations where a flow is defined to be proportional to a state variable which is linked by the flow.

Variables which are declared in storage-allocation statements (i.e., STORAGE., INTEGER., and REAL. statements) or reserved simulation control variables including state variables which are used in the computation, but have not been given a value will cause a mode 4 error. All such variables are initialized to an indefinite value before the data section is processed. Assigning a variable a value in the data section or in the simulation prior to the use of the variable as an operand will avoid the detection of an indefinite value.

An indefinite operand will also be detected if the variable FLOW is not assigned a legal value within the range of each flow declaration. The source and destination state variables which are linked by flows must be assigned legal values prior to the start of simulation, or an indefinite operand will be detected.

Debug reports.

The following sample simulation is shown to illustrate the information contained in a debug report. The listing in example 2.3-1 is the source and data section used which produced the debug report which follows.

Example 2.3-1. A sample simulation containing an error. Note: circled numbers refer to items explained in the text.

```
DEBUG.
STORAGE. P.Q.R
(10-12). CALL PVAL
      FLOW=P*COS(TIME*6.28/50.)
      SUBROUTINE PVAL
      P=TIME/R+Q
      RFTURN
      END
```

$^{7}8_{9}$ *end-of-record separator*

```
TSTRT=0. $ TENDÅ=100. $ DT=1. $ X(10)=100. $ X(12)=0. $ R=2. $ DTPR=20. $
PRINT.
```

```
ARITHMETIC MODE ERROR                 DIAGNOSTIC DUMP                       07/23/73          10.55.59.
                               ①
 TYPE OF ERROR:                         ②
        ERROR MODE = 4

                FLOATING POINT ARITHMETIC UNIT RECIEVED AN INDEFINITE OPERAND
 ①
           OCCURING (APPROXIMATELY) AT ADDRESS 007622B WHICH IS LOCATION 000004B IN ROUTINE PVAL  ③

          NON-STANDARD FLOATING POINT ARITHMETIC - TABLES OF NON-STANDARD RESULTS BY DIVISION

                          DIVIDE (A/B)                      WHERE

                                   B                          +0   =   0000 X...X B
                          +N    -N    +0    -0                -0   =   7777 X...X B
                                                            +INF   =   3777 X...X B  (+ INFINITY)
                     +N   --    --    +INF  -INF            -INF   =   4000 X...X B  (- INFINITY)
                 A   -N   --    --    -INF  +INF            +IND   =   1777 X...X B  (+ INDEFINITE)
                     +0   0     0     +IND  +IND            -IND   =   6000 X...X B  (- INDEFINITE)
                     -0   0     0     +IND  +IND               N   =   ANY WORD EXCEPT +INF, -INF, +IND, -IND, +0, OR -0

 EXCHANGE JUMP PACKAGE:        ⑤         ⑥       ⑦               ⑧                ⑨                       ⑩

   ADDRESS                 REFERENCED     TYPE    INDEX   LOCAL    CONTAINED
  REGISTERS    CONTENTS   VARIABLE NAME  **ARRAY  (DEC)  ADDRESS      IN            VALUE (OCTAL)             DECODED VALUE

                                                                                    OUT OF RANGE
     A0        052000B                                                        
     A1        007610B                                            000015B  XFLOWS    0000 0000 0000 0000 0000B                      0
     A2        006136B                                            000101B  XEXECTV   0000 0000 0000 0001 0663B                   4531
     A3   ④    010422B                                            000010B  XTRNS*    0100 0000 0046 0004 6000B    1.3084312383453-280
     A4        016717B     R              REAL                    001761B  / /       1721 4000 0000 0000 0000B        2.00000000000
     A5        016716B     Q              REAL                    001760B  / /       1777 0000 0000 0000 1761B         + INDEFINITE
     A6        007615B     FLOW           REAL                    000022B  XFLOWS    1777 0000 0000 0000 0001B         + INDEFINITE
     A7        007614B     XMFL           INTEGER                 000021B  XFLOWS    0000 0000 0000 0000 0000B
 ⑪
                                                                                        INCREMENT
    OPERAND                                                                             REGISTERS     CONTENTS
   REGISTERS              CONTENTS                     DECODED VALUE
                                                                                            B0          000000B
      X0         0000 0000 0000 0000 0000B                           0                      B1          011530B
      X1         0000 0000 0000 0000 0000B                           0                      B2          000013B
      X2         0000 0106 6300 0000 0000B               4865124204544                      B3    ⑬    000013B
      X3   ( )   0100 0000 0046 0004 6000B         1.3084312383453-280  ⑫                   B4          000000B
      X4         1721 4000 0000 0000 0000B           2.00000000000                          B5          000001B
      X5         1777 0000 0000 0000 1761B            + INDEFINITE                          B6          000001B
      X6         1777 0000 0000 0000 0001B            + INDEFINITE                          B7          011527B
      X7         1777 0000 0000 0000 0000B            + INDEFINITE
```

```
VARIABLE DUMP  -  XFLOWS

       VARIABLE       TYPE                      LOCAL
         NAME        *ARRAY      LOCATION      ADDRESS      REPEATED          VALUE (OCTAL)                      DECODED VALUE

         XMFL        INTEGER     007614B       000021B                  0000 0000 0000 0000 0000B                                0
         FLOW        REAL        007615B       000022B                  1777 0000 0000 0000 0000B                  + INDEFINITE


VARIABLE DUMP  -  / /

 (III)   XADRS       *REAL       014736B       000000B                  5663 0663 3606 5301 3470B             -1.9006556451888E+37
         TIME        REAL        014737B       000001B                  0000 0000 0000 0000 0000B              0.
         TSTRT       REAL        014740B       000002B                  0000 0000 0000 0000 0000B              0.
         TEND        REAL        014741B       000003B                  1726 6200 0000 0000 0000B                100.0000000000
         DT          REAL        014742B       000004B                  1720 4000 0000 0000 0000B                1.00000000000
         DTPR        REAL        014743B       000005B        (14)      1724 5000 0000 0000 0000B               20.0000000000
         DTPL        REAL        014744B       000006B                  1777 0000 0000 0000 0000B                 + INDEFINITE
         DTFL        REAL        014745B       000007B                  1777 0000 0000 0000 0000B                 + INDEFINITE
         X           *REAL       014746B       000010B         9 _      1777 0000 0000 0000 0000B                 + INDEFINITE
                                                                        1726 6200 0000 0000 0000B                100.0000000000
                                                                        1777 0000 0000 0000 0000B                 + INDEFINITE
                                                                        0000 0000 0000 0000 0000B              0.
                                                             987 _      1777 0000 0000 0000 0000B                 + INDEFINITE
         P           REAL        016715B       001757B                  1777 0000 0000 0000 0000B                 + INDEFINITE
         Q           REAL        016716B       001760B                  1777 0000 0000 0000 0000B                 + INDEFINITE
         R           REAL        016717B       001761B                  1721 4000 0000 0000 0000B                2.000000000000
         XEVSTK      *REAL       016720B       001762B                  0074 0000 0010 0001 0424B              1.7216235906173-282
```

