SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

151 009

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257

SHARE PROGRAM LIBRARY SUBMITTAL FORM

SHARE PROGRAM LIBRARY AGENCY
Triangle Universities Computation Center
Post Office Box 12076
Research Triangle Park, North Carolina USA 27709

SPLA CONTROL NUMBER: 242

This form should be completed and submitted with the program package to the SHARE Program Library Agency at the address shown above. Standards and instructions for submitting programs are in the SHARE Reference Manual, Section 6.

(1) Program Number (to be filled by SPLA) . . . . . 360D - 15.1.009

(2) Title of Program . . . NDTRN2

(3) System Type(s) (Machine) . . . . . IBM series 360/370

(4) Search Key(s) . . . . .

(5) Programming Systems/Languages . . . . 1968 ANSI FORTRAN

(6) Primary Subject Code . . . . .

(7) Minimum System Requirements 126,000 byte partition plus 5000 x80 direct access disk area

(8) New (N) or Revision (R) (if revision, show prior Program Number in Item 1) N

(9) Date of Submittal . . . . .

(10) Documentation (number of original pages submitted) . . . . .

(11) Author's Name and Address . . . . . William I. Davisson & John J. Uhran Jr.
University of Notre Dame , Economics Dept.
Notre Dame, Indiana 46556

(12) Direct Technical Inquiries to Name & Address (if different than Author)

W.I. Davisson
Economics Dept.
Univ. Notre Dame
Notre Dame, Indiana 46556

J.J. Uhran Jr.
Dept. Elect. Engr.
Univ. Notre Dame
Notre Dame, Indiana 46556

(13) Submitter's Installation Membership Code . . . . .

(14) Abstract (should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.

DISCLAIMER

Triangle Universities Computation Center (TUCC) serves solely as the distribution agent for contributed programs and does not test or maintain them. They are distributed essentially in the original form submitted by the author. Neither TUCC nor SHARE, INC., makes any warranty, expressed or implied, as to the documentation, function, or performance of the contributed programs.

SHARE PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide:

a. Purpose
b. Programming Language used
c. Version and modification level or release number
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements

NDTRN2 is a continuous and stochastic simulation and modeling language. NDTRN2 has three integration methods, syntax diagnostic checking, execution time diagnostic checking as well as a wide range of print and plot options. All of these are control-card options. NDTRN2 will do a wide range of continuous and stochastic simulations including the specific SYSTEM DYNAMICS type of models.

NDTRN2 has been developed using the 1968 version of ANSI FORTRAN.

A twenty page implementation guide is included describing installation on all IBM , and most digital equipment corporation computers.

***** please note implementation guide for additional information.

(Please attach additional pages if necessary) . ***** . Total pages attached _____

An "Acknowledgement of Assistance" statement must be attached to this Submittal Form.

Permission to Publish

"I hereby give the SHARE Program Library Agency permission to reprint, reproduce, and distribute this program"

(15) Signature of Submitter and Date W. L. Davisson

(15) Signature of Installation Addressee _____

SPLA 1001 (R) Revised 11-76

NDTRAN2 - IMPLEMENTATION MANUAL

For additional information:

William I. Davisson
Department of Economics
University of Notre Dame
Notre Dame, Indiana 46556
Phone: 219 - 283 - 7021

John J. Uhran Jr.
Department of Electrical Engineering
University of Notre Dame
Notre Dame, Indiana 46556
Phone: 219 - 283 - 7423

University of Notre Dame
1980

IEB-8

NDTRAN2 - IMPLEMENTATION MANUAL

The enclosed magnetic tape (or cards) include the interpreter NDTRAN2 along with 43 test programs. These programs include the WORLD3 model discussed in LIMITS TO GROWTH and DYNAMICS OF GROWTH IN A FINITE WORLD.* The programs included include some illustrative programs that are discussed in Chapter 7 of the NDTRAN2 Manual, a substantial number of test programs (some of which are discussed in Chapter 6) of the NDTRANT2 Manual.

A. Description of NDTRAN2

NDTRAN2 is a dynamic simulation interpreter capable of carrying out dynamic and stochastic simulations. It is available in two basic versions:

1) A research version that will accept a source simulation program of up to 4000 statements.
2) A standard version which has the capability of executing programs somewhat larger than WORLD3 and which will operate on a computer with a 28K FORTRAN batch partition, such as the PDP 11 series computers.

   The standard version is available in a DOUBLE PRECISION version or a SINGLE PRECISION version depending on the requirements of the computer at the installing college or university. All versions sent to installations with PDP 11 series computers are single precision, requiring the half-word integer option.

The source program for NDTRAN2 is approximately 13,000 statements in length of which about 8,000 are COMMENT cards which constitute the basic documentation for the NDTRAN interpreter itself. It is very important that you read the COMMENT statements in the main program NDTRN, since this contains a number of parameters which must be set for the installation computer.

A number of changes have been made in NDTRAN2 as compared with NDTRAN. The major changes are as follows:

1. A change was made in the execution modules whereby the manner of determining the TIME variable was improved and made consistent with the method of integrating the other variables in NDTRAN2.
2. The code was documented with COMMENT statements.
3. The capacity of the NDTRAN2 standard version will fit on the same size computer and the same core partition as NDTRAN, but NDTRAN2 will handle a considerably larger model (optimization).
4. NDTRAN2 is considerably faster than NDTRAN (both using the CHECK option). Depending on the size and nature of the simulation being run, NDTRAN2 appears to be about 100 percent to 500 percent faster than NDTRAN.
5. NDTRAN2 has a CHECK and NOCHECK option. The CHECK option checks for execution-time errors as described in APPENDIX C of the Users Manual. For programs that are

known to be error-free, the user may opt for the NOCHECK option which will provide considerably faster execution times (as compared with the CHECK option).

6. NDTRAN2 has a comparative plot feature for program RERUNS. Rerunning the program, the user may compare the value of variables across all RUNS using the Comparative PRINT and PLOT feature.
7. NDTRAN2 has full documentation capability for any user program.
8. NDTRAN2 permits change of integration method with reruns.
9. The table functions have been improved and made considerably more efficient.
10. If the user obtains the NDTRAN2 standard version and(with sufficient core on the computer) may change to the research size version of NDTRAN2.

B. Test Programs

We found out from our experience with NDTRAN, that users desired more test programs. Therefore we have included on the magnetic tape with NDTRAN2 a total of 44 test programs including many of the standard short programs shown in the manual. We have also include a few major simulation models including the CEDAR LAKE MODEL and WORLD3 , including the overall model and the individual segments or modules of WORLD3 such as the POPULATION module. These have been done with the approval of DENNIS MEADOWS of Dartmouth. All documentation for WORLD3 models are available through MIT Press or from Meadows at Dartmouth College. The sources are LIMITS TO GROWTH and DYNAMICS OF GROWTH IN A FINITE WORLD.

The order of the test programs on the magnetic tape is shown on the figure on the following page. the test programs are, respectively, in separate files that are numbered. Each file contains a file number and a name. The test programs may be retrieved from the magnetic tape using the appropriate file number and name.

C. Direct Access Disk File

NDTRAN and NDTRAN2 were both designed to operate on a small computer, one with limited core capacity. Since the core was limited it was necessary to use the disk file for intermediate storage. Thus NDTRAN2 can handle quite large programs using the intermediate storage file. The manner in which the computer you have handles the disk file can have a significant impact in determining the run-time of your programs.

Our experience has been that all IBM systems require that any direct access disk file be completely formatted before the file may be used, regardless of the size of the program being executed. Therefore on IBM systems (and computers with like operating characteristics) the time that it takes to format the file will be added to the CPU time for the first run.

For instance, a small program (like the CLIFF test program) executing on our IBM 370/168 using the research version of NDTRAN2 has the capability of handling a simulation program with 4000 statements. When the computer formats that file in order to execute any program the charge for the CLIFF program is about $28.00. When the CLIFF program is run a second time against that file the charge is about 15 cents. All persons using IBM systems are urged to consider a type

TAPESCAN 3.4 - GSFC TAPE ANALYSIS AND COPYING PROGRAM 09/14/80 19:14:39.1 INPUT VOLS

VOLUME TABLE OF CONTENTS FOR NDTRN2 9 TRACK DAVISSON

SEQ.	DATA SET NAME	RECFM	LRECL	BLKSIZE	DEN	TRTCH	MAX BLK	MIN BLK	AVG BLK	BLK COUNT
0001	CLIFF	FB	00080	00800	1600		00800	00080	00560	000008
0002	XERR1	FB	00080	00800	1600		00800	00480	00640	000002
0003	CLIP	FB	00080	00800	1600		00800	00320	00540	000003
0004	COAL1	FB	00080	00800	1600		00800	00480	00640	000002
0005	PLOT5	FB	00080	00800	1600		00800	00560	00720	000003
0006	CONCARD	FB	00080	00800	1600		00800	00560	00580	000002
0007	DEL	FB	00080	00800	1600		00800	00720	00760	000002
0008	DELRAMP	FB	00080	00800	1600		00800	00560	00680	000002
0009	ERRORS	FB	00080	00800	1600		00800	00240	00720	000007
0010	FUNCT	FB	00080	00800	1600		00800	00400	00700	000004
0011	INTRST	FB	00080	00800	1600		00800	00560	00680	000002
0012	MACRO	FB	00080	00800	1600		00800	00560	00580	000002
0013	STEPDLS	FB	00080	00800	1600		00800	00320	00560	000002
0014	SMOOTH	FB	00080	00800	1600		00800	00160	00480	000002
0015	TABLES	FB	00080	00800	1600		00800	00400	00600	000002
0016	CLIFF2	FB	00080	00800	1600		00800	00320	00640	000003
0017	COAL2	FB	00080	00800	1600		00800	00480	00540	000002
0018	EXPTST	FB	00080	00800	1600		00800	00560	00720	000003
0019	INTGRT1	FB	00080	00800	1600		00800	00720	00760	000002
0020	INTGRT2	FB	00080	00800	1600		00800	00720	00760	000002
0021	INTGRT3	FB	00080	00800	1600		00800	00720	00760	000002
0022	LAKE	FB	00080	00800	1600		00800	00401	00756	000009
0023	PLOT2	FB	00080	00800	1600		00800	00400	00600	000002
0024	PLOT3	FB	00080	00800	1600		00800	00400	00600	000002
0025	PLOT4	FB	00080	00800	1600		00800	00400	00600	000002
0026	PLOT6	FB	00080	00800	1600		00800	00400	00600	000002
0027	POISSON	FB	00080	00800	1600		00800	00640	00747	000003
0028	PULSE	FB	00080	00800	1600		00800	00480	00640	000002
0029	SPIRAL	FB	00080	00800	1600		00800	00640	00720	000002
0030	STEP	FB	00080	00800	1600		00800	00800	00800	000002
0031	PLOTSTAT	FB	00080	00800	1600		00800	00400	00500	000002
0032	TABLE	FB	00080	00800	1600		00800	00560	00580	000002
0033	TABLE1	FB	00080	00800	1600		00800	00320	00560	000002
0034	XERR2	FB	00080	00800	1600		00800	00560	00580	000002
0035	XERR3	FB	00080	00800	1600		00800	00560	00680	000002
0036	XERR4	FB	00080	00800	1600		00800	00560	00680	000002
0037	XERR5	FB	00080	00800	1600		00800	00560	00580	000002
0038	HEAT	FB	00080	00800	1600		00800	00640	00792	000021
0039	WORLD3	FB	00080	00800	1600		00800	00641	00796	000039
0040	AGRIC	FB	00080	00800	1600		00800	00320	00770	000016
0041	CAPITAL	FB	00080	00800	1600		00800	00320	00756	000011
0042	RES	FB	00080	00800	1600		00800	00480	00735	000005
0043	POP	FB	00080	00800	1600		00800	00560	00784	000015
0044	LINK2	FB	00080	00800	1600		00800	00480	00720	000004
0045	NDTRN2	FB	00080	00800	1600		00800	00720	00800	001127
0046	STANDARD	FB	00080	00800	1600		00800	00800	00800	001127

07

of permanently formatted intermediate file, at least for all large or research runs. The definition of the DEFINE FILE statement found in the NDTRN main program , on line 450; is shown on the following page (page 7). The procedure that we use for executing the research version of NDTRAN2 is shown on page 8.

You may wish to make some test runs to determine the time that may be saved on your computer by using the FT01 file as permanently allocated to your system library.

D. OVERLAY

For all computer installations that do not have sufficient core, an overlay structure is provided which operates for both the standard and research versions of NDTRAN2. The following discussion is keyed to the IBM overlay methods because we are most familiar with them. The actual procedures to load NDTRAN2 to different computers (assuming the indicated overlay) are shown in PART F.

There are four regions for the overlay of NDTRAN2. The first region contains the root segment and the programs that control the execution of NDTRAN2. The overlay structure is shown on page 9. The IBM system, using the REGIONS approach to overlay permits overlay segments in any given region to be swapped in and out as required. Thus any of the segments in OVERLAY REGION TWO may be swapped in and out as required by any segment in OVERLAY REGION ONE. Unless the REGION characteristic is used this is not possible.The overlay is structured (given the subroutines of NDTRAN2) so as to maximize the efficiency of execution of NDTRAN2, and to minimize the amount of swapping done.

DEFINE FILE STATEMENT - NDTRAN2 line 450

The DEFINE FILE STATEMENT assigns unit numbers to and describes the characteristics of direct access disk files.

```
DEFINE FILE 1(5000,80,U,ASC3), ...
            ↑ ↑  ↑ ↑  ↑
            a b  c d  e
```

a) the unit number - direct access I/O statements will reference this file described by this statement using the file number:

```
READ(1'10) RECORD

     unit number
          record # 10
               storage locations beginning
                  with RECORD(1)
```

b) the number of records in the file.

c) the number of integer words per record.

d) FORMAT indicator - U indicates that FORMAT STATEMENTS will not be used and that data should be written/read as it is stored in memory.

e) the name of the associated variable - direct access I/O updates this variable so that it always contains the number of the record one after the last record written or read. The associated variable is updated whether it is used in the I/O statement or not. If another variable is used to supply the record number in an I/O statement it is not modified.

NDTRAN2 does not use the associated variable feature. If it is not supported the result should be unaffected.

```
00010 //FNDTRN6 JOB [redacted]
00020 //    TIME=2,REGION=192K,MSGLEVEL=(1,1)
00030 //S1 EXEC PGM=NDTRAN2
00040 //STEPLIB DD DSN=FNDTRN.NDTRAN2.LOAD,DISP=SHR
00050 //FT01F001 DD DISP=SHR,DSN=ACAD.ND2TRAN
00060 //FT05F001 DD DSN=FNDTRN.SOCSEC.DATA,DISP=SHR
00070 //FT06F001 DD SYSOUT=A
00080 //
READY
```

→ designation of system output - here to line printer

→ Location of simulation program to be executed by NDTRAN2. Identification of the Input File, File 5.

→ The intermediate file used by NDTRAN is a permanent part of the academic main program. File 1. See NDTRN, lines 380 - 450.

→ Location of the NDTRAN2 load module. -- STEPLIB.

```
INSERT MAIN
OVERLAY ONE
INSERT NDT03
OVERLAY ONE
INSERT NDT01,NDT02,NDT08,NDT09,NDT15,NDT19
OVERLAY ONE
INSERT NDT04,NDT48,NDT50,NDT51,NDT52,NDT53,NDT54,NDT55,NDT56,          X
               NDT58,NDT60
OVERLAY ONE
INSERT NDT61,NDT62,NDT66,NDT67,NDT68,NDT69,NDT76
OVERLAY ONE
INSERT NDT63,NDT75,NDT84
OVERLAY ONE
INSERT NDT70,NDT71,NDT72,NDT73,NDT74,NDT79,NDT80
OVERLAY TWO(REGION)
INSERT NDT05,NDT16,NDT17,NDT25,NDT26,NDT28,NDT38
OVERLAY TWO
INSERT NDT07,NDT10,NDT11,NDT30,NDT31,NDT32,NDT33,NDT35,NDT36
OVERLAY TWO
INSERT NDT44,NDT77,NDT78,NDT81,NDT82,NDT83,NDT86,NDT87
OVERLAY TWO
INSERT NDT65
OVERLAY TWO
INSERT NDT64
OVERLAY THREE(REGION)
INSERT NDT18,NDT20,NDT27,NDT37,NDT42,NDT59,NDT85
OVERLAY THREE
INSERT NDT22,NDT46,NDT47,NDT49,NDT57
OVERLAY THREE
INSERT NDT24,NDT29,NDT39
OVERLAY THREE
INSERT NDT44,NDT78,NDT77
OVERLAY FOUR(REGION)
INSERT NDT06,NDT12,NDT13,NDT14,NDT23,NDT43,NDT45
OVERLAY FOUR
 INSERT NDT21,NDT34,NDT40,NDT41
END OF DATA
EDIT
```

NDTRAN2 is basically a two-pass system. The first pass initializes all variables and does a full syntax check. The second pass does the execution of the program and provides for program output. For instance in REGION ONE, the subroutines (other than the root segment) are called in order:

NDT03 - initializes NDTRAN2.
NDT01 and NDT02
- checks syntax of all statments in a program.
NDT04 - provides for all control card options requested.
NDT61 - provides all initial values for the input program and loads the program.
NDT63 - provides for execution of the program(object code).
NDT70 - provides for printed and plotted output.

If you have any questions on the Overlay information please contact us.

E. Size of COMMON Data in Program

The arrays and the intermediate file disk in NDTRAN2 determine the amount of core required, the size of the disk space required and the size of the simulation program that may be handled. There are two versions of NDTRAN2, the standard version and the research version.

1. For the Intermediate File, Line 450 of NDTRN:

a) standard version:

```
DEFINE FILE 1(5000,80,U,ASC1)
```

b) Research version:

```
Define File 1(38000,80,U,ASC1)
```

Additional parameters in NDTRN(main program) must be modified depending on the computer on which you will implement NDTRAN2. These parameters are indicated by COMMEND cards.

2. For the Array Sizes:

ARRAY NAME	STANDARD	RESEARCH
PTRS	45	45
TITLE	120	120
CRSET	39	39
OPER	9	9
SYM	15	15
SUBSC	6	6
TYPCT	20	20
EQCHN	80	80
TOKEN	80	80
CARD1	80	80
CARD2	80	80
ERROR	80	80
OBJCD	160	160
DEF	80	80
XREF	80	80
TMAP	80	80
FCTN	5,22	5,22
SYMTB	5,512	5,512
LITBL	1024	8192
INITO	5772	21644
CDATA	144	144
OUTPT	240	240
EQSRT	2048	16384
VAR	5000	18705
LINE	120	120
OBJCD	5154	37410
SORT	2048	16384

In the event of size problems, variable VAR may be modified in size somewhat. This will affect the overall size of the program that may be executed.

The names of the subroutines and a short comment as to their functions is shown on pages 12 - 15, below.

SUB-OUTLINES

Ø1 - input analyzer

Ø2 - expnd processor

Ø3 - initialize FCTN, SYMTB, MACRO records,

Ø4 - pgm context analysis director

Ø5 - statement data area initializer

Ø6 - title processor

Ø7 - control card processor

Ø8 - equation laxical

Ø9 - output lexical/compiler

1Ø - macro strnt processor

11 - DEF processor

12 - system error msg issuer

13 - lexical error processor

14 - context error processor

15 - table processor

16 - eqn compiler

17 - default DEF generator

18 - RERUN processor

19 - RERUN input processor

2Ø - Group input error processor

21 - equation chain builder

22 - numeric processor

23 - TMAP element builder

24 - variable syntax

25 - function processor

26 - subscript syntax

27 - variable processor

Sub-Outlines (continued page 2.)

28 - temp storage allocation

29 - TMAP

30 - output field processor

31 - output var processor

32 - plot char processor

33 - output range analyze

34 - plot char default assignments

35 - output delimiter processor

36 - output run no. processor

37 - hash entry routine

38 - left of = symbol processor

39 - EXPND/Macro argument list processor

40 - pack

41 - unpack

42 - argument counter

43 - fixed pt. right justify numeric output

44 - floating pt. left justify numeric output

45 - integer numeric output routine

46 - object code builder

47 - left context analyzer

48 - right context analyzer

49 - numeric only context

50 - documentor

51 - level frnt

52 - output context

53 - Autoplot expander

54 - Variable alignment

Sub-outlines (continued page 3.)

55 - note card processor

56 - source/diagnostic listings

57 - output monitor - compile phase

58 - cross reference update

59 - title finish

60 - error printing

61 - options handler

62 - eqn ordering

63 - data buffer update

64 - execution-check

65 - execution- nocheck

66 - stato option processor

67 - symbol table listing processor

68 - cross reference option processor

69 - systems analysis routine

70 - output phase

71 - Print

72 - plot preliminaries

73 - Independent Variable Plot

74 - Time Plot

75 - loader

76 - symbol table tax sort

77 - characteristic computation

78 - Execution Time Output Monitor

79 - Plot Extremum Transfer - Default Checking

80 - Plot Extremum Transfer - Rounding

Sub-outlines(continued

81 - Plot Character Transfer

82 - Plot i.v. name processing

83 - Sort of data buffers

84 - Prints object code listing

85 - Optimizes object code

86 -

87 - not presently implemented

F. Loading NDTRAN2 on your computer

Regardless of the type of computer that you have, the NDTRAN2 interpreter will always be on FILE 1 of your magnetic tape and will always be named appropriately --i.e., STANDARD, RESEARCH PDP11 , etc. The overlay information will always be File 2. All of the data sets on magnetic tape will have the same characteristics(with one exception):

9 track 1600 bpi
EBCDIC or ASCII as requested.
LABELED or UNLABELED

DCB parameters will be:

Record Format	FB (fixed block)
Record Length	80 bytes
Block Size	800 bytes.

The IBM procedure for providing the link-edit of the source NDTRAN2 and placing the created load module is shown below, on page 17.

The PDP/11 link-edit materials are shown beginning on page 18

The program to load NDTRAN2 onto your (IBM) system is shown in the two pages below. The program will compile the source program, overlay the source code and place the load mondule onto your system. The program procedure will also execute all of the test programs on your created load module. After examining the implementation manual, the following lines should be examined:

40 tape setup:	NDTRN2,DISPTH	
280 source :	There are two versions of NDTRAN2, a standard version for smaller core sizes and a research version:	
	Standard version:	label=46 DSN=STANDARD
	Research version:	label=45 DSN=NDTRN2
320 overlay :	lines 320 - 370 modify the object module by adding or concatenating the overlay onto the object module. You may omit this if you do not wish the overlay.	
430 load module:	You may wish to modify this in order to locate the load module appropriately on your system.	