All debug reports contain the following items which refer to the circled numbers in example 2.3-1. A debug report contains three parts. These are (I) an explanation of the type of error, (II) the exchange jump package, and (III) the values of all variables in the simulation when the error was detected.

The first part contains (1) the error mode and (2) an explanation of the error mode (refer to table 2.3-1). After this information, (3) the routine in which the error was detected is listed. In this example an attempt was made to use an indefinite quantity in subroutine PVAL. We can immediately infer that either an indefinite value was generated by an undefined operation earlier in the simulation (such as 0 ÷ 0), or an attempt was made to use a variable which was never initialized. In this example the error was detected in the user-supplied routine PVAL. It is possible to have errors detected in the simulation executive routines. If an error is detected in routine XFLOWS, the illegal condition was detected while a flow was being computed. The executive routine which updates the state variables is called XCSIM. An error in this routine indicates that either the value of a flow is illegal or the value of a state variable is illegal. Errors can also be detected within FORTRAN-intrinsic functions such as ALOG and EXP and will be reported accordingly.

The second part of a debug report is the exchange jump package. This portion of the report reflects the contents and meaning of the operation registers at the time the error was detected. All computations in the computer are accomplished by operating on the values in these registers. The address registers (4) contain the addresses in central memory of variables whose values were currently being used or were recently used. If an address corresponds to the address of a user variable, the name (5), the mode (6), the one-dimensional array location (7) if the variable is an array, the routine or common block in which the variable is located (8), the internal representation (9), and the decimal representation (10) of the value contained in the location of the variable are reported. The operand registers (11) contain the values which were currently being used in the computation. The decimal equivalents (12) of the contents of the operand registers are also supplied. The increment registers (13) usually contain counters such as the indices of DO loops. Their contents are useful in debugging simulations only very rarely. In (8) common blocks are represented by names enclosed in slashes. The blank common block whose entries are denoted by / / refers to the location of storage of reserved-system variables and state variables in addition to variables declared in storage-allocation statements. For the example simulation we find that Q contains an indefinite value. Referring back to the listing in Fig. 2.3-1, we find that Q was never assigned a value. This was the cause of the error. In subroutine PVAL we had attempted to add the value of Q to the quantity TIME/R, but Q had not been given a value. A special note of caution is in order. The failure to determine the cause of an error through the use of the information contained in the exchange jump package is usually caused by trying to digest too much information. Many times much of the information is not relevant to the discovery of the error. In determining the cause of the error in the above example, we had combined the information that the error was detected in subroutine PVAL *along* with the information that Q was indefinite. We were thus not mislead by the fact that

FLOW is indefinite. FLOW is indefinite because it is not assigned a value until the call to PVAL is completed.

The third part of a debug report contains a listing of the values of the variables when the error was detected. Most of the information is self-explanatory. If a variable is an array which contains successive equal values, a repetition factor is used to conserve space (14). In this example the first nine state variables are indefinite. State variables X(10), X(11), and X(12) contain respectively the values 100.0, indefinite and 0. The remaining 987 state variables are indefinite. This is all satisfactory since X(10) and X(12) are the only state variables used in this simulation and are defined (i.e., given legal values). The variable XMFL in the routine XFLOWS has a special meaning. If an error is detected in routine XFLOWS, the value of XMFL + 1 points to the flow which was being computed at the time of the error. If XMFL equals zero, the first flow was being computed. If XMFL equals 10, the 11th flow defined in the simulation was being computed. Do not forget to count all flows in iteratively defined flows.

The following sample simulations are shown to illustrate various types of errors and the procedure for determining the cause of the errors using the debug report as a guide. Each table following a listing and a debug report contains the relevant information used in deducing the cause of the error. Practice is required in recognizing the relevant pieces of information which, when combined, produces an explanation of the error. Many times a single piece of information is misleading unless it is interpreted along with other pieces of information.

Example 2.3-2. Illustration of an uninitialized state variable (see Table 2.3-4).

```
DEBUG.
REAL. N
(2-3). FLOW=N*COS(TIME*6.28/50.) ④
789    end-of-record separator
TSTRT=0. $ TEND=100. $ DT=1. $ X(2)=100. $ N=10. $   ⑤
PLOT. (X(2).X(3))
```

```
ARITHMETIC MODE ERROR              DIAGNOSTIC DUMP                   07/23/73          1[illegible].51.57.

 TYPE OF ERROR:
        ERROR MODE = 4                                     ①

             FLOATING POINT ARITHMETIC UNIT RECIEVED AN INDEFINITE OPERAND

        OCCURING (APPROXIMATELY) AT ADDRESS 010667B WHICH IS LOCATION 000021B IN ROUTINE XCSIM ②

       NON-STANDARD FLOATING POINT ARITHMETIC - TABLES OF NON-STANDARD RESULTS BY DIVISION

                     DIVIDE (A/B)                     WHERE

                           B                             +0   =   0000 X...X B
                   +N    -N    +0    -0                  -0   =   7777 X...X B
                                                        +INF  =   3777 X...X B  (+ INFINITY)
              +N   --    --   +INF  -INF                -INF  =   4000 X...X B  (- INFINITY)
          A   -N   --    --   -INF  +INF                +IND  =   1777 X...X B  (+ INDEFINITE)
              +0   0     0    +IND  +IND                -IND  =   6000 X...X B  (- INDEFINITE)
              -0   0     0    +IND  +IND                  N   =   ANY WORD EXCEPT +INF, -INF, +IND, -IND, +0, OR -0

 EXCHANGE JUMP PACKAGE:

   ADDRESS              REFERENCED     TYPE    INDEX   LOCAL   CONTAINED
  REGISTERS  CONTENTS  VARIABLE NAME  *=ARRAY  (DEC)  ADDRESS     IN           VALUE (OCTAL)                 DECODED VALUE

     A0      052000B                                                         OUT OF RANGE
     A1      014726B     DT            REAL           000004B  / /      1720 4000 0000 0000 0000B       1.00000000000
     A2      010705B                                  000037B  XCSIM    0000 0000 0077 7770 0000B         1073709056
     A3      010711B                                  000043B  XCSIM    5170 0313 4321 1444 6000B  -1.34905939685l9+132
     A4      010706B                                  000040B  XCSIM    0000 0000 0000 0007 7777B              32767
     A5      014734B     X   ③        *REAL      3    000010B  / /      1777 0000 0000 0000 0013B         + INDEFINITE
     A6      014733B     X            *REAL      2    000010B  / /      1726 5500 0000 0000 0000B       90.0000000000
     A7      006020B                                  000000B  XXFL2WS  1723 4777 7777 7777 7777B  10.000000000000

   OPERAND                                                                            INCREMENT
  REGISTERS          CONTENTS                DECODED VALUE                           REGISTERS     CONTENTS

     X0       1723 4777 7777 7777 7777B     10.000000000000                              B0        000000B
     X1       1720 4000 0000 0000 0000B     1.000000000000                               B1        006020B
     X2       0000 0000 0077 7770 0000B           1073709056                             B2        006017B
     X3       0000 0000 0000 0000 0002B                    2                             B3        010712B
     X4       1726 5500 0000 0000 0000B     90.0000000000                                B4        777772B
     X5       1777 0000 0000 0000 0013B       + INDEFINITE                               B5        000001B
     X6       1777 0000 0000 0000 0000B       + INDEFINITE                               B6        777773B
     X7       0000 0000 0000 0000 0003B                    3                             B7        006017B
```

```
VARIABLE DUMP  -  XFLOWS

  VARIABLE     TYPE                    LOCAL
    NAME      *=ARRAY    LOCATION     ADDRESS    REPEATED          VALUE (OCTAL)                  DECODED VALUE

   XMFL       INTEGER    007606B      000017B               0040 0000 0000 0000 00018                           1
   FLOW       REAL       007607B      000020B               1723 4777 7777 7777 77778            10.000000000090

VARIABLE DUMP  -  / /

   XADRS      *REAL      014722B      000000B               5663 0663 3606 5301 34548          -1.9006556451889E+37
   TIME        REAL      014723B      000001B               0000 0000 0000 0000 00008           0.
   TSTRT       REAL      014724B      000002B               0000 0000 0000 0000 00008           0.
   TEND        REAL      014725B      000003B               1726 6200 0000 0000 00008            100.0000000000
   DT          REAL      014726B      000004B               1720 4000 0000 0000 00008            1.000000000000
   DTPR        REAL      014727B      000005B               1777 0000 0000 0000 00008             + INDEFINITE
   DTPL        REAL      014730B      000006B               1720 4000 0000 0000 00008            1.000000000000
   DTFL        REAL      014731B      000007B               1777 0000 0000 0000 00008             + INDEFINITE
   X          *REAL      014732B      000010B               1777 0000 0000 0000 00008             + INDEFINITE
                                                            1726 5500 0000 0000 00008            90.00000000000
                                                997 -       1777 0000 0000 0000 00008             + INDEFINITE
   N           REAL      016701B      001757B               1723 5000 0000 0000 00008            10.00000000000
   REVSTK     *REAL      016702B      001760B               0070 0000 0010 0001 04100           1.0760146479358-283
```

Table 2.3-4. Information used in determining the error in example 2.3-2. Item numbers refer to the circled numbers in example 2.3-2.

Item No.	Information
(1) and (2)	An indefinite quantity was used in XCSIM. Therefore either a flow or state variable was indefinite.
(3)	X(3) was indefinite.
(4) and (5)	X(3) is the source compartment for the flow, but was not initialized in the data section.