INTERMEDIATE FILE

NDTRAN2 requires a direct access disk file for operation. This is defined and shown on page 7 of this manual. It is accessed every time that a model is run. Please see page 4 of this manual for the definition and use of the intermediate file. The file is reference on line 190 of the procedure on page 18.

```
00010 //FNDTRN6 JOB (AF,E079,,30),537261259,W.DAVISSON,
00020 //  NOTIFY=FNDTRN,REGION=192K,TIME=10,MSGLEVEL=(1,1),MSGCLASS=A
00030 //*  THIS IS THE LINK FOR THE STANDARD  VERSION OF NDTRAN2
00040 /*SETUP       NDTRN2,DISPTH
00050 //NDTRAN    PROC SYSOUT=A
00060 //NDTRAN    EXEC PGM=NDTRAN2
00070 //STEPLIB   DD   DSN=FNDTRN.STANDARD.LOAD(NDTRAN2),DISP=SHR
00080 //FT01F001  DD   UNIT=DISK,SPACE=(CYL,(1,1))
00090 //FT02F001  DD   UNIT=DISK,SPACE=(CYL,(1,1))
00100 //FT03F001  DD   UNIT=DISK,SPACE=(CYL,(1,1))
00110 //FT05F001  DD   DDNAME=SYSIN
00120 //FT06F001  DD   SYSOUT=&SYSOUT
00130 //SYSIN     DD   DSN=FNDTRN.STANDARD.LOAD(NDTRAN2),DISP=SHR
00140 //SYSOUT    DD   SYSOUT=A
00150 // PEND
00160 //NDTEST     PROC NAME=,FILE=
00170 //NDTEST     EXEC PGM=NDTRAN2
00180 //STEPLIB    DD   DSN=FNDTRN.STANDARD.LOAD(NDTRAN2),DISP=SHR
00190 //FT01F001   DD   DISP=SHR,DSN=ACAD.ND2TRAN
00200 //FT02F001   DD   UNIT=DISK,SPACE=(CYL,(1,1))
00210 //FT03F001   DD   UNIT=DISK,SPACE=(CYL,(1,1))
00220 //FT05F001   DD   DSN=&NAME.,LABEL=&FILE.,UNIT=TAPE,
00230 //      DISP=(OLD,PASS),VOL=SER=NDTRN2
00240 //FT06F001   DD   SYSOUT=A
00250 // PEND
00260 //FORT     EXEC PGM=IEYFORT
00270 //SYSPRINT DD   SYSOUT=A
00280 //SYSIN    DD   UNIT=TAPE,VOL=SER=NDTRN2,
00290 //           DISP=(OLD,PASS),LABEL=(46,SL),DSN=STANDARD
00300 //SYSLIN   DD   DSN=&&OBJECT,DISP=(,PASS),UNIT=DISK,
00310 //           DCB=BLKSIZE=80,SPACE=(80,(2000,1000),RLSE)
00320 //           EXEC PGM=IEBGENER
00330 //SYSIN    DD   DUMMY
00340 //SYSPRINT DD   SYSOUT=A
00350 //SYSUT1   DD   UNIT=TAPE,DSN=LINK2,VOL=SER=NDTRN2,
00360 //           LABEL=(44,SL),DISP=(OLD,PASS)
00370 //SYSUT2   DD   DSN=&&OBJECT,DISP=(MOD,PASS)
00380 //LKED     EXEC PGM=IEWL,PARM='XCAL,XREF,OVLY,MAP'
00390 //SYSPRINT DD   SYSOUT=A
00400 //SYSUT1   DD   SPACE=(1024,(100,10)),UNIT=DISK,DCB=BLKSIZE=1024
00410 //SYSLIB   DD   DSN=SYS1.FORTLIB,DISP=SHR
00420 //SYSLIN   DD   DSN=&&OBJECT,DISP=(OLD,DELETE)
00430 //SYSLMOD  DD   DSN=FNDTRN.STANDARD.LOAD(NDTRAN2),DISP=(,CATLG),
00440 //           UNIT=DISK,VOL=REF=FNDTRN,SPACE=(TRK,(20,1,1),RLSE),
00450 //           DCB=BLKSIZE=19069
```

```
00460 //      EXEC NDTEST,NAME=CLIFF,FILE=1
00470 //      EXEC NDTEST,NAME=XERR1,FILE=2
00480 //      EXEC NDTEST,NAME=CLIF,FILE=3
00490 //      EXEC NDTEST,NAME=COAL1,FILE=4
00500 //      EXEC NDTEST,NAME=PLOT5,FILE=5
00510 //      EXEC NDTEST,NAME=CONCARD,FILE=6
00520 //      EXEC NDTEST,NAME=DEL,FILE=7
00530 //      EXEC NDTEST,NAME=DELRAMP,FILE=8
00540 //      EXEC NDTEST,NAME=ERRORS,FILE=9
00550 //      EXEC NDTEST,NAME=FUNCT,FILE=10
00560 //      EXEC NDTEST,NAME=INTRST,FILE=11
00570 //      EXEC NDTEST,NAME=MACRO,FILE=12
00580 //      EXEC NDTEST,NAME=STEPDLS,FILE=13
00590 //      EXEC NDTEST,NAME=SMOOTH,FILE=14
00600 //      EXEC NDTEST,NAME=TABLES,FILE=15
00610 //      EXEC NDTEST,NAME=CLIFF2,FILE=16
00620 //      EXEC NDTEST,NAME=COAL2,FILE=17
00630 //      EXEC NDTEST,NAME=EXPTST,FILE=18
00640 //      EXEC NDTEST,NAME=INTGRT1,FILE=19
00650 //      EXEC NDTEST,NAME=INTGRT2,FILE=20
00660 //      EXEC NDTEST,NAME=INTGRT3,FILE=21
00670 //      EXEC NDTEST,NAME=LAKE,FILE=22
00680 //      EXEC NDTEST,NAME=PLOT2,FILE=23
00690 //      EXEC NDTEST,NAME=PLOT3,FILE=24
00700 //      EXEC NDTEST,NAME=PLOT4,FILE=25
00710 //      EXEC NDTEST,NAME=PLOT6,FILE=26
00720 //      EXEC NDTEST,NAME=POISSON,FILE=27
00730 //      EXEC NDTEST,NAME=PULSE,FILE=28
00740 //      EXEC NDTEST,NAME=SPIRAL,FILE=29
00750 //      EXEC NDTEST,NAME=STEP,FILE=30
00760 //      EXEC NDTEST,NAME=PLOTSTAT,FILE=31
00770 //      EXEC NDTEST,NAME=TABLE,FILE=32
00780 //      EXEC NDTEST,NAME=TABLE1,FILE=33
00790 //      EXEC NDTEST,NAME=XERR2,FILE=34
00800 //      EXEC NDTEST,NAME=XERR3,FILE=35
00810 //      EXEC NDTEST,NAME=XERR4,FILE=36
00820 //      EXEC NDTEST,NAME=XERR6,FILE=37
00830 //      EXEC NDTEST,NAME=HEAT,FILE=38
00840 //      EXEC NDTEST,NAME=WORLD3,FILE=39
00850 //      EXEC NDTEST,NAME=AGRIC,FILE=40
00860 //      EXEC NDTEST,NAME=CAPITAL,FILE=41
00870 //      EXEC NDTEST,NAME=RES,FILE=42
00880 //      EXEC NDTEST,NAME=POP,FILE=43
00890 //
```

IBM 370 systems with DOS

If you are planning to load NDTRAN2 onto a system with DOS, the following information will supplement the information provided for the OS installation.

1. Be sure that the INPUT/OUTPUT unit numbers are properly set.

2. It may be necessary to clear the disk before each run. Apparently under DOS systems, the FORTRAN scratch areas are used by other systems (COBOL). Some COBOL programs leave "END-OF-FILE" marks which cannot be read by FORTRAN.It may be necessary to institute a clear dksi procedure or command as part of the NDTRAN2 procedure.

3. In some instances that we have been able to identify the DOS system uses disk files strangely (to us). Our DEFINE FILE statements specified records of 80 words. UNDER DOS they may have to be 320 words in length. Some implementations noted that with the 80 word length only one-quarter of the record being used was stored, and the rest was lost. By changing the DEFINE FILE card to 320 words, the system operated correctly.

A PRIMER FOR NDTRAN:

A Continuous System Interpreter

William I. Davisson

John J. Uhran Jr.

University of Notre Dame
Notre Dame, Indiana
1979

PREFACE

NDTRAN is a continuous and stochastic system modelling interpreter designed for use on small computer systems. We originally had as a goal a system of the size and power of the PDP 11/40 with a 28K FORTRAN partition. NDTRAN is written in ANSI FORTRAN and is presently used on approximately 130 computers at colleges and universities around the world. The work was made possible by a grant from the Fleischmann Foundation.

The project was directed by William I. Davisson and John J. Uhran Jr., both of the University of Notre Dame. All of the programming was done over a four year period by undergraduates at the University of Notre Dame. The system was designed and implemented by Daniel A. Poydence presently with Texas Instruments, Dallas Texas. Other undergraduates who worked on the project were Thomas Everman, Gary Pelkey and Timothy Malloy. Robert Konicek, an undergraduate in the Electrical Engineering Department is presently working on maintenance. Daniel Poydence, Timothy Malloy and Robert Konicek are working with the Directors in maintenance of the final version of the project.

NDTRAN was originally released in 1977. A revised and final version , called NDTRAN2, has been completed and is released as of January 1980. Maintenance will be maintained on NDTRAN2. NDTRAN2 is designed to the same specifications as the original NDTRAN but is several times faster in execution with a number of program enhancements.

This manual is written primarily for NDTRAN2, the final version of the NDTRAN project. In most instances the syntax of the two versions is identical. Where there is a difference the information in the manual text is for NDTRAN2. The information for the first version called NDTRAN is shown in the box areas as appropriate.

> Information for NDTRAN version 1 is shown in the box areas where syntax or procedures of NDTRAN differ from syntax and procedures for NDTRAN2.

NDTRAN contains a large number of syntax error messages that appear in the following form:

```
21  PLOT POS2=2//POS1=1
                 $   $

    2) ***** C R I T I C A L *****          ND0502
    1) ***** C R I T I C A L *****          ND0516
```

The $ sign under a location in the equation line indicates that at that point an error was detected. The error may well have been made prior to that point but NDTRAN detected the error at that indicated point. The error messages are sequentially numbered from left to right. The leftmost $ sign is number 1 and so on. The error messages themselves are numbered and the type of error is identified. Each error message number may be located in the appropriate appendix where the message explains the error.

The messages are different for NDTRAN and NDTRAN2.

NDTRAN2 error messages	APPENDIX A
NDTRAN error messages	APPENDIX B

EXECUTION TIME ERROR MESSAGES AND EXPLANATIONS ARE LOCATED IN APPENDIX C.

SYSTEM ERRORS ARE SHOWN IN APPENDIX D.

GRAPH OF A SPIRAL

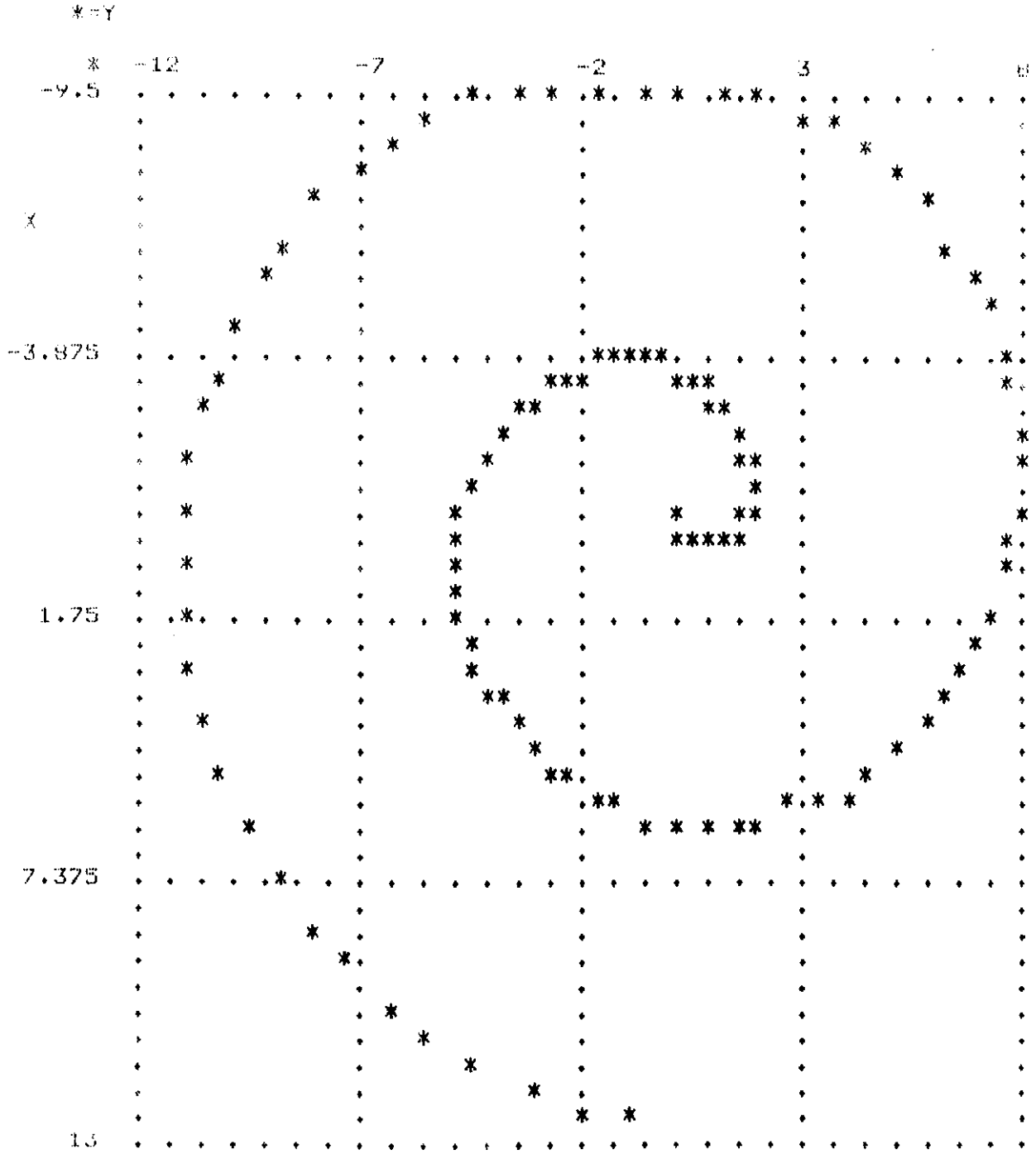


TABLE OF CONTENTS

Chapter I
System, Model and Simulation

A) Introduction

Modeling and simulation may be considered a general method for investigating certain types of problems. However, models can only be used with respect to a system. Thus, let us define all three.

> A System is some part of the real world independent of size or function. Examples are amoeba, a criminal court, a river, an automobile, an automobile accident, a satellite, a city, etc. Anything may qualify for a system. While vague, it is appropriate and all inclusive. A more precise theoretical definition of a system is the collection of distinct entities which are all inter-connected and behave in some goal-seeking way.

There are two approaches to thinking about a system. One may describe a system by thinking about its function--that is, what it is for. For example, a building may be considered as a physical method of protecting man and his activities from the environment. Furthermore, one might wish to consider the types of activities that could take place within the physical limits of the building (system). In short, when

considering the function of the system one considers the function of the system in relation to man.

Alternately, one might think of a system as a structure of substructures in which case the entities of the building and their relationships create the system. This might include a metal frame system, an electrical system, an air conditioning system and the like. The sum total of these separate systems are called sub-systems in that they are subordinate to the overall building. It is important to remember that each of the sub-systems are themselves systems. The building, the system under consideration, is a subsystem of a larger system, for instance, a city.

> Models. Scientists model when the real world system is too complex to be understood by simple observation, or when manipulation and policy direction cannot be directed toward the system itself. If we can observe the system and understand what has happened, then either a specific model is not needed or the intuitive mental model is sufficient. However, if the system is not able to be understood by simple observation, then a specific and precisely defined model might be necessary.

A model may be either a logical or physical construct of the real world system, usually simplified to emphasize certain desired relationships or goals.

There is one key idea concerning models. Models do not stand alone. A model is uniquely related to some primary system. The purpose of a

model is to give insight into or tell us something about the primary system of concern.

Since the idea of a model is not divorced from that of a system, we might indicate the following steps that characterize the methodology of systems science--i.e., the modeling and simulation of primary systems:

1. Identify the objectives, purposes or functions of the system;
2. Identify the system structure;
3. Identify measures of system performance in relation to its intended purpose;
4. Develop a model (logical construct) of the system in light of the goals or purpose of the modeler;
5. Develop the simulation from the model;
6. Compare the simulation performance with the measured performance of the system;
7. Select and alter the simulation as necessary;
8. Determine model validity.

Simulation is what ultimately validates the modeling process. It is a method of converting the model from a logical construct to a phenomenon which will give answers. A common way of thinking of a simulation is that it is a computer program (programmed from the model) that will provide numeric answers. Thus the term computer simulation is often used. The algorithm

for writing the computer program (simulation) is the logical relationships expressed in the model. But it should be noted in passing that simulations can be accomplished without algorithms, quantitive data and computers. The boy who builds and sails his ship on a lake has accomplished these same objectives.

B) Systems Science Simulation

The modeling and simulation approach to problem solving seems to follow four distinct steps. The chronological order of these steps takes one from the generalized to the specific methodology.

1. Scientific Method - a methodology that involves the establishment of a hypothesis and the test of its validity. The traditional methods of testing the hypothesis often give way to more recent methods of simulation, when a direct test of the hypothesis is not possible. An example would be the testing of a hypothesis concerning any system where it is not possible to have access to the system directly.
2. Causal Loop Diagram - The purpose of a causal loop diagram is to show the relationship among the

variables that constitute the model of the system.

3. Flow diagram - the flow diagram represents the structure of the system as defined using the specific symbols of some problem solving methodology. It involves translating a rather generalized causal loop diagram into a specific form.
4. Simulation - translation of the flow diagram into a set of mathematical equations to simulate the system with the special use of a specialized computer language. The one we shall use in this text is called NDTRAN, meaning Notre Dame Translater.

The first two points constitute a generalized methodology applicable to any modeling approach. However, with the use of the flow diagram one narrows the focus and becomes more method dependent, but not necessarily language dependent. For instance , if one were to adopt a form of System Dynamics * modeling and simulation; then steps three and four would become method dependent but one could undertake the computer simulation in any of a number of computer simulation languages including NDTRAN, DYNAMO, DYSMAP, SIMSCRIPT, GPSS, and even FORTRAN.

This manual will not generalize a discussion of modeling and simulation, but will undertake to discuss the nature and uses of a dynamic simulation interpreter called NDTRAN. Reference is made to aspects of System Dynamics * as relevant for the discussion.

---

* as discussed by J.W. Forrester in his INDUSTRIAL DYNAMICS and Dennis Meadows in his DYNAMICS OF GROWTH IN A FINITE WORLD.

Chapter 2

Causal Loop Diagrams

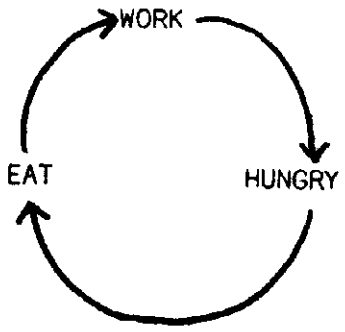
A) Introduction

The purpose of the causal loop diagram is to show the relationships between the variables selected to be used in a model. This cause and effect relation is referred to by many modelers as influence and we shall refer to it here in the same way. In a causal loop diagram (logically) each condition occurs as the result of a previous condition. In short, if we are to understand the behavior of the rabbit population (for example) we must understand the causes of that behavior. Thus, we must understand why the rabbit population behaves as it does.

B) Simple examples

Let us take an example of a man cutting wood with a chainsaw. As he works he gets hungry. When he gets hungry he eats something. After he eats he can again work for a while. A causal loop diagram representing the man with the chainsaw working is shown in Figure 2.1.

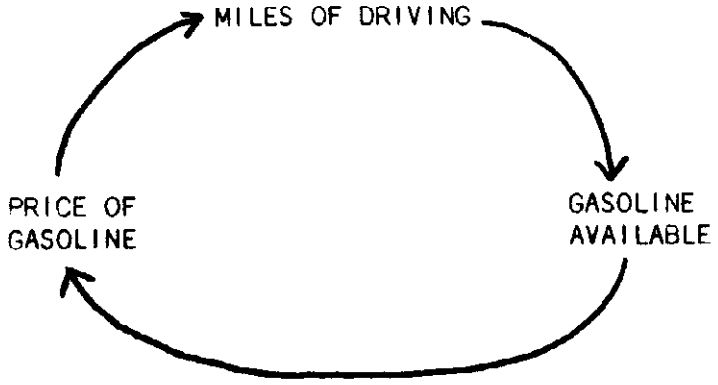
Figure 2.1
Causal Loop Diagram



Notice that the causal loop diagram indicates only two things: First, the direction of the action involved; and second, the influence which exists. That is, the work done causes the man to become hungry, and being hungry causes him to eat, and so on.

Another example that is familiar to everyone since the oil embargo is that depicted in Figure 2.2. The same circular influences occur and no external forces are involved.

Figure 2.2
Causal Loop Diagram



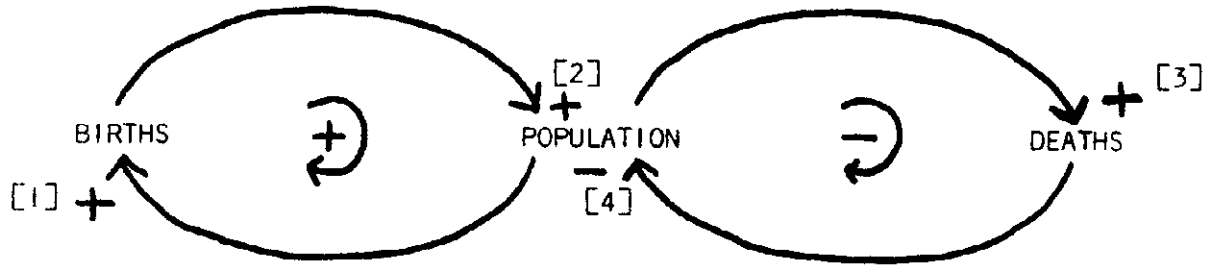
Referring to Figures 2.1 and 2.2 above, let us ask a couple of questions concerning the two causal loop diagrams shown. What are the goals of each of the systems? That is, what is the behavior that is being reflected by each model? What assumptions are necessary in order to define the structure of the model as depicted in the figures? Can you understand and communicate a finer structure of the model shown in Figure 2.1 and 2.2?

C) Additional Detail

With these preliminary notions completed, let us investigate some details of causal loop structures. To begin with,let us examine a generalized causal loop diagram of the size of any population, reduced to its simplest elements. These elements would be the size of the population itself, the number of births per period and the number of deaths per period. In this way we can focus upon the goal of the model and the specific nature of the auxiliary influences on

the goal. The goal of the given model is to examine (predict) the behavior of the population itself. The causal loop diagram (model) is shown in Figure 2.3

Figure 2.3
Causal Loop Diagram



We have added certain elements to the causal loop diagram in Figure 2.3 in that we are more explicit about the nature of the assumptions made concerning the model. These assumptions are indicated by the + and - (respectively, plus and minus signs) which show not only the direction of the influence but also the nature. A "+" sign means a direct influence; as births increase so does the population, and vice-versa. A "-" sign means an inverse influence--i.e., as deaths (per period) increase, the population will tend to decrease. An increase in the deaths per period will not per se cause the population to decrease. However, as the deaths per period increase (unless offset by increased births per period) the population would tend to decrease. An odd number of minus signs in a causal loop diagram means that

the loop itself is characterized by inverse causation. If the number of "-" signs is even, or if there are only "+" signs, then the loop itself is characterized by direct causation or is increasing. These ideas are important for evaluating the model behavior.

Let us look at the causal loop diagram. The first assumption is that the size of the population contributes to the births per period, indicated by the "+" sign labeled /1/ in Figure 2.3 The more births per year the larger will be the population /2/. The larger the population, the more deaths there will be per period /3/, and the more deaths per year the greater will be a negative influence on population /4/. The population itself may still continue growing, for other reasons, but the impact of the death loop itself is to decrease the population, or to slow its rate of growth, or to cause it to decline.

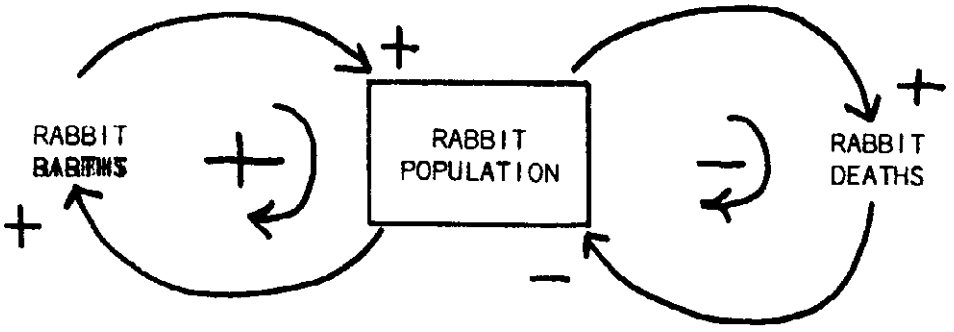
The nature of the birth loop itself is that the loop response to a stimulus (such as increasing the number of births per period or an increase in the population) will be a continued increase in the population variable. In the death loop any increase in the population will contribute to an increase in the number of deaths per period, but this will tend to limit the growth of the population. The death loop result would (by itself) tend to reduce the size of the population. If the birth loop were stronger than the death loop, then the death loop could only tend to inhibit the rate of growth of the population. (Be sure that you can follow the influences and discussion suggested for the population loop.)

D) Complex Example

1. Concept of Level and Rate

As a starting point, consider a causal loop diagram of a population of rabbits on a given island or plateau. The main food is either a natural or man-grown grass or a leafy product, and the major predator is a family of red foxes. Possible variables would then be, the rabbit population, food per rabbit, arable land required, rabbit density, fox population, rabbits killed per month by the foxes. These variables are subsumed in Figure 2.4. Rabbit births and deaths would include the effects of food and foxes on the rabbit population.

Figure 2.4
Causal Loop Diagram



The diagram is very similar to that shown in Figure 2.3 except that it identifies the specific goals of the model. The rectangular symbol (often omitted) in the center of the diagram indicates that the basic purpose of the model is to consider the behavior of the rabbit population. In addition, the population is a variable which measures some quantity or value at every and any period of time. It is (measures) the state of the system and is called a level. This brings out an important point; the level variables in any model should suggest the goals of the model. The nature of the causal loop diagram shown indicates that the goal of this model is the behavior of the rabbit population (not the fox population, or the lettuce supply, or other). Equally important to the behavior of the model are the other variables shown. These represent rates. These rates (in this model) represent the number of rabbits born or dying per unit of time in the model.

The loops themselves represent feedback from the level variables to the rate or decision variables. For instance, assuming a given fertility rate for rabbits and given a constant percentage of female rabbits (to the total rabbit population) the higher the population of rabbits, the more baby rabbits will be born per period of time. The feedback relationship (positive or negative) is shown by the sign and the arrows.