Example 2.3-3. Illustration of the failure to assign a value to FLOW. (see Table 2.3-5).

```
DEBUG.
(1-2). V=X(1)*0.01
       W=X(2)*COS(TIME*6.28/50.)
       FLW=W+V                            ⑥
⁷8₉    end-of-record separator
TSTRT=0. $ TEND=100. $ DT=1. $ X=100.,1. $
PLOT. (X(1)),(X(2))
```

```
ARITHMETIC MODE ERROR                  DIAGNOSTIC DUMP                        07/23/73         20.35.34.

 TYPE OF ERROR:
       ERROR MODE = 4                                                    ①

            FLOATING POINT ARITHMETIC UNIT RECIEVED AN INDEFINITE OPERAND

       OCCURING (APPROXIMATELY) AT ADDRESS 010671B WHICH IS LOCATION 000010B IN ROUTINE XCSIM ②

       NON-STANDARD FLOATING POINT ARITHMETIC - TABLES OF NON-STANDARD RESULTS BY DIVISION

                       DIVIDE (A/B)                          WHERE

                                 B                              +0    =    0000 X...X B
                     +N     -N     +0     -0                    -0    =    7777 X...X B
                                                              +INF    =    3777 X...X B  (+ INFINITY)
                +N   --     --    +INF   -INF                 -INF    =    4000 X...X B  (- INFINITY)
           A    -N   --     --    -INF   +INF                 +IND    =    1777 X...X B  (+ INDEFINITE)
                +0   0      0     +IND   +IND                 -IND    =    6000 X...X B  (- INDEFINITE)
                -0   0      0     +IND   +IND                    N    =    ANY WORD EXCEPT +INF, -INF, +IND, -IND, +0, OR -0

 EXCHANGE JUMP PACKAGE:

   ADDRESS              REFERENCED      TYPE    INDEX   LOCAL   CONT/INED
  REGISTERS  CONTENTS  VARIABLE NAME  *=ARRAY   (DEC)  ADDRESS     IN            VALUE (OCTAL)                DECODED VALUE

                                                                                 OUT OF RANGE
     A0      052000B
     A1      014733B      DT           REAL             000004B   / /      1720 4000 0000 0000 0000B        1.000000000000
     A2      010712B                                    000037B   XCSIM    0000 0000 0077 7770 0000B           1073709056
     A3      010716B                                    000043B   XCSIM    5957 3342 5555 5733 4055B   -3.5757499426797E+57
     A4      006010B                                    000000B   XXFL2WS  1777 0000 0000 0000 0001B       + INDEFINITE
     A5      006015B                                    000001B   XXFL1WS  0000 0000 0004 0010 0002B                 32770
     A6      006016B                                    000000B   XXFL2WS  1777 0000 0000 0000 0001B       + INDEFINITE
     A7      007610B      XMFL         INTEGER          000023B   XFLOWS   0000 0000 0000 0000 0001B                          1

   OPERAND                                                                                  INCREMENT
  REGISTERS           CONTENTS                 DECODED VALUE                                REGISTERS    CONTENTS

     X0        1777 0000 0000 0000 0000B       + INDEFINITE                                     B0        000000B
     X1        1720 4000 0000 0000 0000B     1.000000000000                                     B1        006016B
     X2        0000 0000 0077 7770 0000B         1073709056                                     B2        006015B
     X3        0000 0000 0000 0000 0001B                  1                                     B3        010717B
     X4        1777 0000 0000 0000 0001B       + INDEFINITE                                     B4        777772B
     X5        0000 0000 0000 0010 0002B              32770                                     B5        000001B
     X6        1777 0000 0000 0000 0001B       + INDEFINITE                                     B6        777773B
     X7        0000 0000 0000 0000 0001B                  1                                     B7        006015B

 VARIABLE DUMP  -  XFLOWS

   VARIABLE      TYPE                    LOCAL
     NAME       *=ARRAY     LOCATION    ADDRESS    REPEATED         VALUE (OCTAL)                  DECODED VALUE

     XMFL       INTEGER     007610B     000023B               0000 0000 0000 0000 0001B                       1  ③
     FLOW       REAL        007611B     000024B               1777 0000 0000 0000 0000B          + INDEFINITE
     V          REAL        007612B     000025B               1720 4000 0000 0000 0000B        1.000000000000
     W          REAL        007613B     000026B               1717 7777 7777 7777 7777B       1.0000000000000
     FLW        REAL        007614B     000027B               1721 4000 0000 0000 0000B        2.000000000000    ⑤

 VARIABLE DUMP  -  / /

     XADRS      *REAL       014727B     000000B               5663 0663 3606 5301 3461B     -1.9006556451889E+37
     TIME        REAL       014730B     000001B               0000 0000 0000 0000 0000B      0.
     TSTRT       REAL       014731B     000002B               0000 0000 0000 0000 0000B      0.
     TEND        REAL       014732B     000003B               1726 6200 0000 0000 0000B      100.0000000000
     DT          REAL       014733B     000004B               1720 4000 0000 0000 0000B      1.000000000000
     DTPR        REAL       014734B     000005B               1777 0000 0000 0000 0000B       + INDEFINITE
     DTPL        REAL       014735B     000006B               1720 4000 0000 0000 0000B      1.000000000000
     DTFL        REAL       014736B     000007B               1777 0000 0000 0000 0000B       + INDEFINITE
     X          *REAL       014737B     000010B               1726 6200 0000 0000 0000B      100.0000000000    ④
                                                              1720 4000 0000 0000 0000B      1.000000000000
                                                   997 _      1777 0000 0000 0000 0000B       + INDEFINITE
     XEVSTK     *REAL       016700B     001757B               0100 0000 0010 0001 0415B     2.7545976013290-281
```

Table 2.3-5. Information used in determining the error in example 2.3-3.

Item No.	Information
(1) and (2)	An indefinite quantity was used in XCSIM. Therefore either a flow or a state variable was indefinite.
(3)	The value of FLOW is indefinite.
(4)	The values of the state variables X(1) and X(2) are legal values.
(5) and (6)	A variable FLW contains a legal value but is obviously a mispunch for the variable FLOW.

Example 2.3-4. Illustration of the generation of an infinite operand (see Table 2.3-6).

```
DEBUG.
STORAGE. BIRT,POP,DENS,AREA,ARMIN
      EVENT POPL
      CALL EVENT(4HPOPL,TIME+1.,1)
      BIRT=3.*SIN(TIME*6.28/50.)
      POP=POP+BIRT/DENS
      DENS=POP/AREA            ⑤
      RETURN
      END
      EVENT CRWD
      CALL EVENT(4HCRWD,TIME+1.,1)
      APEA=EXP(-TIME/10.)
      IF(AREA.LT.0.5) AREA=ARMIN          ⑧
      RFTURN
      END
```