2. Additional Complexities

The basic assumptions to the model shown in Figure 2.4 are:

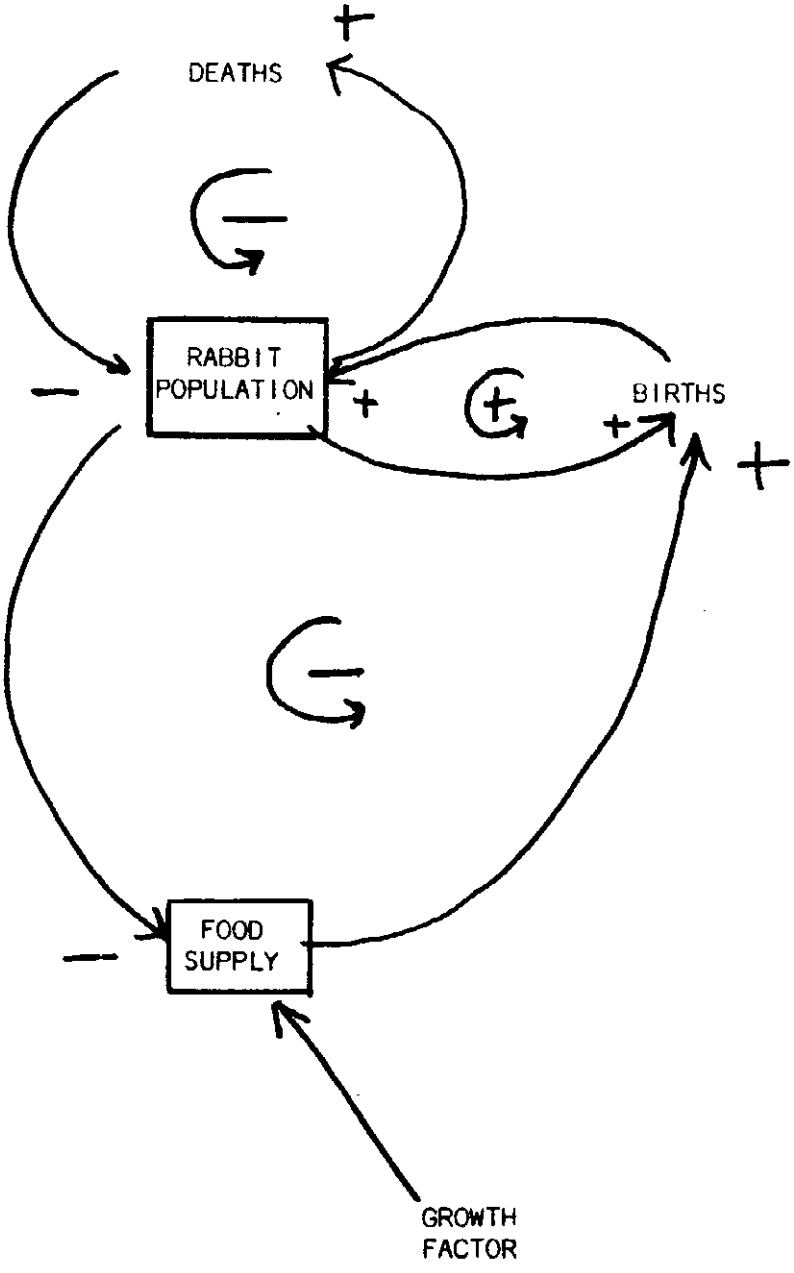
1. The level of the rabbit population is critically important to achieving the goal of the model;
2. All death influences are shown in the death rate including fox predatation on the rabbits, rabbit demise, shortage of rabbit food supply, and so on; and
3. All birth influences are shown in the birth rate including excessive rabbit food supply, fertility rate of rabbits, percentage of rabbit females to total rabbits and the like.

The next question is, what happens if the goal of the model is not adequately served by a single level? Under what conditions might it be necessary to model additional specific level variables excluding the population of foxes. As yet the foxes do not constitute a problem, perhaps because they are not yet in the area, or there is other fox food available that is easier for the foxes to catch. In any event the attainment of the goals of the model make it necessary to have a model containing more than one level variable. Figure 2.5 is a case in point.

This is a causal loop diagram that is a two level model, one of the rabbit population and one of the rabbit food supply. The obvious assumption at this point is that the natural survival conditions for rabbits is more complex than can be handled with a single level variable. Specifically, there might be a critical balance between the food supply and the rabbit population. The assumptions of the model shown in Figure 2.5 are:

1. The level of rabbit food is critically important in determining the level of rabbits;
2. All death influences are included in the death rate;
3. All birth influences are shown in the birth rate;
4. The food supply is the dominant effect in influencing the birth rate. All other effects on birth are therefore subordinate to the food supply.

Figure 2.5
Two level model:
Causal loop diagram



Let us examine Figure 2.5 to determine how it differs from the model shown in Figure 2.4. The goal of the model has not changed, but the structure of the model has changed. The structure of the model consists of the relationships of the variables within the model. The major difference between the two models is the manner in which the births per period are controlled. In short the set of influences that control the number of rabbits born per period has changed, and therefore the structure of the model has changed. At this point we have established the causal loop diagram for a two level rabbit model, where the rabbit population and the rabbit food supply are the critical level variables. But further structural change is required since foxes represent a critical concern in the problem statement. With the addition of foxes we at least have the minimum requirements for a complete model which satisfies the goal.

The modeling of the foxes can be thought to be similar to that of the rabbits except for some influences. Thus we will create a sub-system for the foxes similar to that for the rabbits, which therefore will create a larger closed loop model including a sub-model for rabbits and a sub-model for foxes as shown in Figure 2.6

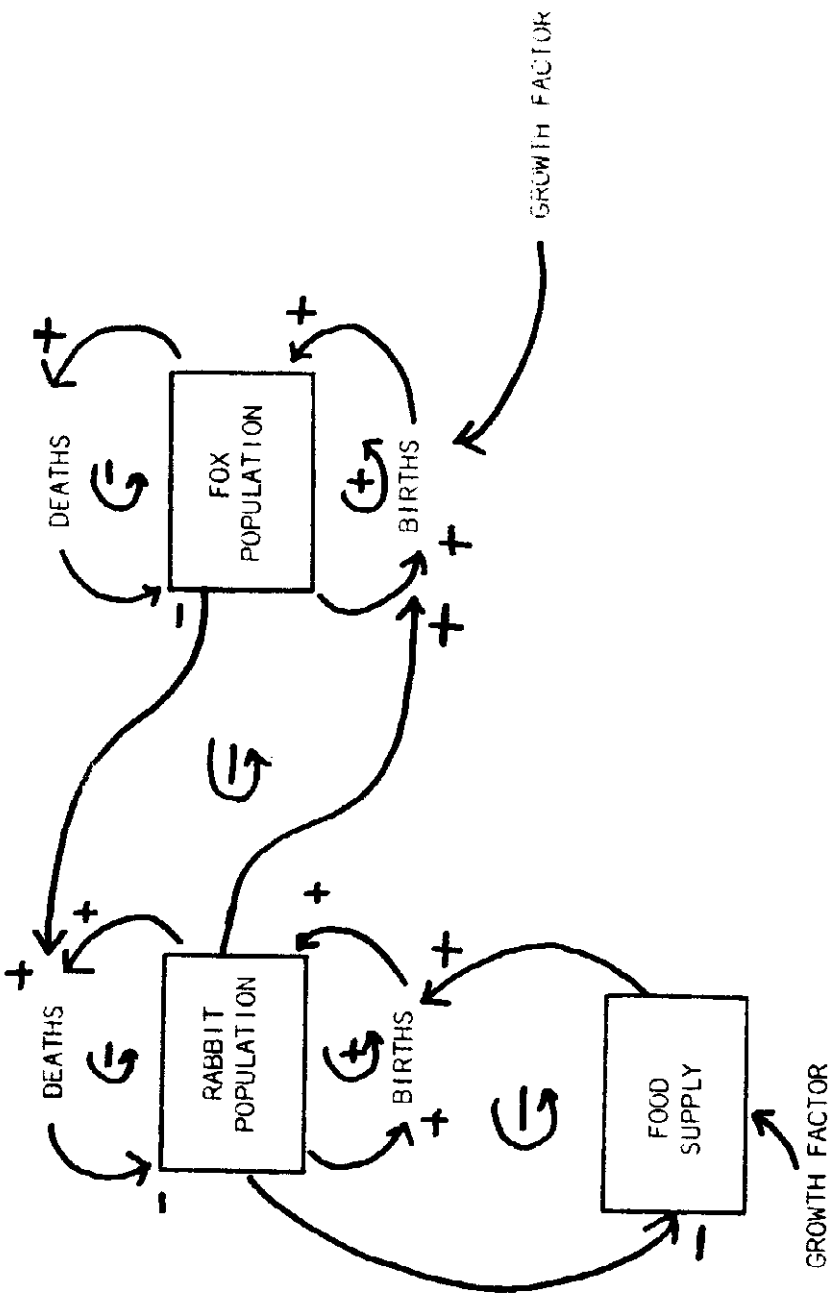


Figure 2.6
Causal loop diagram:
Fox and rabbit subsectors

The left side of Figure 2.6 is the same as that shown for Figure 2.5. In this instance the level of the rabbit food supply is seen as having a critical influence on the number of births per unit of time of the rabbits. The fox population is seen as a separate sub-model.

There are two contacts between the separate sub-models.

First, the rabbit population constitutes a primary food for the foxes. Therefore the level of the rabbit population is an input to the births per period (birth rate) of the foxes. The relationship is positive, in that the larger the rabbit population, the larger is the food supply of the foxes, the larger is the fox birth rate. The model clearly assumes that the food supply of the foxes is the most important element of the fox birth rate. All other influences on the birth rate of the foxes either comes from the size of the fox population itself, or is assumed in the growth constant of the birth rate.

Some of the other elements that influence the birth rate of either foxes or rabbits, other than these noted, would be the percentage of the population that was female, the fertility rate and related phenomena

The second contact point involves the size of the fox population which will have a direct influence on the death rate of the rabbits. The larger the fox population, the larger will be the number of deaths per period for the rabbits. Other factors that might influence the number of deaths of the rabbit population besides the population itself, would be such things as abnormal conditions not included in the food supply, the fox population, etc. The last causal loop diagram, Figure 2.6, shows

immediately that the structure of the model has changed. The important point to note from a structural point of view is that the influences on the death and birth rates have changed. The modeling is accomplished by controlling (modelling) the rates. Thus the structure of the model is identified by specifically ascertaining the way in which the rates are controlled.

E) Summary

A causal loop diagram has two major parts:

1). variables - the rates and levels for the model;

2). links or flows -- directional polarized lines which indicate the general relationships between variables of the model.

Causal loop diagrams also help to establish the structure of a model and determine what the goals are and how they are to be achieved.

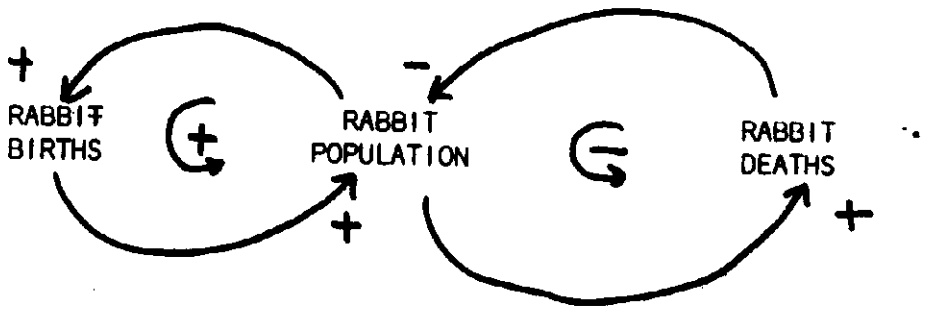
Chapter 3
Flow Diagrams and NDTRAN

Now we wish to introduce the reader to the System Dynamics Flow Diagram and the continuous systems computer interpreter:NDTRAN. Remember that System Dynamics programs represent models of continuous time, dynamic systems and, under special conditions, discrete time systems.

A) Flow Diagrams

In the previous chapter, a simple rabbit population model was shown in causal loop diagram form. A single level variable for the rabbit population and two rate or decision variables ; the birth rate (rabbits born per period) and the death rate (rabbits dying per period). The causal loop diagram is shown as Figure 3.1

Figure 3.1
Causal Loop Diagram: Rabbit Population



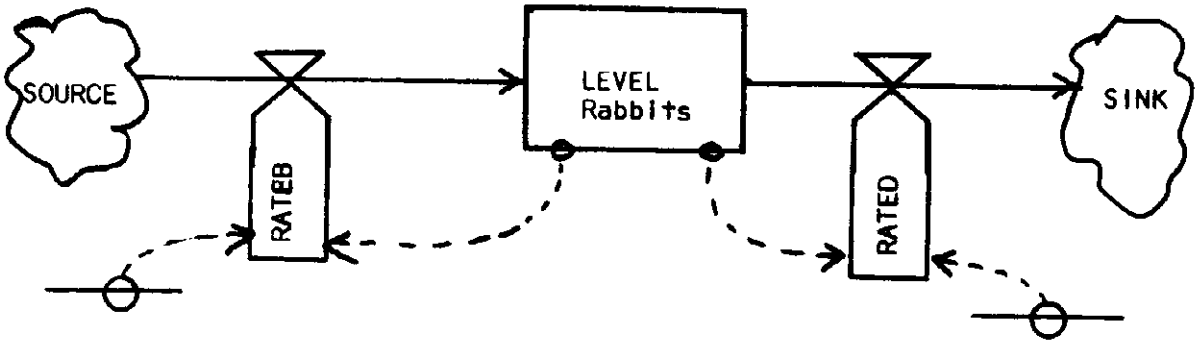
Since System Dynamics Simulation methodology is a type of problem solving methodology, perhaps not the easiest but the best way to explain the nature of System Dynamics Simulation Language is to do so in the context of simple, but specific problems. The approach will be as follows:

1) The Causal Loop Diagram will be translated into a System Dynamics Flow Diagram;
2) The System Dynamics Flow Diagram will be translated into a computer model;
3) Each computer language statement will be explained;
4) the manner in which the program is executed will be explained.

The purpose of the causal loop diagram is to indicate the basic structure of the problem and the basic influences that cause the model to behave as it is hypothesized. The Flow Diagram translates these basic ideas into the "flows" structure of System Dynamics modeling, establishes the specific boundaries, and indicates the nature of the feedback in the model. Figure 3.2 is the Flow Diagram counterpart of the Causal Loop Diagram shown in Figure 3.1 It is necessary to remember that the basic System Dynamics Modeling approach is that of conserved flows. By this we mean that materials or things cannot be created or destroyed but simply moved from one place to another. The conserved flow approach considers that the source is an infinite population of whatever flows into the level. The magnitude of the flows into the level is controlled by the

policy decision (valve symbol) between the source and the level. Thus, rabbits are the only conserved flow in Figure 3.2. Any other flows in a system are <u>unconserved</u> and relate to information. These in and of themselves cannot control the level of any material or any things in the system.

Figure 3.2
Flow Diagram:
Rabbit Population



Flows of people, materials or services

Information flows: basis for feedback

RATEB Rabbits born per period

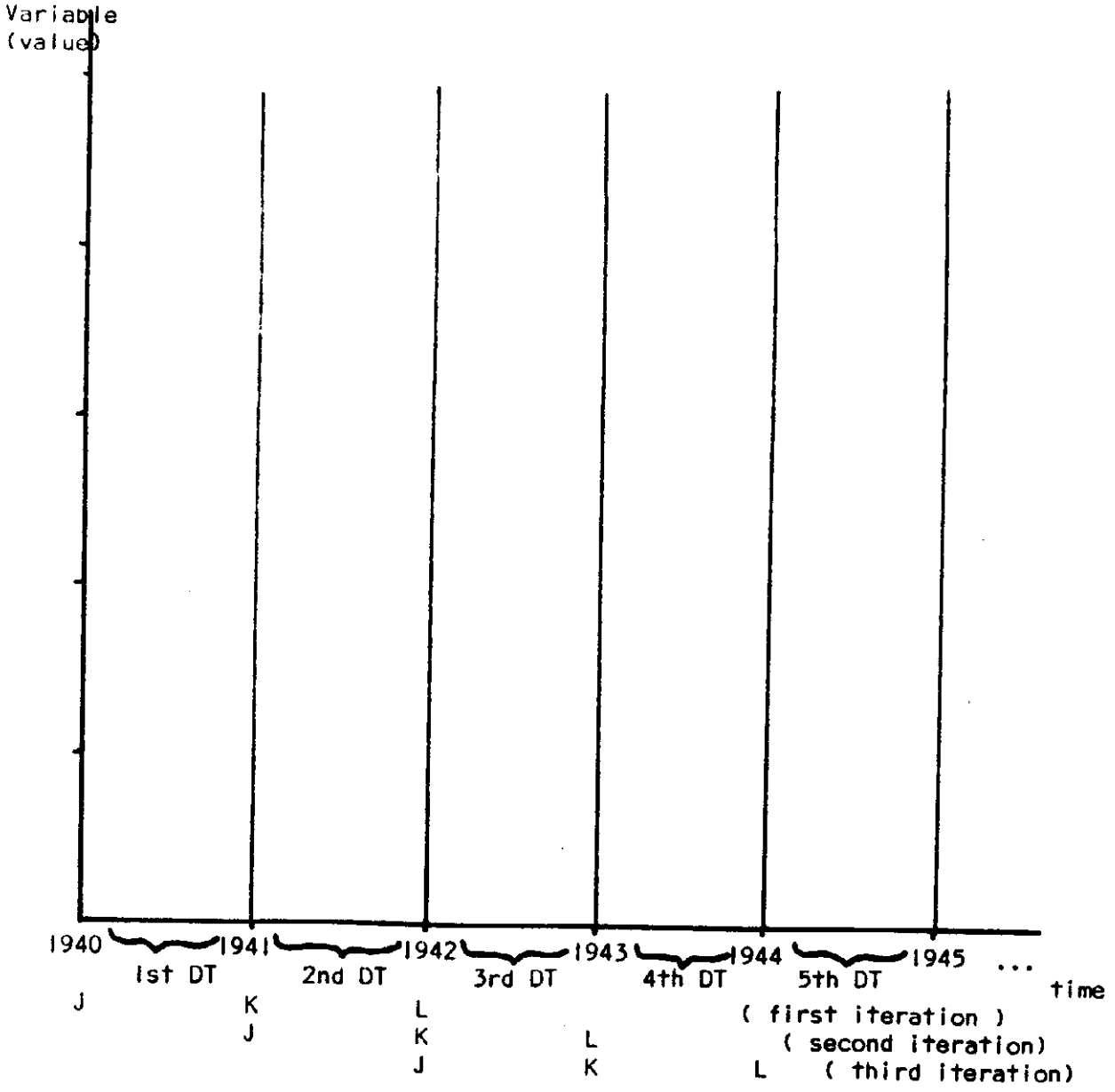
RATED Rabbits died per period

CONSTANT: a value that converts the size of a population to the rate at which it would accomulate over time.

B) Time Data and Subscripting

Systems modeled with Systems Dynamics are always time dependent systems. Thus, the time frame of the system is always shown on the horizontal axis or the abscissa of a graph and the value of one or more system variables is shown on the vertical axis or the ordinate. The period of the model is referred to as the number of iterations in the computer run. Therefore a run period for a simulation is defined as the difference between the time the run ends (STOP) and the time that it begins (START). The length of time between the two is divided into equal time intervals. Each time interval or time segment is called a "solution interval" and is more commmonly referred to as DT. The continous advance of time is thus divided into small intervals of equal length, each of which is one DT (Figure 3.3). By definition, this interval DT must be short enough so that we may assume that constant rates of flow over the interval are a satisfactory approximation to the continously varying rates in the real system /Forrester, J.W., INDUSTRIAL DYNAMICS, pp. 73-74/. A solution interval or DT must be less than the time constant of the individual system. Normally, if one is dealing with a model that has a number of different time constants, then the DT or solution interval should be less than the smallest time constant. A good initial rule of thumb is that the DT period should be less than one-half of the smallest time constant of the system. Figure 3.3 suggests the framework for understanding the nature of the method of execution of the program over time.

Figure 3.3
Systems Dynamics Program
Execution



Start run in the year 1940 and DT = 1: therefore
1940 + DT = 1941
1941 + DT = 1942

Shown is the time scale from 1940-1950 with DT=1 year. Thus the period of the simulation is 10 years divided into 10 equal parts of 1 year each.

For convenience of calculations by the programmer, a specific user convention has been adopted for the subscripting of variables. There are two types of variables that this specifically applies to: rate variables subscripted as RATE.JK or RATE.KL and level variables subscripted as LEVEL.K . In Figure 3.3 the simulation is considered to begin in the year 1940. Thus any value for a variable as of the start of the run (1940) would be the initial value of the variable for the year 1940. The program would either assign or compute an initial value for a variable at the start of a run. The simulation would begin with this assignment of an initial value to the first time point in the run.

System Dynamics defines the subscripts used as follows:

J = immediate past time instant

K = present time instant

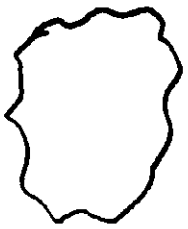
L = immediate future time instant

All calculations are done at time instant K or for time period K to L (KL) . Once every variable has been accounted for, a shift of the subscripts occurs, as from the first iteration to the second iteration in Figure 3.3 This process is carried out until all intervals have been accounted for.

C) Symbol Definitions

While giving an explanation for each symbol in Figure 3.2, we will also try to explain the algorithmic implementation in NDTRAN.

1. Source and Sink

The cloud formation  represents either a source or a sink for some specific accumulation. The best way to define the source or the sink is that it is analogous to a statistically infinite population that, no matter how much is drawn out of it or put into it, there is no impact on the scope or nature of the source or sink. The source or sink has no effect on the overall model, per se.

A Source (or Sink) is always immediately related to a level variable.

2. Level

The System Dynamic symbol for a level is: 

DEFINITION: A level variable is a time varying quantity whose present value is dependent on its past value plus (or minus) any changes that affect the variable in the immediate past.

INTERPRETATION: The number of rabbits in existence today equals the number of rabbits living as of the end of yesterday (J) plus the number born during the time interval (JK).

> The time periods are subscripted as before. The subscript combination JK refers to the time passage between yesterday and today, that is between the last discrete time instant and the present discrete time instant. The subscript combination KL refers to the time passage between today and tomorrow, that is between the present discrete time instant and the next immediate discrete time instant.

The level variable in a System Dynamics model indicates that the process of accumulation or integration is taking place. The new value of the level is obtained by adding to or by subtracting from the previous value, the changes that have occurred during the period. The net change in a level is the rate of accumulation over the past period less the rate of loss (gain) from the level. There is a more detailed discussion of this in the chapter dealing with integration methods in NDTRAN. The important thing to remember in the System Dynamics simulation methodology is that in a flow diagram, every time that a level symbol is seen, it means that the integration or accumulation process is required, or that a level equation is

required at that point in the computer model. There are a couple of different ways of illustrating a level equation in a flow diagram, but the shape is always the same. A variable name may appear in the upper left hand corner which would identify that level in the computer program. A name may appear in the middle of the level symbol which provides the vernacular definition for the variable.

For ease of communicating the model to others, for technical correctness and for ease of cross-reference by the programmer or modeler, it is always a good idea to use the symbols correctly in a System Dynamics Flow Diagram. This simplifies the translation of the model to the computer program.

3. Rates

Now, by convention (as well as logically) a System Dynamics level variable cannot change itself. In the rabbit model it takes spontaneous agreement between a male and a female rabbit before that process can occur. Rabbits, like Queen Victoria, did not spring fully clothed from the head of a large cabbage. This agreement shall be called a policy decision or a rate decision or simply a rate.

We have noted before that a critical area of modeling in System Dynamics is the necessity to identify and to model the rate (policy decision) equations. In the model shown in Figure 3.2, there are two rates that control the accumulations in the level variable; the birth

rate and the death rate. The rate symbol is defined as follows:

DEFINITION: A rate may be regarded as a policy decision that will affect a level equation over time. Rates are the amount of change per period that a level will undergo as a result of the action of a specific rate shown. Rates are related to the derivatives of the levels they effect. A rate(s) as a policy decision, will increase or decrease a level.

INTERPRETATION: The rate at which rabbits will be born (or die) is determined by the number of rabbits, times the fertility of the rabbits. The concept includes the idea that not all rabbits are female (or male) rabbits.

As seen in Figure 3.2 the two System Dynamics concepts needed to define a rate were:

1) a level variable;
2) a constant, that converted the size of the population to a rate at which that size population would accumulate over time.

These are, however, not necessary to define a rate in all cases.