$^{7}8_{9}$ *end-of-record separator*

```
EVENT. POPL,0.
EVENT. CRWD,0.       ④       ⑥
EVENT. HALT,100.
PCP=3. $ DENS=3. $ AREA=1. $ ARMIN=0. $
PLOT. (BIRT),(POP),(DENS)
PLOT. (POP)/DENS                   ⑩
```

```
ARITHMETIC MODE ERROR                    DIAGNOSTIC DUMP                              07/23/73           10.57.46.

TYPE OF ERROR:
       ERROR MODE = 2                                           ①

                FLOATING POINT ARITHMETIC UNIT RECIEVED AN INFINITE OPERAND

       OCCURING (APPROXIMATELY) AT ADDRESS 007636B WHICH IS LOCATION 000012B IN ROUTINE POPL   ②

      NON-STANDARD FLOATING POINT ARITHMETIC - TABLES OF NON-STANDARD RESULTS BY DIVISION


                          DIVIDE (A/B)                        WHERE

                                    B                              +0    =    0000 X...X B
                        +N     -N     +0     -0                    -0    =    7777 X...X B
                                                                 +INF    =    3777 X...X B   (+ INFINITY)
                +N      --     --    +INF   -INF                 -INF    =    4000 X...X B   (- INFINITY)
            A   -N      --     --    -INF   +INF                 +IND    =    1777 X...X B   (+ INDEFINITE)
                +0      0      0     +IND   +IND                 -IND    =    6000 X...X B   (- INDEFINITE)
                -0      0      0     +IND   +IND                    N    =    ANY WORD EXCEPT +INF, -INF, +IND, -IND, +0, OR -0


EXCHANGE JUMP PACKAGE:

   ADDRESS               REFERENCED      TYPE     INDEX   LOCAL     CONTAINED
  REGISTERS  CONTENTS  VARIABLE NAME   *=ARRAY   (DEC)   ADDRESS      IN              VALUE (OCTAL)                DECODED VALUE

                                                                                      OUT OF RANGE
     A0       052000B                                                                                                              
     A1       013435B                                   000040B  SINCOSE   6105 3301 0145 2401 6617B  -2.7555218727710E-07
     A2       013436B                                   000041B  SINCOSE   1663 4334 1433 4416 3607B   2.0629106347665E-09
     A3       013441B                                   000044B  SINCOSE   1713 5252 5252 5252 3467B   4.1666666666470E-02
     A4       013442B                                   000045B  SINCOSE   6061 0000 0000 0000 0004B  -.500000000000
     A5       017046B      DENS          REAL           001761B  / /       3777 0000 0000 0000 0000B          + INFINITE    ③
     A6       005671B                                   000002B  XXEVENT   0000 0000 0000 0000 0010B                   8
     A7       007646B                                   000022B  POPL      1723 5000 0000 0000 0000B    10.0000000000


   OPERAND                                                                                      INCREMENT
  REGISTERS               CONTENTS                    DECODED VALUE                             REGISTERS     CONTENTS

      X0          0000 0000 0000 0000 0000B                        0                                B0         000000B
      X1          6073 3454 2514 6352 6171B        -2.6937350609458E-04                             B1         000001B
      X2          1673 7636 2501 0067 0417B         9.3093336849074E-07                             B2         000001B
      X3          0000 0000 0000 0000 0000B                        0                                B3         000000B
      X4          6061 0000 0000 0000 0004B        -.500000000000                                   B4         000057B
      X5          3777 0000 0000 0000 0000B           + INFINITE                                    B5         000003B
      X6          1717 7171 1274 5512 7144B         .9045827809445                                  B6         017057B
      X7          1721 5332 7015 4170 1313B        2.713748342033                                   B7         000006B



VARIABLE DUMP  -  XFLOWS

  VARIABLE      TYPE                     LOCAL
    NAME      *=ARRAY      LOCATION     ADDRESS      REPEATED          VALUE (OCTAL)                  DECODED VALUE

    XMFL      INTEGER      007623B      000004B                  0003 5003 1700 0000 0000B                           X

VARIABLE DUMP  -  / /

    XADRS     *REAL        015065B      000000B                  5663 0663 3606 5301 3617B      -1.9006556451881E+37
    TIME       REAL        015066B      000001B                  1723 4400 0000 0000 0000B       9.000000000000
    TSTRT      REAL        015067B      000002B                  0000 0000 0000 0000 0000B      0.
    TEND       REAL        015070B      000003B                  1777 0000 0000 0000 0000B         + INDEFINITE
    DT         REAL        015071B      000004B                  1777 0000 0000 0000 0000B         + INDEFINITE
    DTPR       REAL        015072B      000005B                  1777 0000 0000 0000 0000B         + INDEFINITE
    DTPL       REAL        015073B      000006B                  1720 4000 0000 0000 0000B       1.000000000000
    DTFL       REAL        015074B      000007B                  1777 0000 0000 0000 0000B         + INDEFINITE
    X         *REAL        015075B      000010B        999 _     1777 0000 0000 0000 0000B         + INDEFINITE
    BIRT       REAL        017044B      001757B                  1721 5040 7372 3261 3704B       2.532164196570
    POP        REAL        017045B      001760B                  1722 5156 1066 0240 1350B       5.215113658479
    DENS       REAL        017046B      001761B                  3777 0000 0000 0000 0000B          + INFINITE       ⑦
    AREA       REAL        017047B      001762B                  0000 0000 0000 0000 0000B      0.
    AMMIN      REAL        017050B      001763B                  0000 0000 0000 0000 0000B      0.
    XEVSTK    *REAL        01705.B      001764B                  0104 0000 0000 0101 0507B      0.7530724920936-283  ⑧
```

Table 2.3-6. Information used in determining the error in example 2.3-4.

Item No.	Information
(1) and (2)	An infinite operand was used in event POPL.
(3)	The variable DENS was + infinite.
(4)	DENS was initialized to a legal value.
(5)	DENS is computed as POP/AREA.
(6) and (7)	AREA was initialized to 1., but now contains the value 0.
(8)	AREA is recomputed in event CRWD and can assume the value of ARMIN.
(9) and (10)	ARMIN currently has the value zero and was mistakenly initialized to zero.

Example 2.3-5. Illustration of an out-of-range subscript (see Table 2.3-7).

```
DEBUG.
STORAGE. P(5),N          ⑤
(1-I=11,15). J=I+N
      FLOW=P(J)*X(I)
789    end-of-record separator                                ⑥
TSTRT=0. $ TEND=10. $ DT=1. $ DTPR=1. $
P=3*0.1,0.2,0.5 $ X(1)=1000. $ X(11)=30.,20.,25.,2*10. $ N=9000 $
PRINT.
```

```
ARITHMETIC MODE ERROR                  DIAGNOSTIC DUMP                           07/23/73            20.06.33.

 TYPE OF ERROR:
       ERROR MODE = 1                                           ①

              ATTEMPTED TO REFERENCE CENTRAL MEMORY OUTSIDE ESTABLISHED LIMITS

        OCCURING (APPROXIMATELY) AT ADDRESS 007614B WHICH IS LOCATION 000013B IN ROUTINE XFLOWS  ②

 EXCHANGE JUMP PACKAGE:

    ADDRESS             REFERENCED      TYPE    INDEX   LOCAL    CONTAINED
   REGISTERS  CONTENTS  VARIABLE NAME  *=ARRAY  (DEC)  ADDRESS      IN              VALUE (OCTAL)                 DECODED VALUE

                                                                                   OUT OF RANGE
      A0      052000B
      A1      010724B                                  000041B   XCSIM     0000 0000 0000 0000 0000B                 0
                                                                                   OUT OF RANGE
      A2      040323B
      A3      014704B        X          *REAL      11  000010B   / /       1724 7400 0000 0000 0000B        30.0000000000
      A4      007624B        J           INTEGER       000023B   XFLOWS    0000 0000 0000 0002 1463B                 9011
      A5      007622B        I           INTEGER       000021B   XFLOWS    0000 0000 0000 0000 0013B                   11
      A6      007622B        I           INTEGER       000021B   XFLOWS    0000 0000 0000 0000 0013B                   11
      A7      007624B        J           INTEGER       000023B   XFLOWS    0000 0000 0000 0002 1463B                 9011  ④

    OPERAND                                                                           INCREMENT
   REGISTERS            CONTENTS                 DECODED VALUE                        REGISTERS     CONTENTS

      X0       0000 0000 0000 0000 0002B                   2                             B0         000000B
      X1       0000 0000 0000 0000 0000B                   0                             B1         014704B
      X2       0000 0000 0000 0000 0000B                   0                             B2         000001B
      X3       1724 7400 0000 0000 0000B        30.0000000000                            B3         007624B
      X4       0000 0000 0000 0002 1463B                9011                             B4         000004B
      X5       0000 0000 0000 0000 0013B                  11                             B5         000003B
      X6       0000 0000 0000 0000 0013B                  11                             B6         000001B
      X7       0000 0000 0000 0002 1463B                9011                             B7         014710B
 VARIABLE DUMP  -  XFLOWS

   VARIABLE     TYPE                    LOCAL
     NAME      *=ARRAY     LOCATION     ADDRESS    REPEATED          VALUE (OCTAL)                DECODED VALUE    ③

   XMFL       INTEGER      007621B     000020B                0000 0000 0000 0000 0000B                     0
   I          INTEGER      007622B     000021B                0000 0000 0000 0000 0013B                    11
   FLOW       REAL         007623B     000022B                0003 5003 1700 0000 0000B      1.5677347514039-293
   J          INTEGER      007624B     000023B                0000 0000 0000 0002 1463B                  9011

 VARIABLE DUMP  -  / /


   XADRS      *REAL        014662B     000000B                5663 0663 3606 5301 3414B     -1.9006556451891E+37
   TIME        REAL        014663B     000001B                0000 0000 0000 0000 0000B      0.
   TSTRT       REAL        014664B     000002B                0000 0000 0000 0000 0000B      0.
   TEND        REAL        014665B     000003B                1723 5000 0000 0000 0000B      10.0000000000
   DT          REAL        014666B     000004B                1720 4000 0000 0000 0000B      1.00000000000
   DTPR        REAL        014667B     000005B                1720 4000 0000 0000 0000B      1.00000000000
   DTPL        REAL        014670B     000006B                1777 0000 0000 0000 0000B       + INDEFINITE
   DTFL        REAL        014671B     000007B                1777 0000 0000 0000 0000B       + INDEFINITE
   X          *REAL        014672B     000010B                1731 7640 0000 0000 0000B      1000.000000000
                                                    9 -       1777 0000 0000 0000 0000B       + INDEFINITE
                                                              1724 7400 0000 0000 0000B      30.0000000000
                                                              1724 5000 0000 0000 0000B      20.0000000000
                                                              1724 6200 0000 0000 0000B      25.0000000000
                                                    2 -       1723 5000 0000 0000 0000B      10.0000000000
                                                  984 -       1777 0000 0000 0000 0000B       + INDEFINITE
   P          *REAL        016641B     001757B      3 -       1714 6314 6314 6314 6315B      .1000000000000
                                                              1715 6314 6314 6314 6315B      .2000000000000
                                                              1717 4000 0000 0000 0000B      .5000000000000
   N           INTEGER     016646B     001764B                0000 0000 0000 0002 1450B                  9000
   XEVSTK     *REAL        016647B     001765B                0074 0000 0010 0001 0425B      1.7216236034440-282
```

Table 2.3-7. Information used in determining the error in example 2.3-5.

Item No.	Information
(1) and (2)	An attempt was made to reference central memory outside established limits in routine XFLOWS. Therefore the error occurred while evaluating a flow.
(3)	XMFL equals zero; therefore the first flow was being computed.
(4)	The value of J is 9011; much too large.
(5)	J is defined as the sum of I and N.
(6)	N was initialized to 9000, an error. N should have been initialized to -10.

LITERATURE CITED

Anderson, D. M. 1966. Computer programming FORTRAN IV. Appleton-Century-Crofts, Division of Meredith Publishing Co., New York. 435 p.

Control Data Corporation. 1972. Control data 6000 computer systems MIMIC digital simulation language reference manual. Pub. No. 44610400. Revision E. Control Data Corp., Software Documentation Dep., Sunnyvale, Calif.

Control Data Corporation. 1973. FORTRAN extended reference manual, control data 6000 computer systems. Pub. No. 60329100. Revision D. Control Data Corp., Software Documentation Dep., Sunnyvale, Calif.