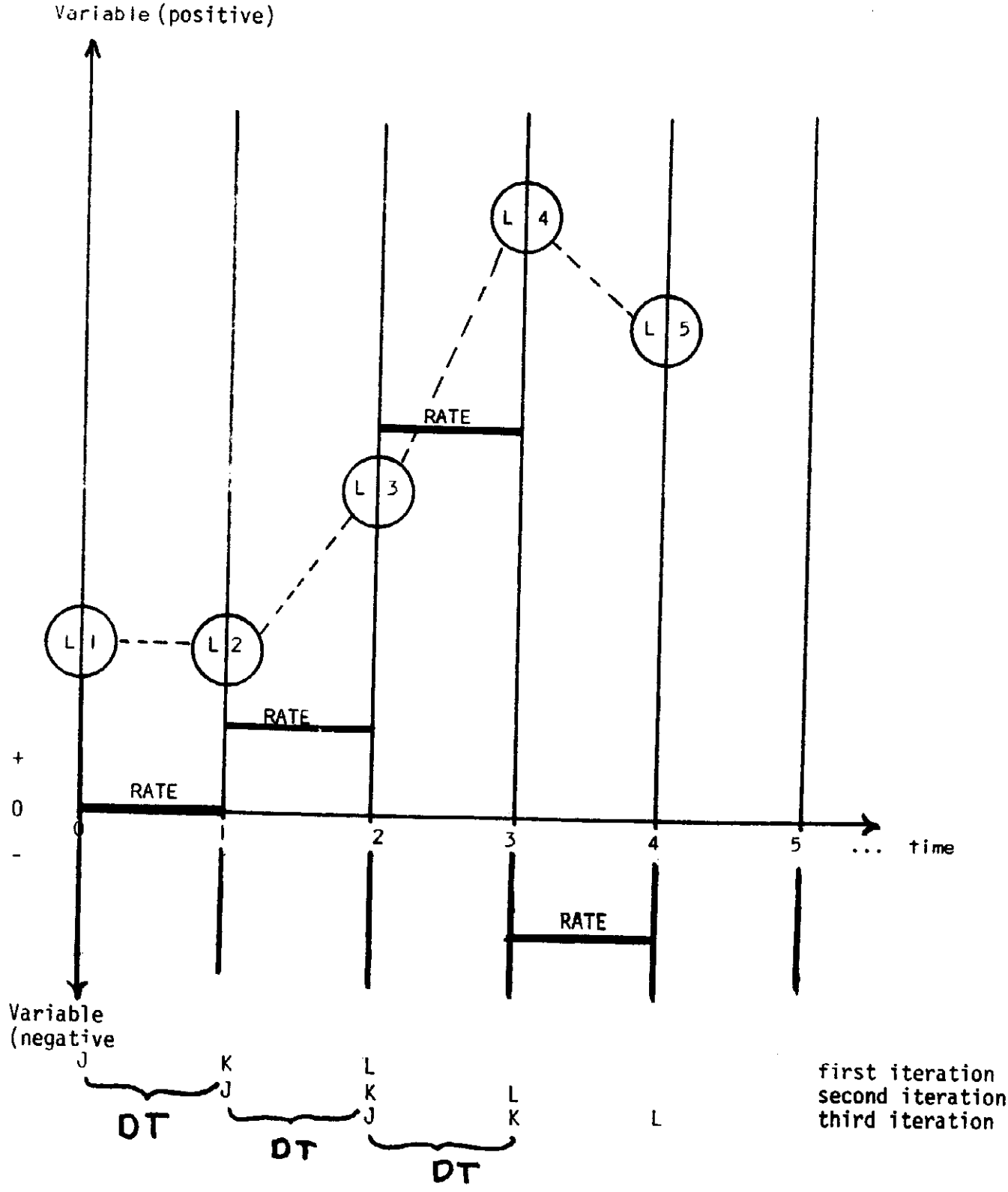
Because rates are a measure of the change of a level variable, they are calculated over an interval DT. Thus we find in System Dynamics programs,rates are always defined with reference to a time period such as RATE.JK or RATE.KL. This represents the magnitude of the rate (number or rabbits born or died) during the time interval JK or KL. Rates are constant during an interval but change from interval to interval. In Figure 3.4 (similar to Figure 3.3) the relationship of rates to levels is shown.

> Let us remember, again, that a level variable is a time-varying quantity whose present value K is dependent on its immediate past value J, plus or minus any change in the variable from J to K .

In short, a level variable at any time K can differ from the level variable at any time J , only if there has been some decision to change the level or quantity of the variable between J and K .

In Figure 3.4, the circle representing the level variable (and identified as L1) represents the initial value of the variable at the start of the simulation run, at time instant 0 (zero). If there were no changes in the initial value of the variable between J and K (instants 0 and 1), the value of the variable (L1) in the immediate past instant J would be the same as the value of the variable (L2) in the present instant K. For illustrative purposes in Figure 3.4 we are assuming that only one rate affects the level, and between J and K for the iteration, the value of the rate variable is zero indicating that L2 will equal L1.

Figure 3.4
System Dynamics Program:
Execution over time



If there were some positive increment to the variable, then the value of the variable at instant K would not be the same as the value of the variable at instant J. For instance, for the second iteration, the rate is positive (above zero), hence the value of the variable at the third discrete time instant (L3) is not the same as the value at the second discrete time instant (L2).

The idea is that the basic elements to the level variable are the initial value and the rate at which the variable will change over time. The given value of a level variable could be defined as an initial value plus the successive increments (or decrements) to that initial value over time. The intuitive logic of the procedure would be as follows:

1. An initial value is determined for each level variable for the beginning of each simulation run;
2. A rate of change between the immediate past instant J and the present instant K is found;
3. This determines the value of the level variable at the period K;
4. The rate of change (Δ LEVEL.KL) between the present instant K and the immediate future instant L is then computed for use in the next iteration sequence.

As of this point the modeler would have determined the value of the level variable as of (some) instant K and the rate of change over the second DT. At this point the program would automatically shift the subscripting, and it would appear as shown in the second interation, Figure 3.4

At the end of the first iteration:

a) value of level variable at instant K is known;

b) rate of change over instant KL is calculated:

THE SUBSCRIPTS SHIFT FROM THE FIRST TO THE SECOND ITERATION:

At the beginning of the second iteration:

a) level variable value at instant K from the first iteration becomes the value of instant J for the second iteration.

b) rate variable KL from the first iteration becomes rate variable JK in the second iteration.

c) value of the level variable at the end of the second DT is calculated;

d) projected change in the level variable for interval KL of the second iteration is calculated:

THE SUBSCRIPTS SHIFT FROM THE SECOND ITERATION TO THE THIRD ITERATION.

In this way the entire simulation over n intervals is carried out by referring only to three successive time instants for each iteration, the immediate past (J), the present (K), and the immediate future (L).

All that is needed to calculate the values of a level variable over time are the initial value for that level variable and the summation of the successive rates of change. This process is normally called integration.*

The series of points in Figure 3.4 identified as L1 and L2 would suggest a level variable that is static or unchanging. The series

---

* This also suggests the simplest form of integration, rectangular integration, and values between defined points can be directly interpolated. A more complete discussion of this is given in Chapter 6.

of points in Figure 3.4 indicated by L3 and L4 would suggest a level variable that is being affected by a positive rate of change and likewise the point L5 suggests a level variable that is being affected by a negative rate of change. *

Because rates are a measure of the change of a level, they are calculated over an interval DT. Thus we find symbols such as RATE.JK or RATE.KL. These represent the number of units that some level variable will change over the interval JK or the interval KL Rates are constant during any one interval, but change from one interval to the next as seen in Figure 3.4. In order for the algorithms to function properly, it is necessary that the rates change only a small percentage from one interval (DT) to another. The rate is discontinuous from interval to interval. A discontinuous positive and negative rate sequence in shown in Figure 3.4

---

* Thus we obtain continuous values for the level variable as shown by the dotted line in Figure 3.4.

Chapter 4
NDTRAN
Language Definitions

A) Card Format

The NDTRAN Interpreter is designed as a batch oriented computer language.* The input to the Interpreter is a set of computer cards containing a System Simulation Program. Conceptually each card containing a statement of a program is divided into three parts:

Part I: Key field identification code;

Part II: Program Statement; All variable names are limited to 6 alphanumeric characters in length.

Part III: Comment or document statement or definition of the statement.

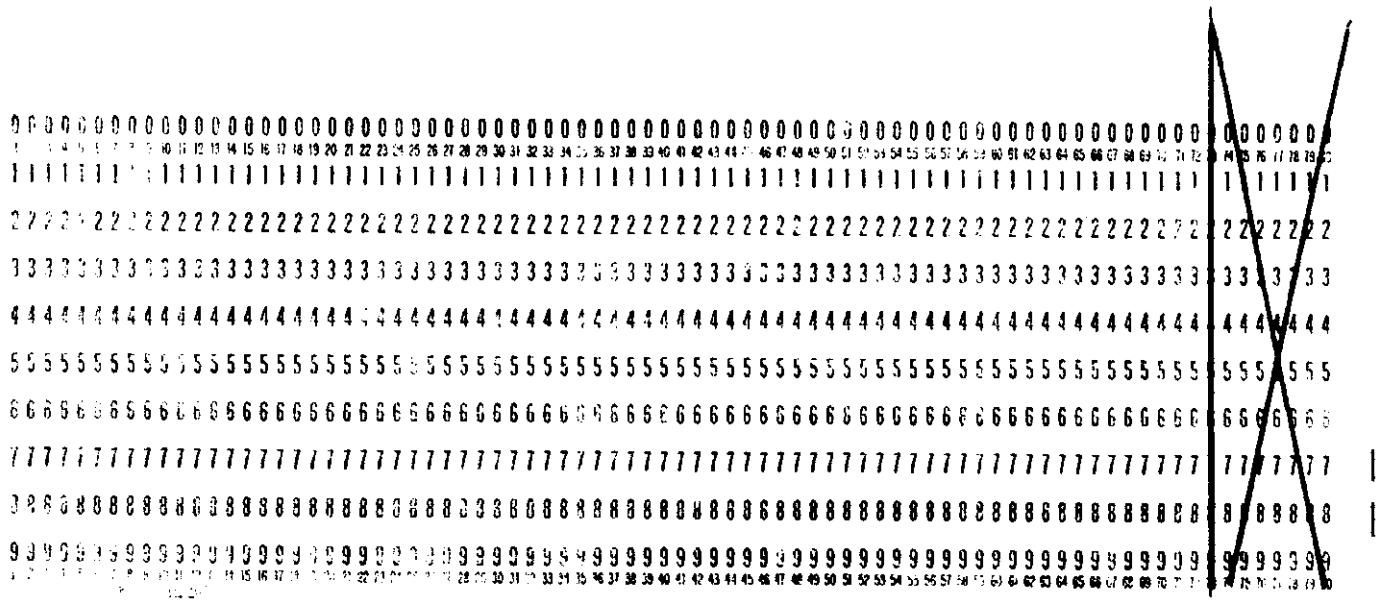
To refresh your memory, Figure 4.1 is a reproduction of a computer card. No Source Statement may extend beyond Column 72 in NDTRAN.

The only absolutely set field on the card is the field beginning in Column 1. The card itself is divided into 80 columns indicated by the small numbers between the 0 and the 1 row, and the small numbers beneath the 9 row. The bottom of the card is the one with the numbers 9 printed across. This is sometimes called the "9-edge" of the card. The top of the card is called the "12-edge". The face of the card contains the printed numbers, the back of the card is clear.

---

* The NDTRAN interpreter also works on interactive systems.

Figure 4.1
Computer Card



The left most column is the first column, or Column 1. For an NDTRAN program, one always enters the type of program statement on the card in Column 1, or beginning in Column 1. Figure 4.2 contains a list of the types of Statements that are usable, and the symbol that is entered beginning in Column 1 of the card to represent that statement.

In all instances the key field symbol will begin in Column 1 of the program statement card, and in all instances, a blank space following the key symbol marks the end of the key symbol itself. The key symbol is used by all programs to identify the type of statement, hence it is a critical part of the program statement.

Figure 4.2
NDTRAN STATEMENTS

	STATEMENT TYPE	KEY SYMBOL
1.	Level equation	L
2.	Rate equation	R
3.	Constant equation	C
4.	Initial Value equation	N
5.	Auxiliary equation	A
6.	Program specification statement	PARM or SPEC (p. 4.34)
7.	Supplementary equation	S
8.	Continuation statement	X
9.	Note statement	NOTE
10.	Table statement	T
11.	EXPND statement and MACRO procedures	EXPND
12.	Print statement	PRINT
13.	PLOT statements	PLOT
14.	RERUN statement	RERUN
15.	Definition statement	DEF

In the table shown in Figure 4.2, some of the key field symbols are more than one character in length. All begin in Column 1 and all are separated from what follows by a blank space. Blank spaces are separators or delimiters that separate fields of information.

Following the key field and the first blank on the program card is the program statement itself. The program statement will be an equation of the type described by statement type in Figure 4.2. The entire statement must be written without blanks appearing anywhere within the statement itself. Following the statement is another blank space.

Following this, the remainder of the card may be left blank or one may enter a definition of the statement-comment-for future reference.

Let us take a sample program statement and illustrate the form that all program statements must take. A Constant statement is identified by a C in Column 1. The purpose of a constant statement is to assign a numeric constant to a variable name. A variable name may be any group of alphanumeric characters, beginning with an alphabetic character that is 6 characters or less in length. There are two ways of entering a numeric constant into a variable name: First, simply use the actual constant; Second, enter the constant in scientific notation form. The ways are illustrated below:

COLUMN 1

1) C VAR1=56

2) C COMMON=5E+10

3) C NCIERC=15E3

4) C WAGE21=1.44 SET WAGE TO 1.44 AS INDEX (1950 VALUE)+
   1 2 3 (Fields)

Field 1 - the key field

Field 2 - the program statement

Field 3 - the definition of the statement. This is a non-operating part of the program card, and the form of this part (following the second blank on the card) is not important. The definition is important as it is used as the necessary input to the DOCUMENTER option of the program.

+Note: 1. The three fields are separated by blanks.

2. Field 2 allows no spaces or blanks.

In NDTRAN version 1, a variable may be any group of alphanumeric characters , beginning with an alphabetic character that is 8 (EIGHT) characters or fewer.

This form of the program statement is invariant for each type of program statement. For instance, a PLOT card, which is one that controls printed output from the NDTRAN language is as follows:

column 1

PLOT WAGE,INV,DEPR,AGRIC

1 2 (fields)

Notice that the program statements begin in column 1, with the key field symbol. Following the key symbol is a space. Following the space are the names of the variables, separated by commas or slashes, which the programmer wishes to plot. Field 1 constitutes the key field symbol and field 2 constitutes the program statement.

There are a total of 12 program control options and 15 statement types in the NDTRAN interpreter. It is the purpose of this chapter to define each control option and each equation type and to show the syntax requirements for that equation type as well as to illustrate its use.

> NDTRAN version 1 contains 10 control options and 14 statement types. The DOCUMENTATION control option and the CHECK control options are not available on version 1. The DEF statement used by the DOCUMENTATION option is not available.

Control Statements

Control statements always contain an * in the key field except for the TITLE statement from in version 2. There are a total of 12 types of control option. The TITLE card usually but not necessarily comes first, with the other option cards in any order. All option statements must come prior to NDTRAN equation statements. The control options are:

1. TITLE statement for program;
2. Cross reference table option;
3. Diagnostic warning option;
4. Documentation option;
5. Integration method option;
6. Object code option;
7. Output option;
8. Source list option;
9. Statistics option;
10. Symbol table option;
11. GO or NOGO option;
12. Check option.

It is not necessary to put in all nine control cards or control statements in order to obtain all of the options in NDTRAN. For instance if a programmer put in none of the control cards, the following default options would be provided with the compilation and execution of the program:

1. no title would be provided on the program listing and output;
2. no cross-reference table would be provided;
3. diagnostic warning messages would be provided;
4. no program documentation would occur;
5. The Adams-Bashforth integration method would be used (see Chapter 6);*
6. no object code listing would be printed out;
7. wide output would be provided on plots, prints, and other printed options;
8. the source program would be listed;
9. the program statistics (summary) would be printed;
10. no symbol table would be printed.
11. the program GO option would be in effect and the program would execute.
12. no checks are made during execution so that overflows or underflows are handled by the FORTRAN operating system in use rather than by the special built-in checks for such occurences.

Please note that in the title card, the word TITLE appears and then one blank space before the actual title begins. In all other control cards the * appears and one blank space before the option control word.

It is important to note that if a syntax error is made in a control card, NDTRAN will ignore that card. Any subsequent control cards will be processed.

---

* For NDTRAN2 the default integration method is set at the time of installation of NDTRAN2 on your computer.

IN THE FOLLOWING SECTIONS, THE MAIN DESCRIPTION IS FOR NDTRAN VERSION 2, WHILE THE DESCRIPTION INSIDE THE RECTANGLE AREA IS THE DESCRIPTION FOR NDTRAN VERSION 1. In most instances the description of the operations are identical. Where they are different, the description for NDTRAN version 1 is also included inside the block area.

It is important to note that in NDTRAN Version 1, if a syntax error is made in a control card, NDTRAN will assume that there are no more control cards. Any subsequent control cards will be assumed to be note cards and the default options will be in effect, as noted in Figure 4.3.1

In NDTRAN Version 2 Variables may be only 6 alphanumeric characters in length.

In NDTRAN Version 1 Variables may be 8 alphanumeric characters in length.

1. TITLE STATEMENT

The title statement provides a title for each page of the output of the computer run. It would normally be the first of the control cards encountered. It takes the following form:

```
        Column 1
        ↓
TITLE PRELIMINARY MODEL OF WAGE SECTOR
```

```
* PRELIMINARY MODEL OF WAGE SECTOR
```

The title card simply is an indication to the programmer of the problem being programmed. It will appear at the top of each page of the computer run. The default is no title since it is undefined.

2. CROSS REFERENCE LISTING

In NDTRAN, the programmer has the option of obtaining a cross reference listing with each run of the program. This is probably the most important control function available for the programmer who desires aid in de-bugging a program. The method of obtaining a cross-reference listing is to use the control card:

```
        Column 1
        ↓
* XREF
```

```
*XREF
```

The cross reference listing shows a non-repetitive list of all of the variable names used in the program in the left-most column. The next column is the definition column. NDTRAN provides its own line number for every program statement in the program. Each program begins with line number 1. The definition column of the cross reference table shows

```
Each program in NDTRAN Version 1 begins with line number 10001.
```

the number of the statement in the program where each variable is defined -- that is where each variable appears on the left side of an = sign. The third column of the cross reference listing shows all of the other statements in the program where the defined variable appears (to the right of an = sign). The default condition will exclude a printout of the cross-reference table.

3. DIAGNOSTIC WARNING OPTION

When writing a program in NDTRAN, the Interpreter will accept only syntactically correct program statements. The syntax error messages are listed in Appendix A for NDTRAN Version 2 and in Appendix B for NDTRAN Version 1. The default condition allows all of the syntax messages to be printed out. If the warning syntax messages are not desired, then beginning in column 1, the following control card would be used:

```
Column 1
|
* NOWARN        ( eliminates only type 3 messages on p. 4.11)

*NOWARN         ( eliminates only type 3 messages on p. 4.11)
```

There are two types of error messages in the NDTRAN Interpreter. The first is the syntax error message and the second is the execution-time error message. Only syntax errors are included in this option.+

A syntax error message is where the simulation program has an error in the form of the statement of one of the statements of the program. The error is flagged with a $ sign and the number of an

---

+The execution-time errors will be discussed in Chapter 6.

error message is printed out. NDTRAN recognizes three types of syntax errors:

1) critical errors that will inhibit the execution of the program.

2) errors that will normally not inhibit the execution of the program but might give incorrect results. The assumption here is that one might want to see the program output even if there might be an error in the program.

3) warnings that will not inhibit the execution of the program and will probably permit correct results. This constitutes syntax errors that the interpreter NDTRAN tries to correct.

Each of the three types of syntax errors is identified by the NDTRAN interpreter when the warning option is in effect.

4. DOCUMENTATION OPTION

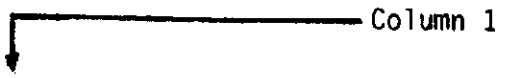
NDTRAN permits the programmer to define each of the critical variables in the program. These definitions are normally placed in field 3 of the statement as defined on page 4.4. Alternatively the definitions may be placed in DEF statements at the end of the program, or immediately following each appropriate statement. The documentation of the output program is automatic with the DOCUMENTATION control card:

* DOC

Documentation of a program is not available in NDTRAN Version 1.

5. INTEGRATION METHOD OPTION

As noted fully in Chapter 6 NDTRAN provides three different methods for carrying out numerical integrations on the computer: Euler Lower sum, Runge-Kutta and Adams-Bashforth. If the control card is omitted, the default integration method is as set by the systems programmer when NDTRAN is implemented on your own computing system. The default may be set to either of the three options noted at that time. The control card appears as:



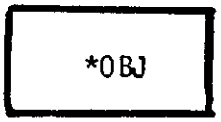

```
* EULER
* RKINT
* ABINY
```

```
*EULER
*RKINT
*ABINY
```

The default integration method in NDTRAN version 1 is the Adams-Bashforth method.

6. OBJECT CODE OPTION

NDTRAN generates its own pseudo-object-code from the source program input. The object code is then executed by NDTRAN in providing the desired print and plot output. This object code is generally not useful to most simulation programmers. The default condition causes the object code printout to be omitted. The object code printout may be obtained by:



. OUTPUT SIZE OPTION

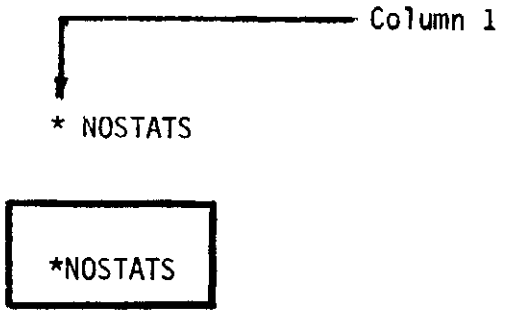
NDTRAN provides two widths for all printed or plotted output. The first is the standard computer paper width, with the NDTRAN plot on a width of 121 characters. The narrow plot option is for those desiring output on a communications terminal of some sort or who otherwise desire an output width of 72 characters. The default option is the standard wide plot. To obtain the narrow plot enter as a control card:

The narrow option printing provides printed and plotted output on standard typewriter paper size. When using the NARROW option the symbol table is difficult to read (as it is not controlled by the NARROW option), but the cross-reference is usable. When using the WIDE option the word TITLE plus the title itself may take up all 72 characters on the input statement. When using the NARROW option the word TITLE plus the title itself must take up a maximum of 54 characters.

When Using NDTRAN Version 1, the maximum title length in characters and NOTE definitions preceeding PRINT AND PLOT statements is 48 characters.

8. STATISTICS OPTION

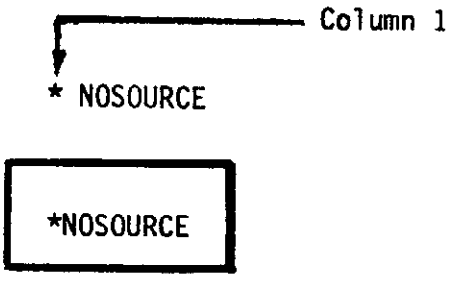
NDTRAN makes certain calculations when a program is compiling and it prints out this summary just before program execution begins. Should the user desire to suppress the statistical output information, the following control card should be entered beginning in Column 1 at some point with the other control cards and before the program itself begins:



The standard default condition in the absence of the NOSTATS control card is that the statistical information will be printed out.

9. SOURCE LISTING

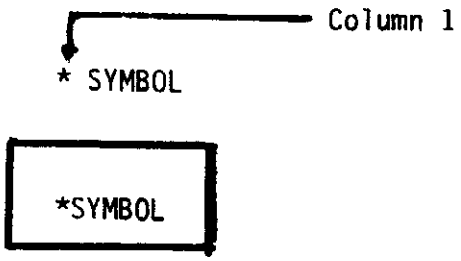
The SOURCE LISTING control card simply provides a listing of the source program. The default condition of NDTRAN in the absence of a control card is that the source listing will be provided. In order to ensure that the source program will not be listed, enter the following control card on column 1:



10. SYMBOL TABLE LISTING

In NDTRAN one of the options is to show a listing of the variable names in the program, and to indicate the type of variable. In this instance a variable is thought of as a SYMBOL. Therefore, a Symbol table listing is a list of all of the variable names used in the program with an indication of the type of variable that it is and the type of output desired for that variable (symbol).

The leftmost column shows the name of the variable in the program. The second column shows the type of variable. The third column shows the type of output requested for the variable. The fourth column is the internal program number for the variable. The fifth column is the function number for each function used in the program. The rightmost column indicates the number of elements in each table function and the core location of the beginning table number for each table. To obtain the Symbol Table listing enter:



11. GO AND NOGO OPTION

The standard default for the NDTRAN Interpreter is that execution will be attempted following compilation. In some instances, the programmer might wish to obtain some of the control card output options without executing the program. In this instance the control card would be entered as follows:

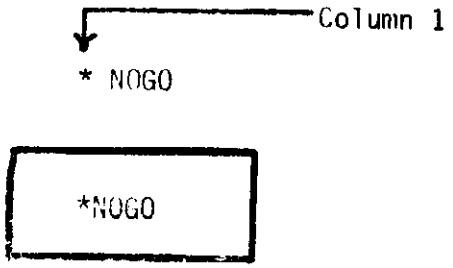


Illustration of Control Statement Options

Perhaps the easiest way to explain many of the points noted above regarding the control card options is to execute a program containing the indicated options and errors. In the following section the reader should note that there are certain syntax differences between NDTRAN Version 1 and Version 2.

1. Example 1.

The rabbit model shown in Figure 3.2 was translated into the source program shown in Figure 4.3 before being prepared for key punching on cards.

Figure 4.3
Rabbit Population
Syntax Errors

```
TITLE ERROR OPTIONS
* EULER
* XREF
* NARROW
X SYMBOL
L RAB.K=INTGRL(RB.JK-RX.JK)
C RABI=150
R RB.KL=RAB.K*FERT
R RD.KL=RAB.K*DEAT
C FERT=.06
C DEAT=.04
PARM DT=.08
PARM START=0
PARM STOP=100
PARM PLTPER=10
PARM prtper=10
NOTE RABBIT MODEL PRINTED DATA
PRINT RAB,RB,RD
NOTE RABBIT MODEL PLOTTED DATA
PLOT RAB,RB,RD
```

There are five syntax errors in this program shown as listed by the computer. Figure 4.3 shows the program as it was typed before being submitted to the NDTRAN interpreter for computer execution. Figure 4.4 indicates how NDTRAN handled the errors noted. The discussion of the errors below refers to Figure 4.3. You may wish to compare the discussion below with the way that NDTRAN handled the errors. The syntax errors in Figure 4.3 are:

a) a keypunch error was made on line 5 of the program on a control card where an X was punched instead of a * . NDTRAN would regard this as an attempt to continue the previous line, which may not be done on control cards.

b) line 6 of the program has two errors:

   1) the first refers to variable RX. A keypunch error was made and the RX should have been RD, (see line 9). This is a CRITICAL error.

   2) The second error is that the level variable RAB was not initialized. This is a CRITICAL error.

c) On line 7 the variable RABI was entered and not used. This would result in a WARNING error only.

d) The error on line 9 is similar to that on line 7, in that it indicates that variable RD was not used in the program. THE WARNING MESSAGE IN NDTRAN2 CAN BE A POWERFUL DIAGNOSTIC HELP.