Gustafson, J. D., and G. S. Innis. 1972. SIMCOMP version 2.0 user's manual. US/IBP Grassland Biome Tech. Rep. No. 138. Colorado State Univ., Fort Collins. 62 p.

Innis, G. S. 1972. The second derivation and population modeling: Another view. Ecology 53(4):720-723.

Markowitz, H. M., B. Hausner, and H. W. Karr. 1963. SIMSCRIPT: A simulation programming language. Prentice Hall, Inc., Englewood Cliffs, N. J. 138 p.

Naylor, T. H., J. L. Balintfy, D. S. Burdick, and K. Chu. 1966. Computer simulation techniques. John Wiley & Sons, Inc., New York. 352 p.

Parton, W. J., and G. S. Innis. 1972. Some graphs and their functional forms. US/IBP Grassland Biome Tech. Rep. No. 153. Colorado State Univ., Fort Collins. 41 p.

Smith, F. M. 1971. A volumetric-threshold infiltration model. Ph.D. Diss. Colorado State Univ., Fort Collins. 234 p.

APPENDIX A

DIAGNOSTICS

Compilation Diagnostics

The SIMCOMP compiler will list error messages immediately following the source card containing the rule infraction. Errors are either fatal, which are prefaced by *****FE, or nonfatal, which are prefaced by *****NF. Fatal errors cause abnormal termination, and execution of the simulation will not occur. Nonfatal errors only result in a diagnostic being issued, but should be corrected in a subsequent run. Unpredictable results in the execution are possible if nonfatal errors exist in the source section.

```
*****FE   A FIELD IN WHICH A CONSTANT SHOULD APPEAR IS MISSING OR IS NEGATIVE
*****FE   A VARIABLE DECLARATION IS INCOMPLETE AT CARD END
*****FE   ABOVE CARD ILLEGAL AT THIS POINT
*****FE   ARITHMETIC PHRASE MUST BE USED IN CONJUNCTION WITH A DO... PHRASE
*****FE   CHARACTER "_" IS ILLEGAL
*****FE   CHARACTER "_" IS ILLEGAL IN COLUMN __
*****FE   EXPECTED SUBSCRIPT MISSING
*****FE   EXPECTED VARIABLE NAME MISSING
*****FE   FLOW DIRECTIVE UNTERMINATED AT CARD END
*****FE   FLOW EXPRESSION SUB-FIELD "__________..." CONTAINS MORE THAN 10 NON-BLANK CHARS
*****FE   FLOW INDICES (____-____) PRODUCED BY THE ABOVE LABEL ARE OUTSIDE THE RANGE 1 - 999
*****FE   FLOW ITERATION PHRASE CONTROL VARIABLE "_____" MUST BE A 5 CHAR OR LESS INTEGER VARIABLE
*****FE   FLOW PHRASE "________________________________________..." CONTAINS MORE THAN 40 NON-BLANK CHARS
*****FE   INSUFFICIENT FIELD LENGTH, INCREASE BY (NO. OF FLOWS - ___)
*****FE   NUMBER OF DECLARED VARIABLES HAS EXCEEDED ______
*****FE   NUMBER OF FLOWS EXCEEDS 9999
*****FE   NUMBER OF USER-DEFINED EVENTS EXCEEDS 100
*****FE   ROUTINE NAME LONGER THAN 5 CHARS OR MISSING
*****FE   ROUTINE NAME MISSING
*****FE   ROUTINE NAME "_______..." LONGER THAN 7 CHARS
*****FE   ROUTINE NAME "______' STARTS WITH AN ILLEGAL CHAR
*****FE   SUBSCRIPT "____..." IS LONGER THAN 4 CHARS
*****FE   SUBSCRIPT "____" GREATER THAN 1023
*****FE   SUBSCRIPT "____" NOT DECODABLE
*****FE   THE DO... PHRASE CONTROL VARIABLE MUST BE THE OPERAND IN THE ARITHMETIC PHRASE
*****FE   VARIABLE "_____..." IS LONGER THAN 5 CHARS
*****FE   VARIABLE "_____" BEGINS WITH A NON-ALPHABETICAL CHAR
*****NF   VARIABLE "_____" BEGINS WITH CHAR "X"
*****NF   VARIABLE "_____" HAS BEEN PREVIOUSLY DECLARED, LAST DECLARATION IS ASSUMED CORRECT
*****FE   VARIABLE "_____" IS A RESERVED SYSTEM VARIABLE
```

Data Section Diagnostics

Errors encountered while processing the data section are reported in the output by a general message of the following form:

```
***** ERROR IN PRINT REQUEST
***** ERROR IN FLOW PRINT REQUEST
***** ERROR IN EXOGENOUS EVENT REQUEST
***** ERROR IN DATA ASSIGNMENT
***** ERROR IN PLOT REQUEST
```