Figure 4.4

```
PAGE 1    ERROR OPTIONS                                    (C) 1978 UND

     * * * * *    S O U R C E    L I S T I N G    * * * * *

     0001  TITLE ERROR OPTIONS
     0002  * EULER
     0003  * XREF
     0004  * NARROW
           X SYMBOL

            0) *****      E R R O R      *****          ND0101

     0005  L RAB.K=INTGRL(RB.JK-RX.JK)
             $                    $
            2) *****  C R I T I C A L   *****          ND0516
            1) *****  C R I T I C A L   *****          ND0524

     0006  C RABI=150
             $
            1) *****   W A R N I N G    *****          ND0588

     0007  R RB.KL=RAB.K*FERT
     0008  R RD.KL=RAB.K*DEAT
             $
            1) *****   W A R N I N G    *****          ND0587

     0009  C FERT=.06
     0010  C DEAT=.04
     0011  PARM DT=.08
     0012  PARM START=0
     0013  PARM STOP=100
     0014  PARM PLTPER=10
     0015  PARM PRTPER=10
     0016  NOTE RABBIT MODEL PRINTED DATA
     0017  PRINT RAB,RB,RD
     0018  NOTE RABBIT MODEL PLOTTED DATA
     0019  PLOT RAB,RB,RD
PAGE 2    ERROR OPTIONS                                    (C) 1978 UND

     * * *    S T A T S    A N D    O P T I O N S    * * *

       19 SOURCE STATEMENTS

        5 DIAGNOSTIC MESSAGES

             2 WARNINGS

             1 ERROR

             2 CRITICALS
```

In summary, NDTRAN provides a WARNING flag to all variables not used in the program, whatever the reason. The ERROR condition is because of an incorrect entry in a control card field, causing NDTRAN to ignore that control card statement. The CRITICAL ERROR occurs because of improper entry of variable names which cannot be recovered from for correct execution. The same program for Version 1 appears as Figure 4.3.1 along with the computer output in Figure 4.4.1.

Figure 4.3.1
Rabbit Population
Syntax Errors

```
**********    S O U R C E    L I S T I N G    **********

10001   * ERROR OPTIONS
10002   *XREF
10003   XSYMBOL
10004   *EULER
10005   *OBJ
10006   *NARROW
10007   L RAB.K=INTEGRAL(RB.JK-RX.JK)
10008   C RABI=150
10009   R RB.KL=RAB.K*FERT
10010   R RD.KL=RAB.K*DEAT
10011   C FERT=.06,DEAT=.04
10012   SPEC DT=.08,START=0,STOP=100,PRTPER=10
10013   NOTE RABBIT DATA
10014   PRINT RD,RB,RAB
```

Figure 4.4.1

There are three syntax errors in the program. The first has to do with the syntax of control card statements. This error in line 10003 concerns the statement XSYMBOL. The correct form would be *SYMBOL. NDTRAN, once it encounters a syntactically incorrect control card will assume that there are no more control cards and default options will be in effect.

There are two additional syntax errors. No initial value was provided for the level variable RAB, thus generating an error in line 10007. The second, error in line 10007 was the misspelling of Variable RD as variable RX.

In NDTRAN Version 1, the source program and the error conditions appear separately as shown below:

Rabbit Population:
Syntax Errors:
DIAGNOSTIC MESSAGES OPTION

```
**********    D I A G N O S T I C    M E S S A G E S    ****

10003  XSYMBOL

       0) *****  C R I T I C A L  *****          ND0100

10007  L RAB.K=INTEGRAL(RB.JK-RX.JK)
         $                      $
       1) *****  C R I T I C A L  *****          ND0524
       2) *****  C R I T I C A L  *****          ND0516
```

*** THE WARNING SYNTAX ERRORS DO NOT SERVE THE SAME FUNCTION IN NDTRAN VERSION 1, HENCE THEY DO NOT APPEAR IN THIS PROGRAM OUTPUT.

From the Version 2 Diagnostic Error messages shown in Appendix A we find for this program.

Error Number:	Error Message:
101	The previous card may not be continued. This continuation card will not be processed.
516	No equation exists for this variable. It is therefore undefined.
524	This level variable must be given an initial value.
587	This variable is used for output. It should have been defined as a supplementary.
588	This variable is not used for output and has no affect on any other variable.

From the Version 1 Diagnostic Error messages shown in Appendix B, we find the errors generated by the program of Figure 4.4.1. They are not identical in number or content.

Error Number:	Error Message:
100	The card type indicator which begins in column one is not recognized by NDTRAN. That card will not be processed.
516	No equation exists for this variable; it is therefore undefined.
524	This level variable must be given an initial value.

2. Example 2.

Figure 4.5 is the statistics option output for NDTRAN (see Figure 4.3 & 4.4) and it is labeled appropriately. Starting at the top of the statistics and options section, we noted that there were 15 source statements in the program with 5 diagnostic messages.

Figure 4.5
Statistics and Options

```
PAGE 2    ERROR OPTIONS                                    (C) 1978 UND

    * * *    S T A T S    A N D    O P T I O N S    * * *

      19 SOURCE STATEMENTS

       5 DIAGNOSTIC MESSAGES

            2 WARNINGS

            1 ERROR

            2 CRITICALS

    CARD TYPE          OCCURANCE

      C                    3
      PARM                 5
      L                    1
      R                    2
      X                    1
      *                    3
      NOTE                 2
      PRINT                1
      PLOT                 1
      TITLE                1

    OPTIONS IN EFFECT:

      NOCHECK      NOSYSTEM     NODOCUMENT   NARROW
      STATS        NOGO         NOSYMBOL     XREF
      WARN         NOOBJECT     SOURCE       NOTIME

    INTEGRATION METHOD:

         EULER LOWER SUM
```

The last part of the Figure 4.5 indicates that the following options were operative:

> The NOCHECK execution option was in effect, no SYSTEM information would be made available, the program would not be DOCUMENTED, the NARROW output PRINT and PLOT option was in effect, STATISTICS and OPTIONS would be printed, the program would not execute (NOGO) because of the critical errors in the program, no SYMBOL table would be printed, no CROSS-REFERENCE listing would be printed, DIAGNOSTIC warnings would be printed, no OBJECT code listing would be operative, the SOURCE program would be listed, no TIME estimate would be given and finally, the Euler Lower Sum integration option was in effect.

3. Example 3.

The CROSS-REFERENCE table option is illustrated in Figure 4.6. The first column shows the variable names in the program. The second column shows the line number where they were defined. The third column (with extensions) shows the places in the program where each variable is used (Right side of = sign). Remember, in an NDTRAN program, the variable may be used before it is defined, as NDTRAN does its own sort of the program statements prior to compilation of the program. Figure 4.6 shows the model of Figure 4.3 as corrected, with its CROSS REFERENCE table.

Figure 4.6
Rabbit Population
Cross Reference Table

PAGE 3 ERROR OPTIONS (C) 1978 UND

* * * * * C R O S S R E F E R E N C E * * * * *

VARIABLE NAME	DEFINITION	REFERENCES			
DEAT	0010	0008			
DT	0011				
FERT	0009	0007			
PLTPER	0014				
PRTPER	0015				
RAB	0005	0007	0008	0017	0019
RABI	0006				
RB	0007	0005	0017	0019	
RD	0008				
START	0012				
STOP	0013				
TIME					

4. Example 4.

As for the remainder of the control card options, the easiest way for the student or user to familiarize oneself with them is to simply experiment with them. Run the program with no control cards in effect and notice the default options that are printed. Then run several programs that require specific control card output options, and execute them. The one additional control card option that may require some explanation is the DOCUMENTATION option.

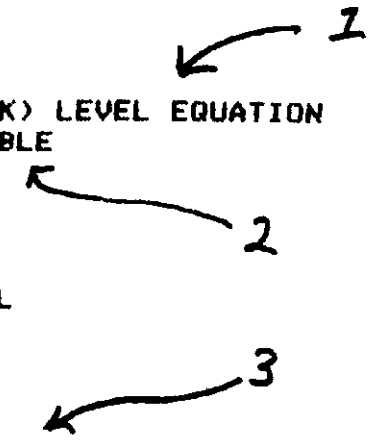
The DOCUMENTATION Option provides a way for the programmer to define any variables that appear in the program and in this way to document the program. Each time the variable is used in any equation, the definition will appear under the appropriate equation(s).

The DOCUMENTATION option is not available on NDTRAN Version 1.

A program that is set up for DOCUMENTATION is shown in Figure 4.7.

Figure 4.7
DOCUMENTER TEST PROGRAM

```
00010 TITLE DOCUMENTER TEST
00020 * DOC
00030 * EULER
00040 * NARROW
00050 L LEVEL.K=INTGRL(RATE1.JK-RATE2.JK) LEVEL EQUATION
00060 DEF RATE1 IS THE FIRST RATE VARIABLE
00070 R RATE1.KL=LEVEL.K*CON1
00080 R RATE2.KL=LEVEL.K*CON2
00090 C CON1=.1
00100 C CON2=.05
00110 N LEVEL=100 INITIAL VALUE OF LEVEL
00120 PARM STOP=20
00130 PARM DT=1
00140 PARM PLTPER=2
00150 PRINT LEVEL
00160 DEF RATE2 IS THE SECOND RATE VARIABLE
00170 DEF CON1 IS THE FIRST CONSTANT
00180 DEF CON2 IS THE SECOND CONSTANT
READY
```



The program in Figure 4.7 has the variable definitions contained in it, in the following ways:

1. In the first equation, indicated by arrow 1 , the variable defined is the variable appearing to the left of the = sign. The definition appears in field 3 , as defined on page 4.4 . Field 1 is the KEY field. It is followed by one space and then by the equation. The equation is in field 2. Field 2 is followed by one space then field 3. When in this form the definition always refers to the variable to the left of the = sign.
2. The second statement in the model is a DEF statement, indicated by arrow 2. The definition in the DEF statement is for the named variable that immediately follows the key field. In this instance the variable RATE1 is being defined.
3. Some programmers prefer to place the DEF statements at the end of the program as shown by arrow 3.

Thus variables may be defined in the definition field of the statement or on DEF statements that may be placed at any point in the program <u>after</u> the control option statements. DEF statements or statements using the definition field <u>should not be continued</u>. The program, when executed appears with the documentation as shown in Figure 4.8.

Figure 4.8
Program Documentation

```
PAGE 1     DOCUMENTER TEST                                    (C) 1978 UND

     * * * * *    S O U R C E    L I S T I N G    * * * * *

     0001  TITLE DOCUMENTER TEST
     0002  * DOC
     0003  * EULER
     0004  * NARROW
     0005  L LEVEL.K=INTGRL(RATE1.JK-RATE2.JK) LEVEL EQUATION

               LEVEL   - LEVEL EQUATION
               RATE1   - IS THE FIRST RATE VARIABLE
               RATE2   - IS THE SECOND RATE VARIABLE

     0006  DEF RATE1 IS THE FIRST RATE VARIABLE
     0007  R RATE1.KL=LEVEL.K*CON1

               RATE1   - IS THE FIRST RATE VARIABLE
               LEVEL   - LEVEL EQUATION
               CON1    - IS THE FIRST CONSTANT

     0008  R RATE2.KL=LEVEL.K*CON2

               RATE2   - IS THE SECOND RATE VARIABLE
               LEVEL   - LEVEL EQUATION
               CON2    - IS THE SECOND CONSTANT

     0009  C CON1=.1

               CON1    - IS THE FIRST CONSTANT

     0010  C CON2=.05

               CON2    - IS THE SECOND CONSTANT

     0011  N LEVEL=100 INITIAL VALUE OF LEVEL

               LEVEL   - LEVEL EQUATION

     0012  PARM STOP=20
     0013  PARM DT=1
     0014  PARM PLTPER=2
     0015  PRINT LEVEL

               LEVEL   - LEVEL EQUATION

     0016  DEF RATE2 IS THE SECOND RATE VARIABLE
     0017  DEF CON1 IS THE FIRST CONSTANT
     0018  DEF CON2 IS THE SECOND CONSTANT
```

C) Definition and Syntax of Equation Types

There are 14 basic types of equations in NDTRAN for Version 1 and for Version 2. However, the reader should be aware that the syntax of certain equations differs somewhat between versions and will be so noted. We wish to take these in order, beginning with the state variable equation more frequently called a level equation.

1. LEVEL EQUATION

A level equation defines a state or level variable that is a time-varying quantity whose present value is dependent on its own past value PLUS or MINUS any changes in that variable over the most recent time period. For instance, the size of a given population today is the result of its own size yesterday plus or minus any changes in that population over the past day. In a colloquial way the subscript J refers to yesterday, the subscript K refers to today and the subscript L refers to tomorrow. More precisely the subscript J refers to a given time instant in the immediate past. The subscript K refers to the given present time instant and the subscript L refers to a time instant in the immediate future. We refer to the time between J and K (whether it refer to one day, or one month or one year) as the DT period. The DT period is always set appropriately to the given model. The NDTRAN language form of the level equation is as follows:

```
L RAB.K=INTGRL(RATEIN.JK-RATOUT.JK)
```

```
L RAB.K=INTEGRAL(RATEIN.JK-RATOUT.JK)
```

There are a number of different computer language implementations similar to NDTRAN and not all use the same form for the level equation. The NDTRAN version is as shown above with the specific integration method desired established with a control card. This point is discussed

specifically in Chapter 6. *

2. RATE EQUATION

A rate equation may be regarded as a policy decision that will affect the measure of a level equation over time. Rates are the amount of change that a level equation will undergo in a defined time period (DT). A rate will increase a level or decrease a level or, together with the other rates affecting the same level, will limit the change in the level.

An interpretation of this definition may be given as follows. The rate at which savings will accumulate in an account of a given size is dependent on the earning power of the savings. Here, the term rate is always used when referring to the number of units that a level of a given size will change in a given period of time. If you have $1000 in a savings account, and you will earn 6 percent on that $1000 per year that is left in the account (compounded), then the rate at which the funds will increase is 6 percent times the amount in the account for the year. For the first year, the rate of accumulation of the savings account would be $60.00. The calculation would be:

RATE	=	constant	(times)	Level
$60.00	=	.06	*	$1000.

*The form of the Level Equation shown for NDTRAN above is a bit different from DYNAMO, but it implies precisely the same type of operation. Most computer inplementations follow the form:

L RAB.K=RAB.J+(DT)(RATEIN.JK-RATOUT.JK)

The quantity of the population at time K is equal to the quantity of the population at time J plus the change in the increases to the population less the decreases over the solution period for the model (DT). While the form is different. The effect of the integration technique on the variable is identical using the euler integration method.

Most people are used to referring to the constant .06 as the rate of interest. The term rate as used here has a specific meaning: the amount by which the level variable will change per period of time. The earning power of the savings account is expressed as a constant of .06 per period. The period of time of compounding, a year in this example, is the solution interval for the model or the DT. However, these two periods do not have to be the same. In many instances it is important to remember that the manner in which the rate equation is set up depends upon the solution interval taken. For instance, if the interest compounded the savings account on a yearly basis, the rate might be calculated in one way. If the account compounded daily, the rate equation would be set up another way. One must pay attention that the manner in which the rate equation is defined is consistent with the length of the solution interval chosen for the model. The constant is defined in units, thus DT must be the adjusting factor.

The NDTRAN language form of a rate equation is:

```
R RATE.KL=LEVEL.K*CONST
```

A rate variable, on the left side of the = sign, may be defined by a level variable, a constant, and/or an auxiliary variable on the right.

3. CONSTANT EQUATION

A constant in NDTRAN does not change with time. For example, the constant (CONST) defined above is the interest constant. The major purpose of a constant is to aid in the conversion of a level variable or an auxiliary variable to a rate variable.

The form of the constant equation is:

```
C CONST=.06
C CON=56E3
```

4. INITIAL VALUE EQUATION

We have already seen that the level variable is a time-varying variable which is dependent on some previous value. Thus the level variable must have a value that is established before the run begins. The initial value equation provides an initial or immediate past value for the level variable at the start of the first iteration of the simulation of the model. The initial value is found in an equation with the following form:

```
N LEVEL=150
```

An alternative form for providing an initial value for a level, used if you wish to change the initial value of a level equation in a RERUN, (See below) is as follows:

```
N LEVEL=LEVEL1
C LEVEL1=150
```

By changing the constant equation in the RERUN mode it is possible to assign new initial values for level equations on a succession of comparative runs.

5. AUXILIARY EQUATION

Since rates can only be determined by levels, constants or auxiliaries, one frequently requires the use of intermediate or auxiliary variables. For instance, in the American Economy one of the most widely used indicators in the system is the Gross National Product, or GNP. This variable is an input to a good many decisions involving rates of change of levels. For example, the levels in the economy are reasonably clearly defined as: the number of houses, the number of factories, the miles of road, the acres of cultivated land, and the like. These would constitute clearly defined state variables, or levels. Gross National

Product, however, is a defined concept. It is not in fact a level or state variable. The definition of GNP is final product output counted once, or the sum of value added. However, by definition it excludes a substantial number of produced goods and services. A purchased meal for instance, is included in GNP but a meal produced at home is not. Simply, GNP is a defined concept, that is defined at each appropriate point of time, but it is not a state variable.

An auxiliary variable then, is one that is defined by other level variables, other auxiliary variables, and/or constants. An auxiliary variable, then, has two basic purposes:

A) as a defined variable;

B) in using functions.

We could assume that we are dealing with a population model of the United States. The model is broken down by sex, but not by ethnic group. Further, the age groups are as follows:

a) under 18

b) 18 - 24

c) 25 - 29

...

n) over 85

Using an auxiliary equation it would be possible to add up and place in a variable all male population under the age of 55, if that were needed. Further, if one were doing a study of social security recipients one might add up all male and female persons under the age of 18 and over the age of 65. This would illustrate the use of the auxiliary equation. The other major use of the auxiliary equation is shown in the Chapter 5, when functions are discussed.

The form of the auxiliary equation is:

A VALUE.K=VAR1.K+VAR2.K+VAR3.K

Alternative forms of the Auxiliary Equation are:

A RUF.K=CLIP(RUF2,RUF1,TIME.K,LEV)

A RUM.K=TABHL(VAL,PC.K,0,600,100)

A IN.K=(OUT.K)(1-FAC.K)

In short, nearly any type of modelling operation where it is necessary to define the value of a variable as of a given time, would require an auxiliary equation.

6. PARM STATEMENT

PARM statements contain information to the program and to the interpreter regarding the parameters required for executing the program. The basic rule in Version 2 of NDTRAN is that each PARM card may contain only one parameter to NDTRAN, but the parameter may be defined by a numeric constant or by an equation.*

The normal information for the PARM statements are:

a) the solution interval or DT for the program;

b) the starting time for the program, or START;

c) the stopping time for the program, or STOP;

d) the print period for the program or PRTPER, which tells the program how often you desire to print the value of a variable;

e) the plot period for the program or PLTPER, which tells the program how often you desire to plot the value of a variable.

*This is different from the DYNAMO approach which defines each parameter by a numeric constant.

```
PARM DT=1
PARM START=1900
PARM STOP=2000
PARM PRTPER=5
PARM PLTPER=5
```

```
SPEC DT=1,START=1950,STOP=2000,PRTPER=5,PLTPER=5
```

The form of the PARM cards, or the SPEC card for Version 1, would allow the program to start its run at the year 1950 and to stop at the year 2000. Essentially this amounts to initializing the variable TIME to the value of the START parameter and to allow the program to execute until the value of TIME (an implicit variable) is equal to the stop value of the STOP parameter. The variable TIME is never defined explicitly. It is automatically initialized when the run begins. Despite not being explicitly defined in a given equation, it may be used as often as desired on the right hand side of an = sign in most equations, or as the argument of a function. The use of all PARM cards is required. If one is not used the standard defaults are given in the following Table:

Table 4.1
PARM Card Defaults

Card	Default
DT	Runs until Stats option is executed.
START	Sets TIME = 0.0 and executes properly.
STOP	Will run until the actual execution of model.
PLTPER	Executes properly but deletes all Plots.
PRTPER	Executes properly but deletes all numerical printouts.

7. SUPPLEMENTARY EQUATION

A Supplementary equation constitutes an equation to define a variable where the only purpose of the variable is for output purposes in a PRINT or PLOT statement. In this equation, any variable defined on the left side of the equation may not appear on the right side of any other equation except that of another supplementary equation. The form of the supplementary equation is:

```
S SVAR.K=VAR1.K*RATE1.JK/CONST
```

Any type of variable may be used on the right side of a supplementary equation since supplementary variables have a restricted use.

8. CONTINUATION STATEMENT

Any program statement that will not fit between columns 1 and 72 of a program statement card may be continued on a second statement card. This is done by placing an X in column 1 of the following card. One space is then skipped and the statement is continued. An example is shown below, in the context of a TABLE statement, which is the statement that is most likely to exceed the single input card. Only one continuation card may be used per statement. All 72 columns do not have to be filled to use a continuation card, but it is advisable to end the first card with an appropriate delimiter (,or/).

```
T ALT=9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,
X 24,25,26,27,28,29,30
```

```
T ALT=9/10/11/12/13/14/15/16/17/18/19/20/21/22/23/
X 24/25/26/27/28/29/30
```

9. NOTE statement

A NOTE statement is a non-executable statement in the NDTRAN interpreter that is used to define or comment upon a particular part of the program. Note statements may either be blank or contain an alphabetic or numeric definition.

NOTE THE FOLLOWING EQUATION DEFINES WORK

10. TABLE statement

A TABLE statement is a part of the Table Function and is more properly discussed in Chapter 5, Part D.

11. EXPND Statement and MACRO Procedures:

a) User Defined Procedures

1. MACRO Definition

NDTRAN allows the writing of procedures (similar to MACRO procedures in assembly language) if such is desired, and if such are not available as built-in functions or procedures. This is accomplished using the MACRO capability together with the EXPND statement that are part of the NDTRAN statement procedures. Essentially the user writes the MACRO procedures as follows:

```
MACRO MAIN(POS,VEL,CONST)
L POS.K=INTGRL(VEL.JK)
N POS=0
R VEL.KL=CONST*TIME.K
MEND
```

```
MACRO MAIN(POS,VEL,CONST)
L POS.K=INTEGRAL(VEL.JK)
N POS=0
R VEL.KL=CONST*TIME.K
MEND
```

The MACRO procedures begins with the word MACRO in column 1 of the card as shown above and terminates with the word MEND. MACRO DEFINITIONS MUST ALWAYS BE INSERTED INTO THE PROGRAM BEFORE ANY EXPND statements for that MACRO.*

The interpreation to the MACRO statement is:

The MACRO key word indicates that a user defined function or subroutine is to be developed. The first word or name following the blank is the name of the procedure (MAIN). The dummy arguments, up to 18 in number allowed to be used, are enclosed in parens ( ) following the name of the procedure. In this instance the three arguments are POS, VEL and CONST. Arguments to a MACRO procedure never use time subscripts.

*This is one of two exceptions to the lack of ordering required of NDTRAN programs. The other is that RERUN cards must come at the end of the main program.

There is a relatively wide latitude for statements, following the rules already established, that can be used as arguments in a MACRO procedure. The illustration above indicates that the arguments include:

POS	a level variable;
VEL	a rate variable;
CONST	a constant.

2. EXPND Statement

The MACRO procedure may be called or used as many times as desired by the use of the EXPND statement. This consists of the key word EXPND followed by the name of the procedure, followed in parens ( ) by the actual arguments desired to be used. Figure 4.8 is a sample user generated program which illustrates the use of the user written MACRO procedure shown above.

Figure 4.9 is the actual program produced by the interpreter and is given as a source listing. Note the expansion which takes place due to the MACRO Usage.

Figure 4.8
Program Using MACRO Statement

```
TITLE USE OF MACRO CAPABILITY
* NARROW
* CHECK
* NOSTATS
MACRO MAIN(POS,VEL,CONST)
L POS.K=INTGRL(VEL.JK)
N POS=0
R VEL.KL=CONST*TIME.K
MEND
EXPND MAIN(POS1,VEL1,CONST1)
C CONST1=2
EXPND MAIN(POS2,VEL2,CONST2)
C CONST2=3
PARM DT=1
PARM START=0
PARM STOP=20
PARM PLTPER=1
PLOT POS2=2,POS1=1
```

Figure 4.9
MACRO Statement Procedure
User Defined

```
* * * * *   S O U R C E   L I S T I N G   * * * * *

0001   TITLE USE OF MACRO CAPABILITY
0002   * NARROW
0003   * CHECK
0004   * NOSTATS
0005   MACRO MAIN(POS,VEL,CONST)
0006   L POS.K=INTGRL(VEL.JK)
0007   N POS=0
0008   R VEL.KL=CONST*TIME.K
0009   MEND
0010   EXPND MAIN(POS1,VEL1,CONST1)
0011+  L POS1.K=INTGRL(VEL1.JK)
0012+  N POS1=0
0013+  R VEL1.KL=CONST1*TIME.K
0014+  MEND
0015   C CONST1=2
0016   EXPND MAIN(POS2,VEL2,CONST2)
0017+  L POS2.K=INTGRL(VEL2.JK)
0018+  N POS2=0
0019+  R VEL2.KL=CONST2*TIME.K
0020+  MEND
0021   C CONST2=3
0022   PARM DT=1
0023   PARM START=0
0024   PARM STOP=20
0025   PARM PLTPER=1
0026   PLOT POS2=2,POS1=1
```

+ · generated

+ · generated

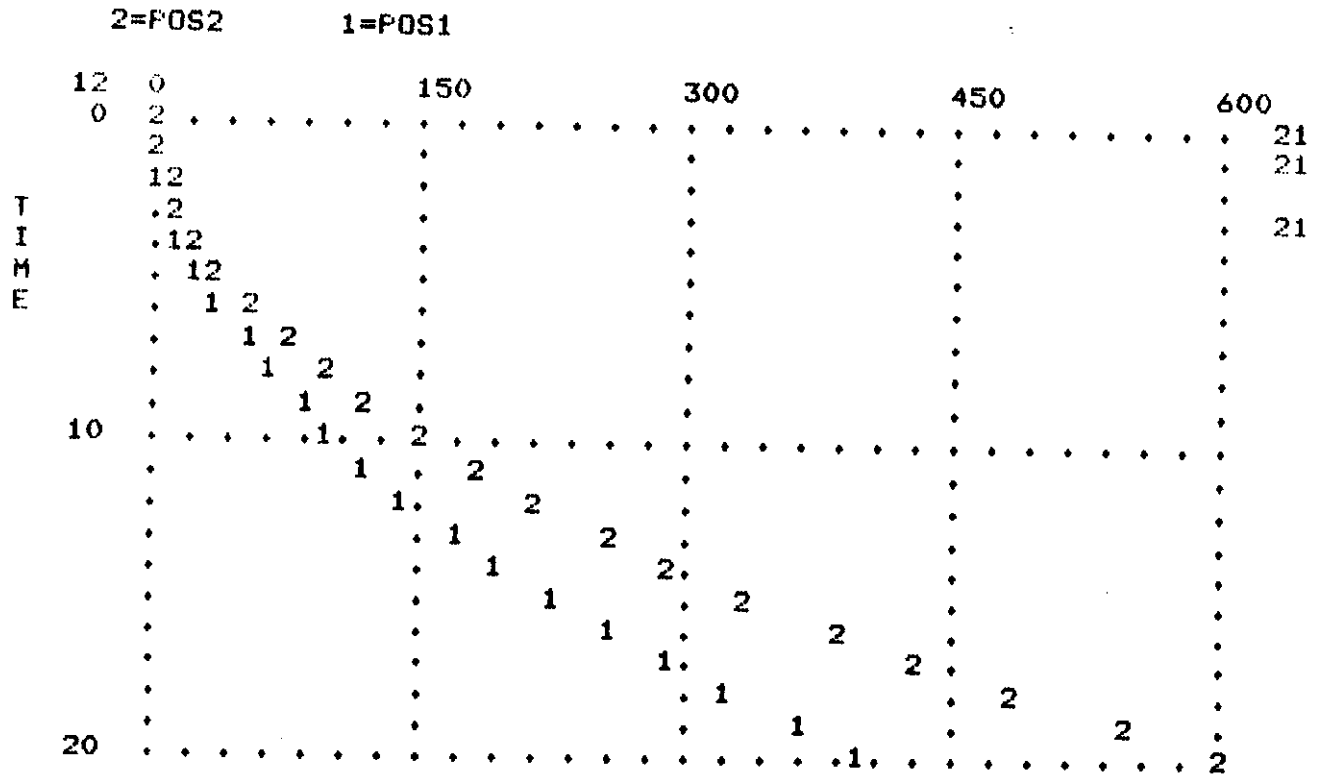
In the program, the user defined MACRO procedure (appearing before the EXPND statements) is given in lines 5 through 9. The first EXPND statement is in line 10 and the second EXPND statement is in line 12(figure 4.8). It is possible for the user to have as many EXPND statements as is desired for any given MACRO procedure. As the expansion occurs, the arguments from the EXPND statement replace the dummy arguments of the MACRO definition statement. This substitution is done by corresponding positions in the argument list of the MACRO statement and the EXPND statement. Figure 4.10 is the output from the program shown in Figure 4.8.

Figure 4.9.1
MACRO Statement Procedure
User Defined

```
**********    S O U R C E    L I S T I N G    **********

10001   * USE OF MACRO CAPABILITY
10002   *NARROW
10003   *NOSTATS
10004   MACRO MAIN(POS,VEL,CONST)
10005   L POS.K=INTEGRAL(VEL.JK)
10006   N POS=0
10007   R VEL.KL=CONST*TIME.K
10008   MEND
10009   EXPND MAIN(POS1,VEL1,CONST1)
10010+  L POS1.K=INTEGRAL(VEL1.JK)
10011+  N POS1=0
10012+  R VEL1.KL=CONST1*TIME.K
10013+  MEND
10014   C CONST1=2
10015   EXPND MAIN(POS2,VEL2,CONST2)
10016+  L POS2.K=INTEGRAL(VEL2.JK)
10017+  N POS2=0
10018+  R VEL2.KL=CONST2*TIME.K
10019+  MEND
10020   C CONST2=3
10021   PLOT POS2=2,POS1=1
10022   SPEC DT=1,START=0,STOP=20,PLTPER=1
```

The only difference in the MACRO procedures for Version 1 and Version 2 are in the nature of the line numbers attached to each statement, the word INTEGRAL, and the substitution of the SPEC statement for the PARM statements.

Figure 4.10
MACRO User Procedure
Program Plot



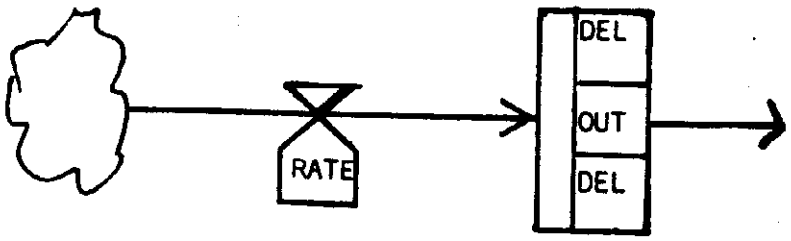
The example used in Figure 4.9 and 4.9.1 is that of a rocket simulation, where the position of the rocket at any given time instant is a linear function of time. The MACRO procedure defines this. POS1 in the plot of Figure 4.10 shows the position of the rocket over time when a given coeffient is used with TIME. POS2 in the plot shows the position of the rocket over time when a larger coefficient is used. The user defined procedure is called two times. Each time that it is called NDTRAN prints out the complete user defined procedure with the appropriate arguments in place. Each of the expanded procedure statements is preceeded by a + sign. In this way the student may verify that the user defined procedure has been properly set up and that the proper arguments are substituted into the user defined procedure at the appropriate locations in the program.

b) Delay Procedures

NDTRAN has a number of procedures that contain integration and are normally inserted in an information or materials flow path. As defined in INDUSTRIAL DYNAMICS and PRINCIPLES OF SYSTEMS, rates may not directly affect different rates, but may affect other rates only through levels. There is perfectly sound modeling theory to support this. Complex social systems do not react immediately to a given stimulus. For instance, assume that 35 packages are placed in the postal system at the same time. If they were all delivered simultaneously and without any time lapse between posting the package and delivery of the package, the result would in fact be a STEP function, defined below. In fact there is a time lapse between the posting of the package at the mailing station and receipt of the package at the delivery address. In NDTRAN these material delays are defined by use of the DELAY1, DELAY3 and SMOOTH functions.

Figure 4.11 shows the Flow Diagram of a system containing a materials, goods or persons Delay Function which might be a DELAY1, DELAY3 or SMOOTH.

Figure 4.11
Materials Delay



Information delays, identified as DLINF1 and DLINF3 are placed in the information channel which means that they come between LEVELS and subsequent RATES. The Flow Diagram in Figure 4.12 below indicates a first order negative feedback loop with and without a delay procedure. The information delay procedure might be either a DLINF1 or a DLINF3.

The delay procedures cause certain elementary equations to be generated and these are always shown on the program source listing, with the numbers of the generated equations being followed by a + sign. The symbol for the materials delay procedure and the information delay procedure, respectively, is shown below. The solid arrow into the

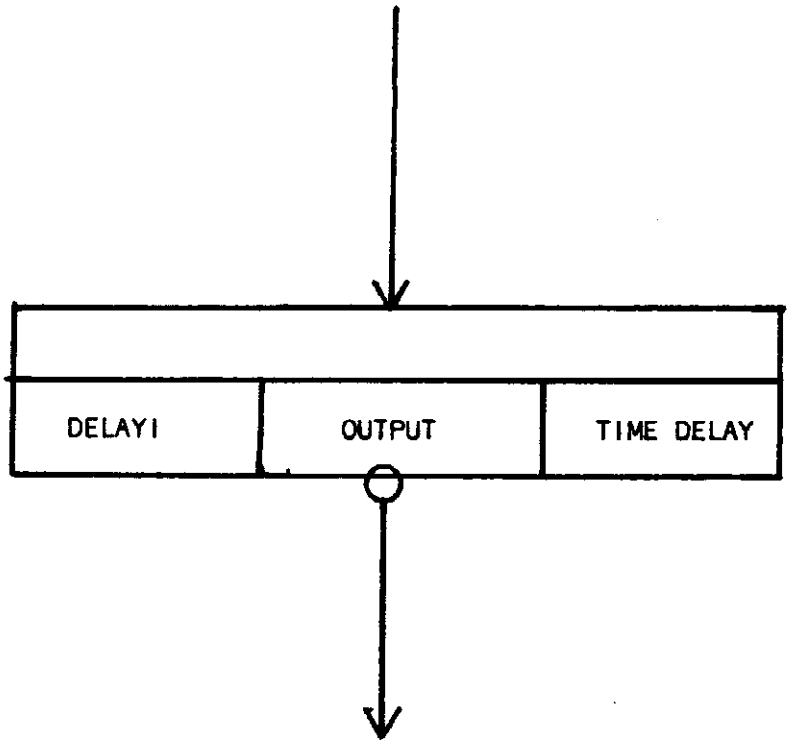
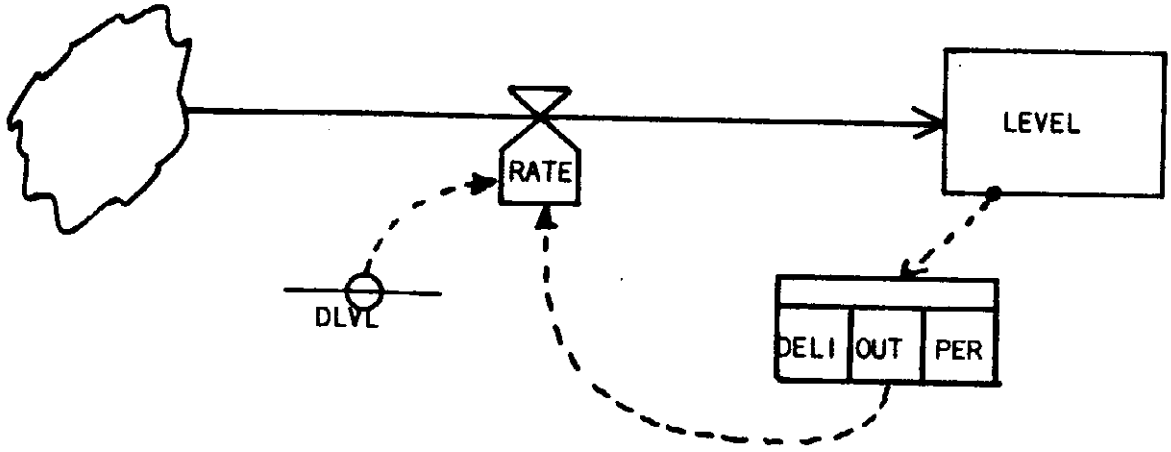
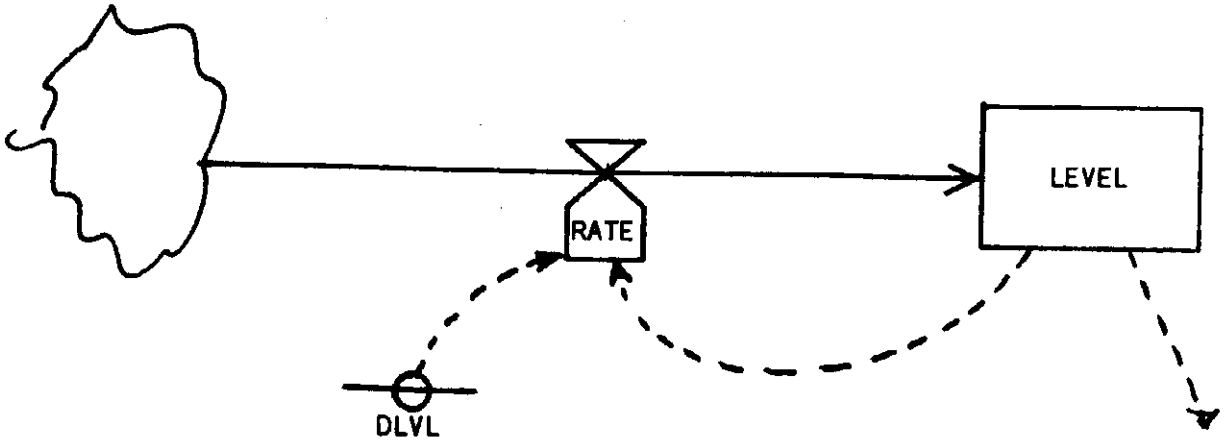


Figure 4.12
First Order Negative Feedback Loop

A. Information DELAY



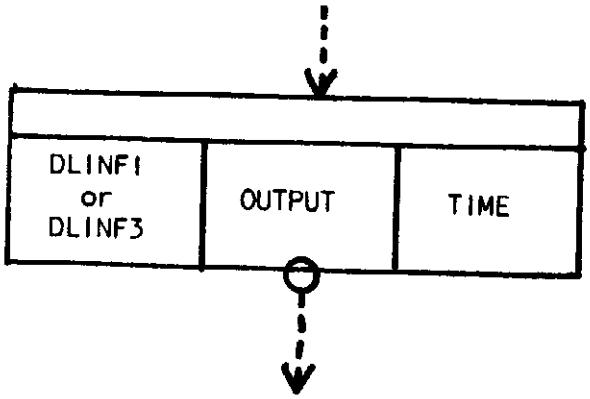
B. No Information DELAY



delay procedure symbol indicates that this is a materials delay. The name in the lower left hand box of the symbol is the name of the delay procedure -- i.e., DELAY1 or DELAY3.* The name of the output variable as specified in the equations as well as the type of variable is shown in the middle box, along with a symbol for the type of equation, usually a RATE VARIABLE.

The delay time, time constant, or the averaging parameter is shown in the lower right hand box. That is, one might wish to delay the output by 15 DT units, by 6 DT units, etc.

The symbol for the information delay procedures, DLINF1 and DLINF3 is shown below with dotted lines in and out:



The information in the lower left hand box shows the name of the procedure, that is DLINF1 or DLINF3. The name in the lower middle box shows the name of the output variable from the procedure as well as the type of variable it is. Usually the output variable from an information delay will be an AUXILIARY variable. The average time delay is given in the lower right side. Figure 4.13 and 4.13.1 show NDTRAN programs that have DLINF1 and DLINF3 reacting to STEP functions. The output is shown on figure 4.14.

---

*Structurally the SMOOTH procedure is similar to a DELAY1 procedure and is generally so identified in the literature. SMOOTH produces a first order exponential average of a physical (conserved) flow.

The NDTRAN language form to call these procedures is shown below:

PROCEDURE NAME	NDTRAN LANGUAGE SYNTAX
DLINF1	EXPND DLINF1(OUT,IN,TRX)
DLINF3	EXPND DLINF3(OUT,IN,TRX)
SMOOTH	EXPND SMOOTH(OUT,IN,SMTM)
DELAY3	EXPND DELAY3(OUT,IN,DEL)
DELAY1	EXPND DELAY1(OUT,IN,DEL)

In all instances, the variables are identified as:

OUT	The output variable from the MACRO procedure.
IN	The input variable to be delayed or smoothed.
TRX	Time to recognize X in time units where X is the level or auxiliary variable whose time is being delayed.
DEL	Delay (in time units) between the input and the output or between IN and OUT.
SMTM	Smoothing (averaging) time.

The Version 1 Language form to call these procedures is shown below:

PROCEDURE NAME	NDTRAN LANGUAGE SYNTAX
DLINF1	EXPND DLINF1(OUT,IN,TRX,INIT)
DLINF3	EXPND DLINF3(OUT,IN,TRX,INIT)
SMOOTH	EXPND SMOOTH(OUT,IN,SMTM,INIT)
DELAY3	EXPND DELAY3(OUT,IN,DEL,INIT)
DELAY1	EXPND DELAY1(OUT,IN,DEL,INIT)

In all instances, the variables are identified as:

OUT	The output variable from the MACRO procedure.
IN	The input variable to be delayed or smoothed.
INIT	The initial value for the variable to be smoothed.
TRX	Time to recognize X in time units where X is the level or auxiliary variable whose time is being delayed.
DEL	Delay (in time units) between the input and the output or between IN and OUT.
SMTM	Smoothing (averaging) time.

The INIT or initial value for the variables to be delayed is unique for the different types of procedures where IN_0 =

PROCEDURES	INITIAL VALUE CALCULATION
SMOOTH	INIT = IN_0 * SMTM
DLINF1	INIT = IN_0
DLINF3	INIT = IN_0
DELAY1	INIT = IN_0 * DEL
DELAY3	INIT = IN_0 * (DEL/3)

Figure 4.13 shows the program and Figure 4.14 the plotted output of a STEP function and the response of DLINF1 and DLINF3 to that STEP Function. In Figure 4.13, line 6 is the output of the step function. As shown in the program and on the plot, the variable INPUT.K has a value of 0 (zero) until period 5, when its value is changed to 1 (one). The value then remains at the level 1 for the rest of the program. The value of the step function is then input to the DLINF1 and DLINF3, with the results being plotted. The first order delay, DLINF1, is plotted with the A symbol and DLINF3 is plotted with the B symbol.

Figure 4.13
Step Function and Delays

```
*****   S O U R C E   L I S T I N G   *****
0001   TITLE STEP RESPONSE OF DLINF1 AND DLINF3
0002   * NARROW
0003   * NOWARN
0004   * NOSTATS
0005   * EULER
0006   A INPUT.K=STEP(1,5)
0007   EXPND DLINF1(OUT1,INPUT,TRX)
0008+  R $R11.KL=(INPUT.K-$L11.K)/TRX
0009+  L $L11.K=INTGRL($R11.JK)
0010+  N $L11=INPUT
0011+  A OUT1.K=$L11.K
0012+  MEND
0013   EXPND DLINF3(OUT3,INPUT,TRX)
0014+  R $R12.KL=(INPUT.K-$L12.K)/(TRX/3)
0015+  L $L12.K=INTGRL($R12.JK)
0016+  N $L12=INPUT
0017+  R $R22.KL=($L12.K-$L22.K)/(TRX/3)
0018+  L $L22.K=INTGRL($R22.JK)
0019+  N $L22=INPUT
0020+  R $R32.KL=($L22.K-$L32.K)/(TRX/3)
0021+  L $L32.K=INTGRL($R32.JK)
0022+  N $L32=INPUT
0023+  A OUT3.K=$L32.K
0024+  MEND
0025   C TRX=15
0026   PARM DT=1
0027   PARM START=0
0028   PARM STOP=50
0029   PARM PLTPER=1
0030   PLOT INPUT=S,OUT1=A,OUT3=B
```

Figure 4.15 is a program that indicates the SMOOTH (delay) procedure, where the input to the procedure is a sine curve defined as SIN(6.28*TIME). On the plot of Figure 4.16 the smoothed SINE wave is plotted with an S symbol while the regular unsmoothed SINE wave is plotted with the U sumbol.

Figure 4.13.1
Step Response of DLINF1 and DLINF3

```
10001   * STEP RESPONSE OF DLINF1 AND DLINF3
10002   *NARROW
10003   *NOSTATS
10004   *EULER
10005   A INPUT.K=STEP(1,5)
10006   EXPND DLINF1(OUT1,INPUT,TRX,INIT)
10007+ R $R11.KL=(INPUT.K-$L11.K)/TRX
10008+ L $L11.K=INTEGRAL($R11.JK)
10009+ N $L11=INIT
10010+ A OUT1.K=$L11.K
10011+ MEND
10012   EXPND DLINF3(OUT3,INPUT,TRX,INIT)
10013+ R $R12.KL=(INPUT.K-$L12.K)/(TRX/3)
10014+ L $L12.K=INTEGRAL($R12.JK)
10015+ N $L12=INIT
10016+ R $R22.KL=($L12.K-$L22.K)/(TRX/3)
10017+ L $L22.K=INTEGRAL($R22.JK)
10018+ N $L22=INIT
10019+ R $R32.KL=($L22.K-$L32.K)/(TRX/3)
10020+ L $L32.K=INTEGRAL($R32.JK)
10021+ N $L32=INIT
10022+ A OUT3.K=$L32.K
10023+ MEND
10024   C TRX=15
10025   C INIT=0
10026   PLOT INPUT=S,OUT1=A,OUT3=B
10027   SPEC DT=1,LENGTH=50,PLTPER=1
```

Figure 4.14
Step Response of DLINF1 and DLINF3
Plotted Output

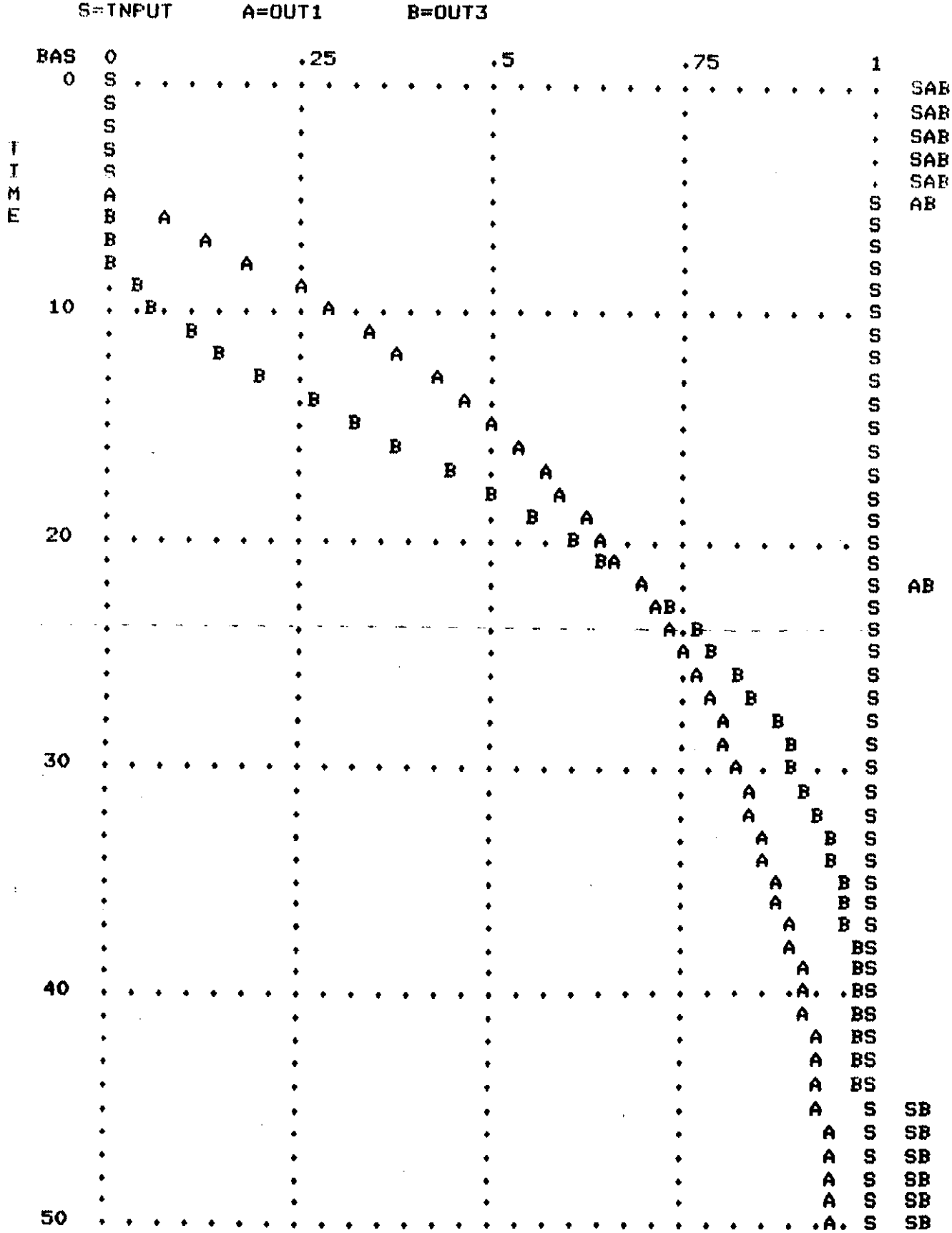


Figure 4.15
SMOOTH Function

```
*****   S O U R C E   L I S T I N G   *****

0001   TITLE TEST OF SMOOTH MACRO
0002   * NARROW
0003   * CHECK
0004   * NOSTATS
0005   C SMTM=.1
0006   A UNSMTH.K=SIN(6.28*TIME.K)
0007   R UNSMTR.KL=UNSMTH.K
0008   EXPND SMOOTH(SMTH,UNSMTR,SMTM)
0009+  L $L11.K=INTGRL(UNSMTR.JK-$R11.JK)
0010+  N $L11=UNSMTR*SMTM
0011+  A SMTH.K=$L11.K/SMTM
0012+  R $R11.KL=SMTH.K
0013+  MEND
0014   PARM DT=.01
0015   PARM START=0
0016   PARM STOP=1
0017   PARM PLTPER=.02
0018   PLOT UNSMTH=U,SMTH=S
PAGE 2     TEST OF SMOOTH MACRO
```