One of these messages is followed by the card containing the infraction and one of the following diagnostics. All errors reported in the data section are nonfatal. The system will attempt to execute the simulation regardless of errors in the data section. Execution errors can occur because of errors in data assignment and exogenous-event requests.

```
CHARACTER "_" IS ILLEGAL IN COLUMN __
DATA ITEM "____________________..." LONGER THAN 20 CHARS
DATA REPETITION FACTOR "__________..." LONGER THAN 10 CHARS
DATA REPETITION FACTOR "__________" LESS THAN OR EQUAL TO ZERO
DATA REPETITION FACTOR "__________" NOT DECODABLE
EVENT NAME "_____..." LONGER THAN 5 CHARS
EVENT "_____" IS NON-EXISTANT
EVENT "_____" SCHEDULED AT TIME __________ AT PRIORITY OF __
EXPECTED FLOW INDEX MISSING IN OR BEFORE COLUMN __
EXPECTED VARIABLE NAME MISSING IN OR BEFORE COLUMN __
FLOW INDEX "___..." LONGER THAN 3 CHARS
FLOW INDEX "___" NOT DECODABLE OR OUT OF RANGE
FLOW INDICES UNTERMINATED AT CARD END
FLOW PRINTING REQUESTED - NO FLOWS DEFINED
FLOW (__,__) DOES NOT EXIST
ILLEGAL CHARACTER DETECTED "_"
ILLEGAL CHARACTER IN RANGE DECLARATION
IMPROPERLY FORMATTED LOG REQUEST
INTEGER VARIABLE _____ WAS ASSIGNED A REAL VALUE IN THE DATA SECTION
MISSING EXPECTED VARIABLE NAME OR DATA ITEM IN OR BEFORE COLUMN
MORE THAN 100 VARIABLES NAMED IN PLOT REQUESTS, THIS AND SUBSEQUENT PLOT REQUESTS IGNORED
MORE THAN 200 VARIABLES REQUESTED FOR PRINT
NO. OF EXOGENOUSLY SCHEDULED EVENTS EXCEEDS 20, ABOVE REQUEST IGNORED
NO. OF GROUPS PER PLOT IS .GT. 5
NO. OF VARIABLES PER PLOT IS .GT. 5
RANGE DECLARATION .GT. 10 CHARACTERS--THE UPPER LIMIT
REAL VARIABLE _____ WAS ASSIGNED AN INTEGER VALUE IN THE DATA SECTION
SUBSCRIPT "____..." LONGER THAN 4 CHARS IN COLUMN __
SUBSCRIPT "____" NOT DECODABLE
TIME OR PRIORITY LONGER THAN 20 CHARS AT COLUMN __
VARIABLE HAS .GT. 3 SUBSCRIPTS
VARIABLE NAME IS .GT. 5 CHARACTERS
VARIABLE SUBSCRIPT .GT. 999--THE UPPER LIMIT
VARIABLE "_____..." LONGER THAN 5 CHARS
VARIABLE "_____" WAS NOT COMPLETELY DECLARED BY CARD END
VARIABLE "_____" WAS NOT DECLARED IN A <STORAGE.> STATEMENT
```

APPENDIX B

DECK ORGANIZATION AND CONTROL CARDS

A typical SIMCOMP job is executed by means of the following control cards.

```
TAxxx, Annnnnnn.  (job card)
ATTACH,SIMCOM,SIMCOM3,CY=1,MR=1,ID=NREL.
SIMCOM.
```

${}^{7}8_{9}$ (end of record)

⋮

source section

⋮

${}^{7}8_{9}$

⋮

data section

⋮

${}^{6}7{}_{8}{}_{9}$ (end of file)

In actuality more control cards than those shown are utilized in executing the simulation. The SIMCOMP compiler generates a series of control cards which are used subsequent to the loading and execution of the compiler. As described in section 2.1.7, a SIMCOMP simulation can be executed in three different modes. If a NOGO. execution directive is included in the source section, the generation of these control cards by the compiler is inhibited. If the default mode or the DEBUG. mode of execution is selected, standard sets of control cards are generated automatically which are used after the SIMCOM. control card is executed. If a NOGO. directive is included in the source section, the following control cards and deck structure is equivalent to the deck in the above example with the default execution mode selected, i.e., by the absence of any execution directives. In the following case the user is supplying the control cards rather than having the compiler generate them automatically.

```
TAxxx, Annnnnnn.  (job card)
ATTACH,SIMCOM,SIMCOM3,CY=1,MR=1,ID=NREL.
SIMCOM.
FTN,I=SIMPRG,ROUND=T-*/,S=0,LRN=0.
ATTACH,B,SIMCOM3,CY=2,MR=1,ID=NREL.
ATTACH,LIB,SIMCOM3,CY=3,MR=1,ID=NREL.
SELECT.
COPYBF,B,LGO.
LOAD,LGO.
NOGO.1/
REWIND,NEWT1.
SELECT,P=PRELOAD,I=PRELOAD.
PRELOAD,NEWT1,MAIN.
MAIN.

789

NOGO.
          .
          .
          .
   source section
          .
          .
          .
789
          .
          .
          .
   data section
          .
          .
          .
6789
```

Similarly the following example is equivalent to the first example if a DEBUG. directive had been included in the source section. Here again the user is supplying the required additional control cards since the automatic

---

1/ Not to be confused with the special execution directive NOGO. which is included in the source section.

generation of the control cards is suppressed by the NOGO. directive in the source section.

```
TAxxx, Annnnnnn.  (job card)
ATTACH,SIMCOM,SIMCOM3,CY=1,MR=1,ID=NREL.
SIMCOM.
FTN,I=SIMPRG,LN=DEBUG,R=1,S=0,ROUND=T-*/.
ATTACH,B,SIMCOM3,CY=2,MR=1,ID=NREL.
ATTACH,LIB,SIMCOM3,CY=3,MR=1,ID=NREL.
SELECT.
COPYBF,B,LGO.
MAP,PART.
LOAD,LGO.
NOGO.
REWIND,NEWT1.
SELECT,P=PRELOAD,I=PRELOAD.
PRELOAD,NEWT1,MAIN.
MAIN.
```

$^{7}8_{9}$

NOGO.

⋮

source section

⋮

$^{7}8_{9}$

⋮

data section

⋮

$^{6}{}_{7}{}_{8}{}_{9}$

<u>Job Limits</u>. The job card illustrated in the above examples implicitly requests the minimum amount of time, pages printed, cards punched, and core required for a simple SIMCOMP job. These limits are:

Limit	Mnemonic on Job card	Meaning
Time	T16	16 seconds CPU time
Core	CM43000	43000 octal words of central memory
Printed pages	PR10	10 printed pages
Punched cards	PU10	10 punched cards

These limits are usually adequate only for the smaller SIMCOMP jobs. The limits specified on the job card should reflect the physical size of the simulation and the number of time steps (i.e., (TEND-TSTRT)/DT or the total number of events executed) used during the execution of the simulation. This is true for the time and core requirements. The number of printed pages is a function of the listing length and the number of output requests. A punch limit is required only if routine PUNCHD is called (refer to section 2.1.5). Only experience can be used to estimate these limits.

INDEX