Figure 4.15.1
SMOOTH Function

```
**********   S O U R C E   L I S T I N G   **********

10001   * TEST OF SMOOTH MACRO
10002   *NARROW
10003   *NOSTATS
10004   C SMTM=.1
10005   C INIT=0
10006   A UNSMTH.K=SIN(6.28*TIME.K)
10007   R UNSMTHR.KL=UNSMTH.K
10008   EXPND SMOOTH(SMTH,UNSMTHR,SMTM,INIT)
10009+  L $L11.K=INTEGRAL(UNSMTHR.JK-$R11.JK)
10010+  N $L11=INIT
10011+  A SMTH.K=$L11.K/SMTM
10012+  R $R11.KL=SMTH.K
10013+  MEND
10014   PLOT UNSMTH=U,SMTH=S
10015   SPEC DT=.01,PLTPER=.02,START=0,LENGTH=1
```

Figure 4.16
Test of the SMOOTH Procedure:
Plotted Output

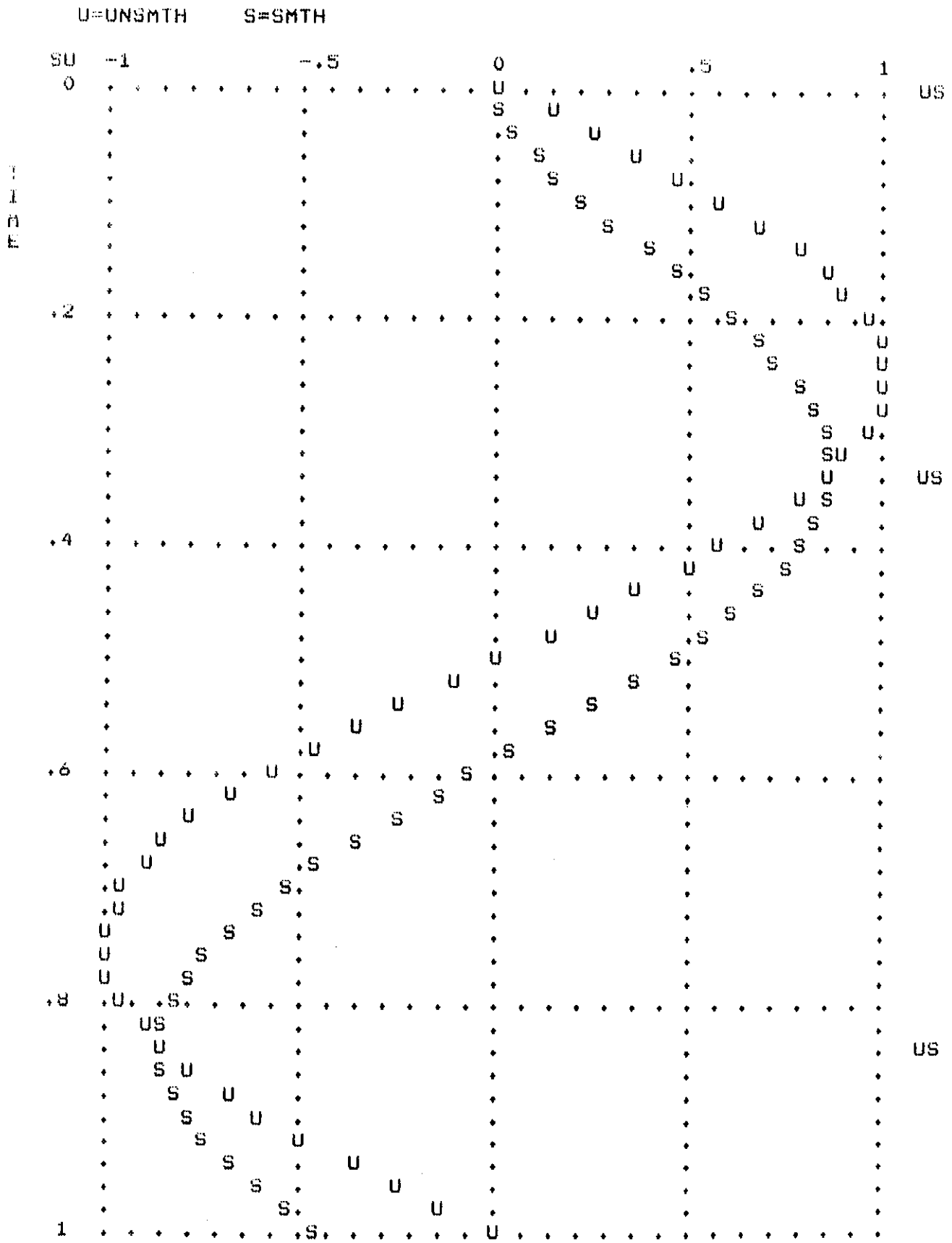


Figure 4.17 and 4.18, respectively, show the program to generate a DELAY1 and DELAY3 and the plotted output from that program. The input to the DELAY functions is a RAMP function. The source program shows

Figure 4.17
DELAY1 and DELAY3 Test with RAMP

```
* * * * *    S O U R C E    L I S T I N G    * * * * *

0001   TITLE DELAY TEST WITH RAMP
0002   * NARROW
0003   * NOSTATS
0004   * CHECK
0005   * EULER
0006   R Q.KL=Q1+RAMP(RS,RT)
0007   C Q1=5
0008   C RS=1
0009   C RT=5
0010   EXPND DELAY3(OUT,Q,DEL)
0011+  L $L11.K=INTGRL(Q.JK-$R11.JK)
0012+  N $L11=Q*DEL/3
0013+  R $R11.KL=$L11.K/(DEL/3)
0014+  L $L21.K=INTGRL($R11.JK-$R21.JK)
0015+  N $L21=Q*DEL/3
0016+  R $R21.KL=$L21.K/(DEL/3)
0017+  L $L31.K=INTGRL($R21.JK-OUT.JK)
0018+  N $L31=Q*DEL/3
0019+  R OUT.KL=$L31.K/(DEL/3)
0020+  MEND
0021   NOTE TEST OF DELAY ONE
0022   EXPND DELAY1(OUT1,Q,DEL)
0023+  L $L12.K=INTGRL(Q.JK-OUT1.JK)
0024+  N $L12=Q*DEL
0025+  R OUT1.KL=$L12.K/DEL
0026+  MEND
0027   C DEL=20
0028   PARM DT=1
0029   PARM START=0
0030   PARM STOP=50
0031   PARM PLTPER=1
0032   PLOT OUT=3(0,60),Q=R,OUT1=1
```

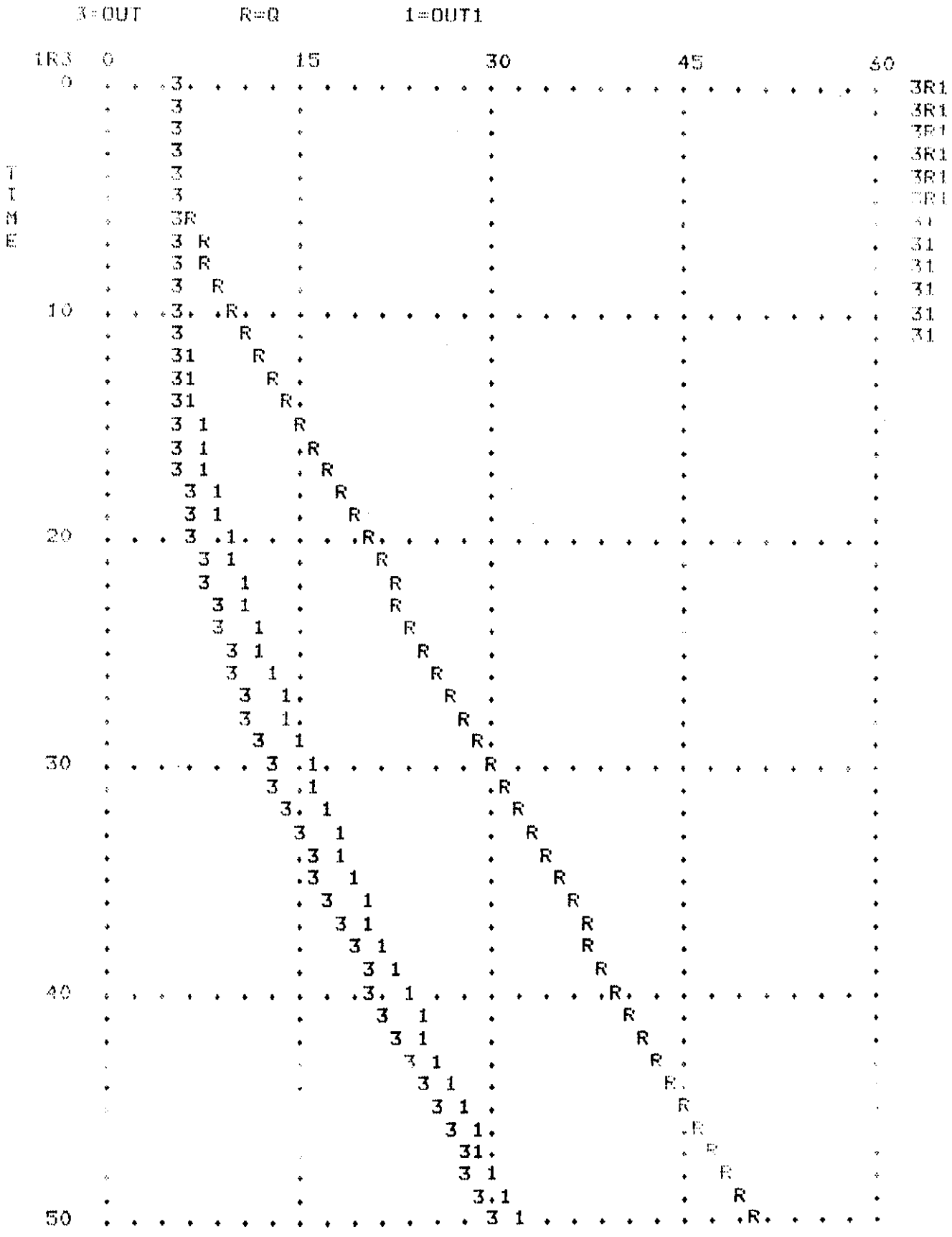
the RAMP Function on line 6, with the RAMP value being input to the DELAY3 on line 10 and to the DELAY1 on line 22. The RAMP value is plotted with an R, the DELAY3 with a 3, and the DELAY1 with a 1 symbol.

Figure 4.17.1
DELAY1 and DELAY3 Test with RAMP

```
**********    S O U R C E    L I S T I N G    **********

10001   * DELAY TEST WITH RAMP
10002   *NARROW
10003   *NOSTATS
10004   *EULER
10005   R Q.KL=Q1+RAMP(RS,RT)
10006   C Q1=5
10007   C RS=1
10008   C RT=5
10009   EXPND DELAY3(OUT,Q,DEL,INIT)
10010+  L $L11.K=INTEGRAL(Q.JK-$R11.JK)
10011+  N $L11=INIT
10012+  R $R11.KL=$L11.K/(DEL/3)
10013+  L $L21.K=INTEGRAL($R11.JK-$R21.JK)
10014+  N $L21=INIT
10015+  R $R21.KL=$L21.K/(DEL/3)
10016+  L $L31.K=INTEGRAL($R21.JK-OUT.JK)
10017+  N $L31=INIT
10018+  R OUT.KL=$L31.K/(DEL/3)
10019+  MEND
10020   NOTE TEST OF DELAY 1
10021   EXPND DELAY1(OUT1,Q,DEL,INIT1)
10022+  L $L12.K=INTEGRAL(Q.JK-OUT1.JK)
10023+  N $L12=INIT1
10024+  R OUT1.KL=$L12.K/DEL
10025+  MEND
10026   C INIT1=100
10027   C DEL=20
10028   C INIT=33.33
10029   PLOT OUT=3(0,60),Q=R,OUT1=1
10030   SPEC DT=1,START=0,STOP=50,PLTPER=1
```

The RAMP function is defined on line 10005. The RAMP value is input to the DELAY3 in line 10009 and to the DELAY1 in line 10021.

Figure 4.18
DELAY1 and DELAY3 Test with RAMP



It is possible using combinations of functions or combinations of procedures to obtain results not available from any existing function or procedure. Let us say that for some reason it was desirable to show a second order DELAY because of some circumstance. This could be done using a DELAY1 to delay a regularly programmed first order negative feedback loop. The first order negative feedback loop effect would be expected to approach a given goal, as for instance a company attempting to attain a desired level of inventory. If there were some delay in receiving (or perhaps ordering) goods there would be an overshoot and oscillation effect with a final stabilization. This type of combination of functions or procedures may be used any time that a unique effect is desired. The combinations of functions or procedures may be usually employed with all of the functions in NDTRAN.

Chapter 5

NDTRAN: Print, Plot and Functions

This chapter seems dictated rather by considerations of length than considerations of aesthetics. The three topics covered here will be: Printed Output, Plotted Output and Functions.

A. Printed and Plotted Output

1. PRINT Statement

The PRINT statement establishes the variables that are desired to be printed as output. The PRINT command is:(recall that all NDTRAN language statements begin in COLUMN 1):

The title to the program will appear automatically as a header to the printed output. If you desire additional information as a title to the printed output, simply place a NOTE card with some appropriate information immediately preceeding the PRINT statement. If there are several blank and non-blank NOTE cards immediately preceeding the PRINT statement, NDTRAN takes the first non-blank NOTE card as the additional information for the title to the printed output.

In the source program of Figure 5.1 we note that the title to the table output is the information in the title control card followed by the information in the NOTE card that immediately preceeds the PRINT statement. The printed output resulting from Figure 5.1 is shown in Figure 5.2.

Figure 5.1
Rabbit Model: Population

```
PAGE 1    RABBIT MODEL                                    (C) 1978 UND

     * * * * *    S O U R C E    L I S T I N G    * * * * *

0001  TITLE RABBIT MODEL
0002  * EULER
0003  * XREF
0004  * NARROW
0005  L RAB.K=INTGRL(RB.JK-RD.JK)
0006  N RAB=150
0007  R RB.KL=RAB.K*FERT
0008  R RD.KL=RAB.K*DEAT
0009  C FERT=.06
0010  C DEAT=.04
0011  PARM DT=.08
0012  PARM START=0
0013  PARM STOP=100
0014  PARM PLTPER=10
0015  PARM PRTPER=10
0016  NOTE RABBIT MODEL PRINTED DATA
0017  PRINT RAB,RB,RD
0018  NOTE RABBIT MODEL PLOTTED DATA
0019  PLOT RAB,RB,RD
```

Figure 5.1.1
Rabbit Model: Population

```
* RABBIT MODEL
*EULER
*XREF
*NARROW
L RAB.K=INTEGRAL(RB.JK-RD.JK)
N RAB=150
R RB.KL=RAB.K*FERT
R RD.KL=RAB.K*DEAT
C FERT=.06,DEAT=.04
SPEC DT=.08,START=0,STOP=100,PRTPER=10
NOTE RABBIT MODEL PRINTED DATA
PRINT RAB,RB,RD
```

Figure 5.2
Rabbit Population
Printed Data

RABBIT MODEL - RABBIT MODEL PRINTED DATA (C) 1978 UND

TIME E+00	RAB E+03	RB E+00	RD E+00
0.00	0.1500	9.000	6.000
10.00	0.1832	10.991	7.327
20.00	0.2237	13.422	8.948
30.00	0.2732	16.391	10.927
40.00	0.3336	20.017	13.345
50.00	0.4074	24.445	16.297
60.00	0.4975	29.852	19.902
70.00	0.6076	36.456	24.304
80.00	0.7420	44.520	29.680
90.00	0.9061	54.369	36.246
100.00	1.1066	66.395	44.264

The reader should note that the information printed in the table appears in scientific notation or E-Format. The information may appear in either F-Format or E-Format. Should the characteristic of any number become larger than 4 or less than -1 the information printed will change automatically to an E-Format. The accuracy of output is 5 digits. Thus, scaling is set by the largest number for each variable. The reader may wish to experiment a bit with a simple model to determine the way that NDTRAN scales output.

> In NDTRAN version 1 some of the variables may be in scientific notation , and other not.
>
> Version 1 handles the form of the printed output somewhat differently. The accuracy is the same.

When the wide output option is used, NDTRAN will print a maximum of 10 variables per print statement across the page. When the narrow option is used, NDTRAN will print a maximum of 5 variables. The TIME variable is printed in both instances. NDTRAN suppresses printing of all variables on a PRINT statement beyond the fifth variable for the narrow option and beyond the tenth variable for the wide option.

2. PLOT Statement

The NDTRAN user may obtain plots of variables against time, called the TIME PLOT, or to generate a variable against variable plot. The latter form is generally considered to be a sensitivity plot where one variable is a goal variable and the other is a policy variable and one desires to determine whether a goal variable is sensitive to policy changes.

Time PLOT

The PLOT statement establishes the variables that the programmer wishes to display as a graph, with time as the independent variable and some other variables as the ordinate. The PLOT statement has several possible forms, a number of which are noted below:

i. scaling by first variable: PLOT RD,RB,RAB

ii. automatic scaling : PLOT RD/RB/RAB

iii. scaling by definition : PLOT RD(0,45)/RB(0,70)/RAB(0,1200)

iv. scaling by definition and scaling by default : PLOT RAB(0,1200)/RB/RD

v. independent scaling:

1) PLOT RAB(0,1200),X/RD,RB/Z

2) PLOT RAB(0,1200),X/RD,RB(0,40)/Z

3) PLOT RAB(0,*)

In all instances above we have allowed the PLOT symbol-- i.e., that which appears on the graph -- to default to the symbols provided by NDTRAN. It is possible to assign separate plot symbols as indicated below:

vi. PLOT RAB=1(0,1200)

vii. PLOT RAB=+(0,*)

Let us take these few possibilities one at a time and explain them fully before showing a few examples of the PLOT statement options.

i. Automatic scaling on first variable, PLOT symbol default.

PLOT RAB,RD,RB

Note first that the order of the variables was changed in this plot statement as compared with the illustration in i. on the previous page. Whichever variable is placed first, when the variables are separated by commas will be the variable on which the other variables are scaled. In the case illustrated the programmer desires to plot all three variables on variable RAB and to use the default plot symbols. On a PLOT statement when the variables are separated by commas then all variables on the PLOT statement are scaled on the same scale as the first variable. To assign PLOT symbols to the three variables under the same scaling conditions, the statement would be:

PLOT RAB=1,RD=2,RB=*

ii. Automatic Scaling, PLOT symbol default.

PLOT RAB/RD/RB

In this instance the programmer desires to plot three variables and the upper and lower limit of the variables are not known. It is desired to plot each of the variables on independent scales (independent

of each other) hence the slash marks "/" must be used to separate the variables to be plotted. NDTRAN will automatically assign a PLOT symbol for each variable and will automatically fully scale each variable on its own scale basis. If the programmer desires to PLOT the three variables as indicated here but to assign PLOT symbols , the form used would be:

PLOT RAB=1/RD=*/RB=A

The same automatic and seperate scaling would be used, but the variables would be plotted on the graph with the symbols, respectively as shown to the right of each equals sign.

iii. Scaling by definition, PLOT symbol default.

PLOT RAB(0,1200)/RD(0,45)/RB(0,70)

In this instance the programmer desires to specify the scale for the plotting of each variable. This is most useful in any run after the first run where the programmer has some idea of the range and the starting value for each variable. Should the programmer, on the initial run, know the initial or lower value of a variable, but not the upper value it may be desirable to establish the lower plot limit but to allow automatic scaling for the upper limit. This would be done as follows:

PLOT RAB(o,*)/RD(0,*)/RB(0,*)

In this instance NDTRAN will provide the plot symbol and will automatically scale the variables, independently from the given lower limit to the calculated upper limit. Again PLOT symbols may be supplied as follows:

PLOT RAB=1(0,*)/RD=A(0,*)/RB=B(0,*)

iv. Scaling by definition and by default.

```
PLOT RAB(0,1200)/RB/RD
```

In this instance NDTRAN will scale the variable RAB as indicated between the lower limit of zero and the upper limit of 1200. However, each of the variables RB and RD will be scaled automatically and independently of each other and of RAB.

v. Independent scaling.

It is possible to give three illustrations of the kind of independent scaling that is possible with NDTRAN. In the first instance:

```
PLOT RAB(0,1200),X/RB,RD/Z
```

In this case variable RAB will be plotted on the scale from zero to 1200. Variable X will be plotted on the same scale as RAB. Variables RAB and X will be independent of the scaling of all other variables in the PLOT statement. Variables RD and RB will also be independent of all other variables shown, but variable RD will be plotted on the same scale as RB. Variable Z will be independently scaled.

The second possibility is shown below:

```
PLOT RAB(0,1200),X/RD,RB(0,40)/Z
```

This is the same as the previous example except for the middle variables RD and RB. In this instance the scale limits for RB are shown and the variable RD will be scaled on the same basis as RB.

The third possibility is:

```
PLOT RAB(0,*)
```

This is the most useful kind of scaling for a first or preliminary run of the program where the programmer knows the lower limit, but is not clear as to the upper limit of the variable being calculated.

In this instance NDTRAN will accept the zero as the lower limit, but will provide automatic scaling on the upper limit. If the programmer desires to use this but wishes to assign a plot symbol, then:

PLOT RAB=1(0,*)

Let us take a few examples of these plot possibilities. The program used was as shown in Figure 5.1, with the exception that the PLOT statement options vary. The plot statement options used are indicated for each table. In some instances the FERT and DEAT constants were changed somewhat to make the distinction between the variables more distinct on the plotted output.

Figure 5.3 is an illustration of scaling by the first variable, or where the delimiters in the PLOT statement are commas. This illustrates two common errors in using any PLOT graph. First, the PLOT period chosen was too large, so that the plot is relatively small, hence distorted. The second problem is that since the variables are plotted on the basis of the scaling of the first variable (in this case the variable with the smallest range) not all of the points for all of the variables are plotted. Figure 5.4 shows the same method of scaling with a more appropriate plot size and arrangement of variables.

Figure 5.3
Plot Scaling:First Variable

```
PLOT RD,RB,RAB                          PLTPER=5     STOP=50
FERT=.08                                DEAT=.02
```

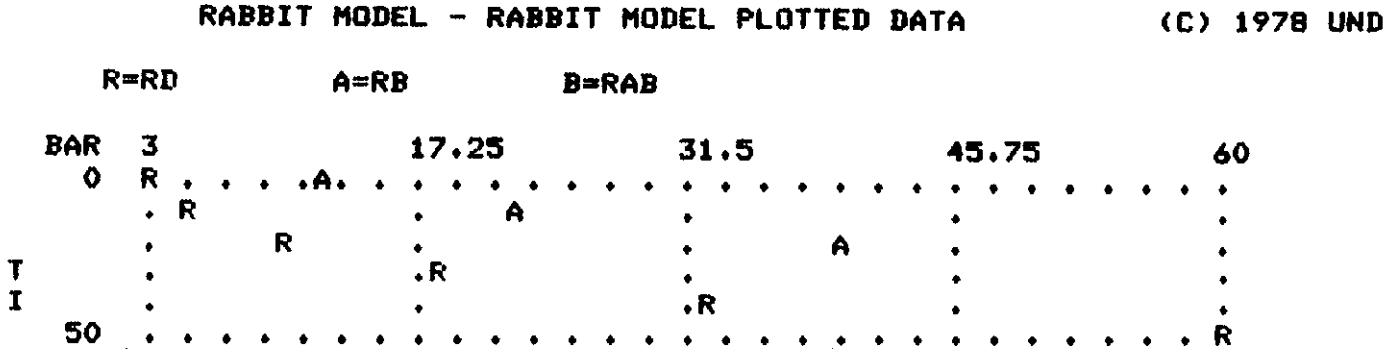
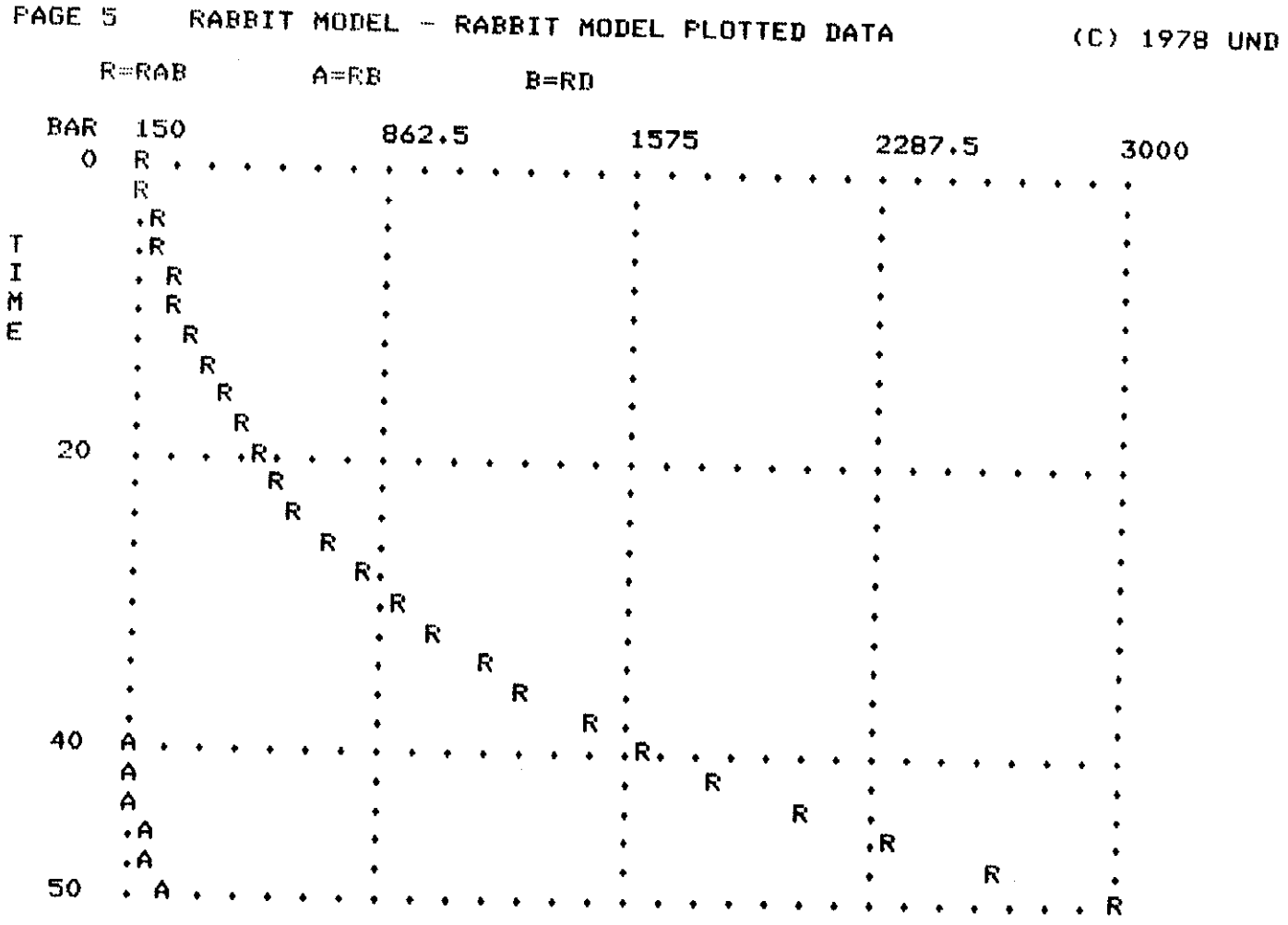


Figure 5.4
Plot Scaling: First Variable

```
PLOT RAB,RB,RD                          PLTPER=2      STOP=50
FERT=.08                                DEAT=.02
```

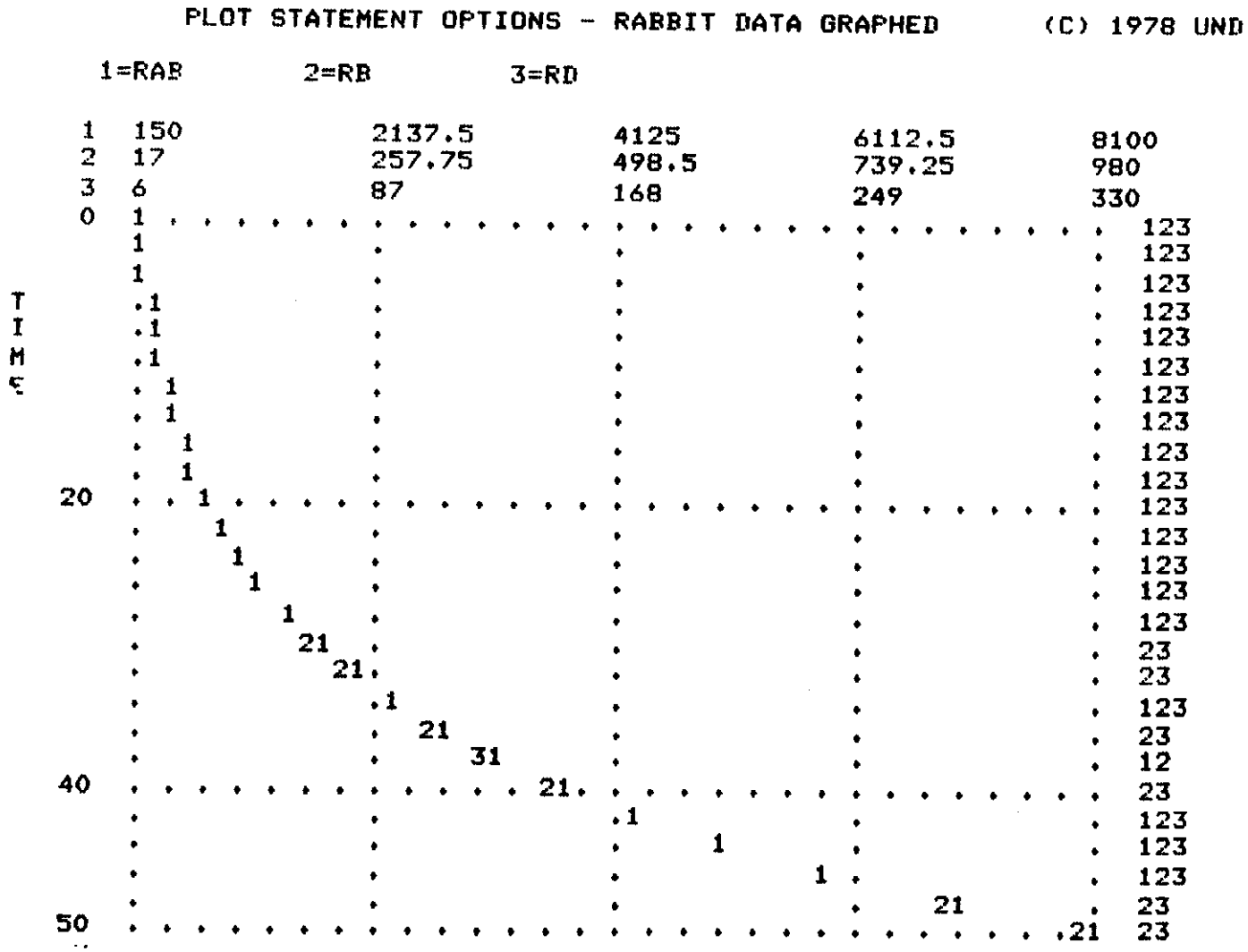


Even in this instance the variable RD did not appear on the graph since it did not yet (by TIME equals 50) reach a minimum value of 150, which was the minimum value for variable RAB. Variable RB reached the minimum value by period 40.

Perhaps the most common method for scaling variables is to scale each variable independently as in Figure 5.5. In this instance the programmer

Figure 5.5
PLOT Scaling: Independent Scaling

```
PLOT RAB=1/RB=2/RD=3                    PLTPER=2        STOP=50
FERT=.12                                DEAT=.04
```



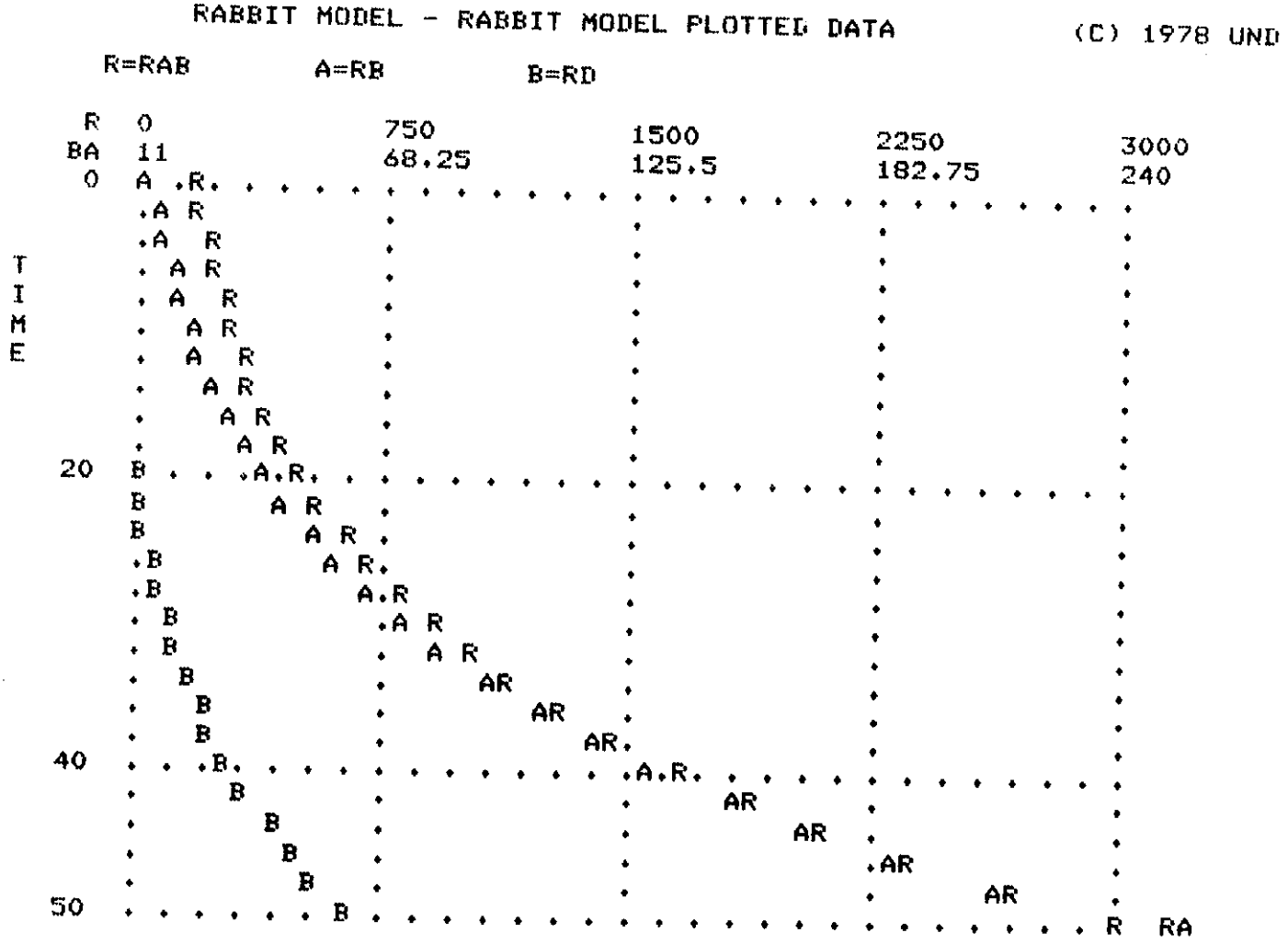
In this type of plot the scale for each variable is indicated at the top of the graph, and the assigned PLOT symbols are also indicated. There is some danger in using this type of plotting, as shown above in as much as the variables may be such that they all plot in the same position on the graph, although scaled independently. The variables are identified by their plot symbols at the top of the graph. The fact that many of the points for the different variables OVERLAY is shown by the numbers at the right side of the graph. Only a 1 symbol is plotted but the right side of the graph indicates that variable symbols 1 and 2 and 3 overlay for that specific point.

Another option that may be used to independently scale one variable and to scale the other two together is shown in Figure 5.6. The first variable RAB is given a lower limit and the upper limit is scaled by NDTRAN with the second and third variables scaled on the second variable RB. Again one common point shows on the right side of the graph.

The thing that the programmer using NDTRAN must remember is that the purpose of a graph (plotted output) is to show the relationships of the data variables as well as their magnitude as accurately as possible. Therefore the perspective of the output is important. The programmer may wish to make a number of different kinds of PLOT runs in order to obtain precisely the most appropriate plot for the variables to be presented. Scaling is always shown at the top of the graph. PLOT points overlaid are shown on the right side of the graph. For maximum accuracy use the plotted output with the tables for the same variables.

Figure 5.6
Plot Scaling: Split Scaling

```
PLOT RAB(0,*)/RB,RD          STOP=50          PLTPER=2
FERT=.08                     DEAT=.02
```



Independent Variable PLOT

Another feature of NDTRAN plotting capability is the ability to generate variable against variable plots. These could be used in sensitivity testing of models, that is comparing what happens to a goal variable if a policy variable is changed. All of the conventions noted above apply to the variable against variable plotting. The only difference is that following the primary variables that are to be plotted on the Y AXIS, a double slash is placed in the PLOT card. Following this the independent variable or the variable to be plotted on the X AXIS is given. The form is:

PLOT RAB//RB

The other options are retained as noted above. This is illustrated in Figure 5.7.

It is easy to get additional plots on the same graph. Figure 5.8 shows that the programmer desires to plot RB against RAB and RD against RAB. In this instance as in many cases the two sets of points exactly overaly. THIS OVERLAY IS NOT SHOWN ON THE RIGHT SIDE OF THE VARIABLE AGAINST VARIABLE PLOT AS IT IS IN THE TIME PLOT. Figure 5.9 shows the same plotted output but with the variable RD being plotted on the scale of RB (instead of independently as in Figure 5.8). The variable against variable plot in NDTRAN is a standard size as shown in the figures and operates on the NARROW option basis. If one should increase the point density by changing the PLTPER parameter, the size of the plots remains the same but the density of points changes.

Figure 5.7
Plotting: Variable by Variable

PLOT RAB//RB PLTPER=2

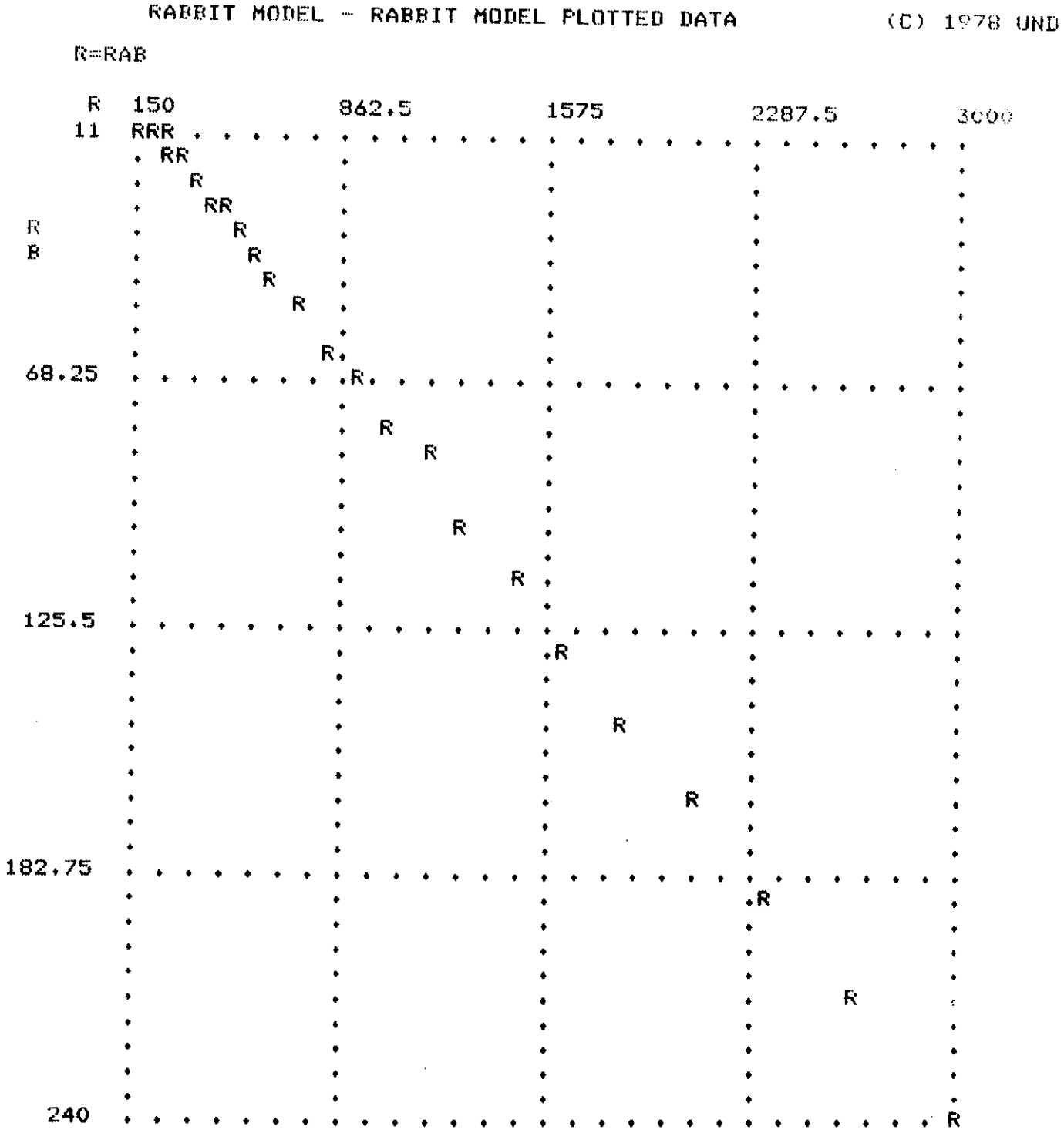


Figure 5.8
Plotting: Variable by Variable

PLOT RB=1/RD=2//RAB PLTPER=2

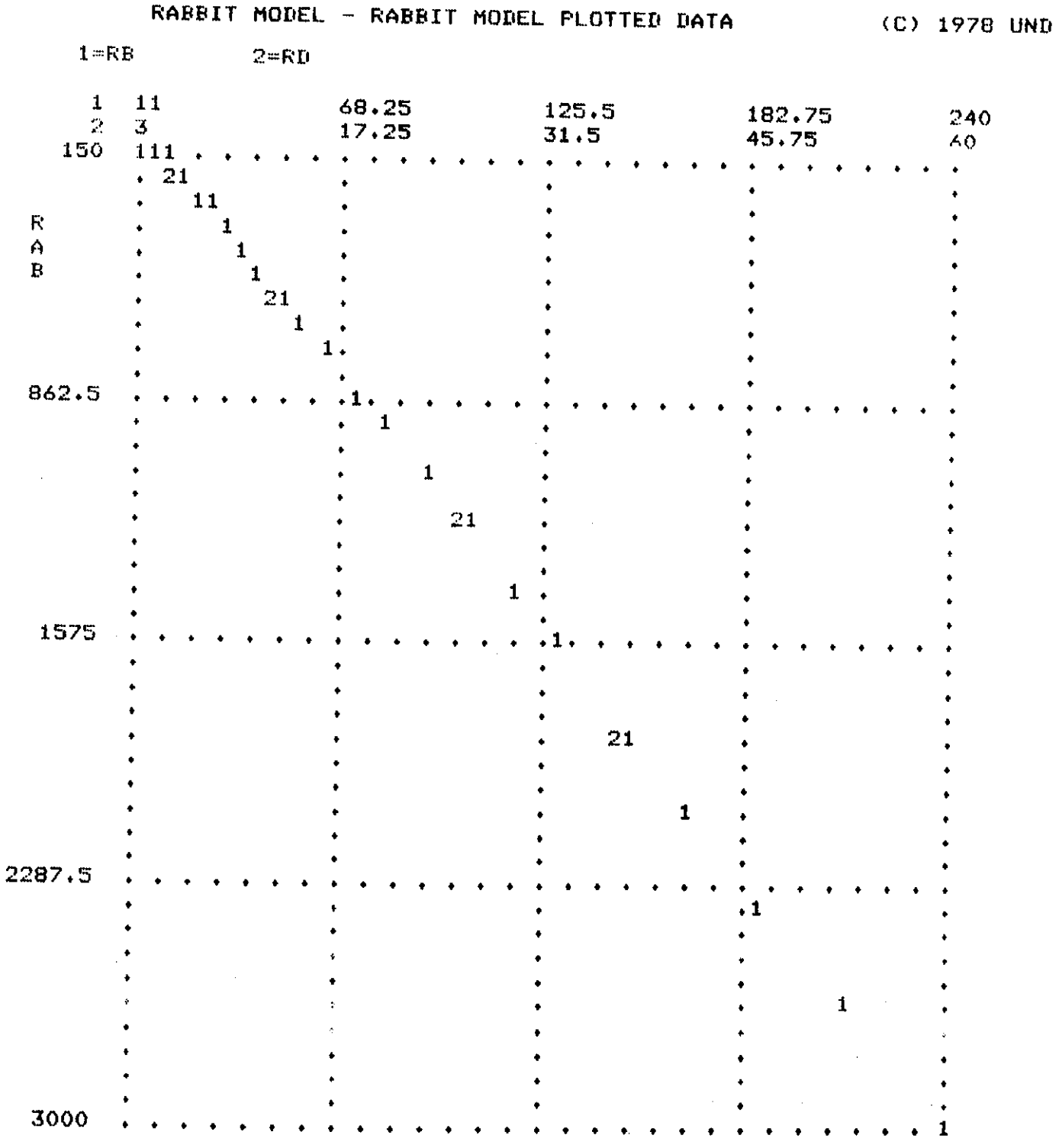
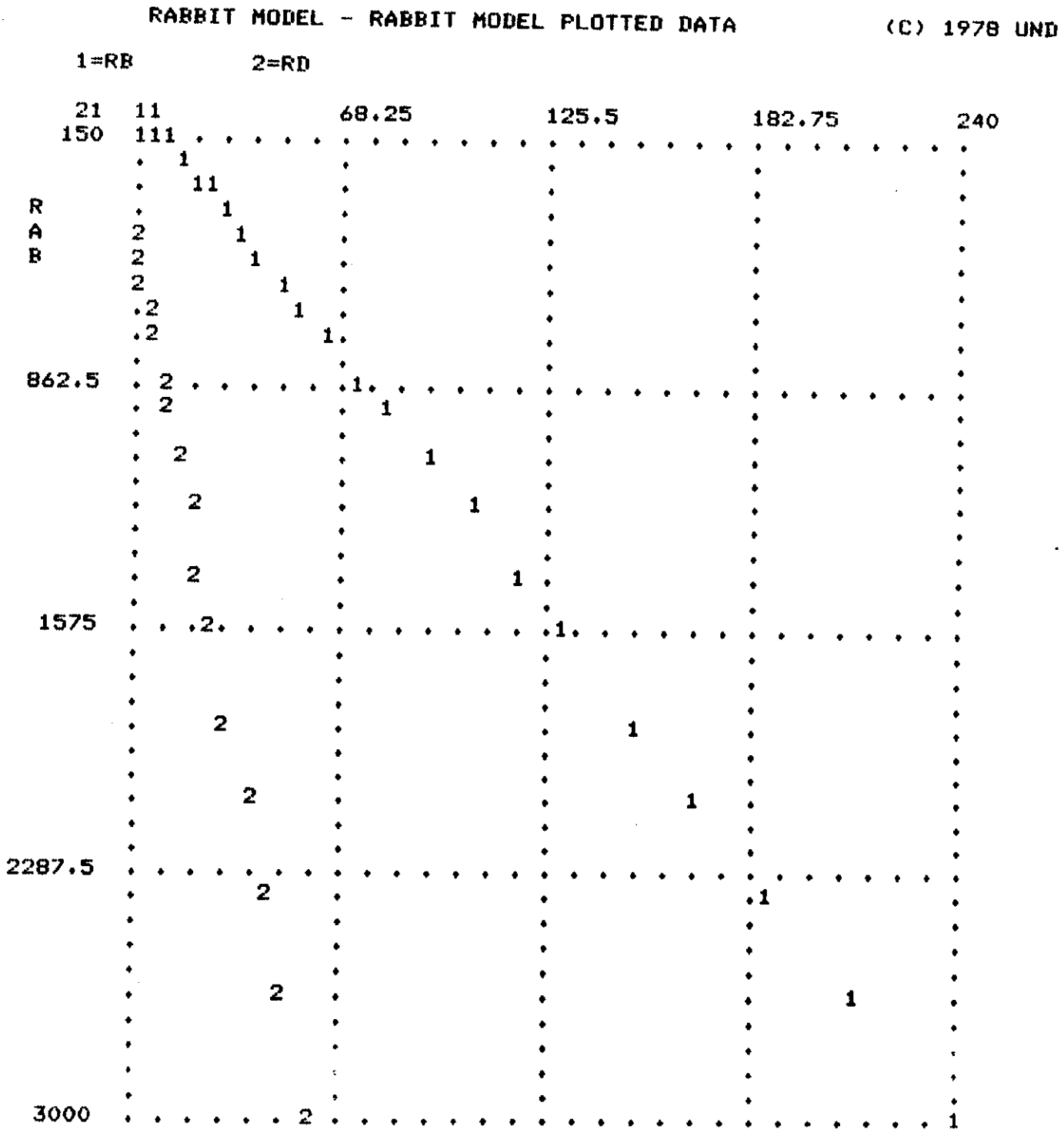


Figure 5.9
Plotting: Variable by Variable

```
PLOT RB=1,RD=2//RAB                                    PLTPER=1
```



Rerun and Comparative

It is possible to have NDTRAN programs make a number of different runs in the same program, using the RERUN card. Four parts of a program may be changed in any or each RERUN from the main program or from a previous rerun:

1) constants;
2) tables;
3) DT size;
4) Integration method.

The problem shown below indicates the standard rabbit model shown elsewhere but with two rerun statements. In the first rerun the value of the DEAT variable is changed. In the second rerun the integration method is changed. The changed integration method is to determine what occurs as the model is changed from a discrete (stochastic) model to a continuous model. Whenever a variable is change from the main program in any rerun, or from one rerun to another rerun the changes remain in effect for all subsequent reruns.

The following comments refer to a program shown in Figure 5.10. The program is set up with whatever control options are desired and with the program specified in any way desired by the modeler. All of the PRINT and PLOT statements come at the end of the main program and before any of the reruns. For the program, we desire to PRINT and PLOT the variables RAB and RB:

1) Main Program	PRINT RAB,RB,RD PLOT RAB/RB
2) First Rerun	PRINT RAB.2,RB.2,RD.2 PLOT RAB.2/RB.2
3) Second Rerun	PRINT RAB.3,RB.3,RD.3 PLOT RAB.3/RB.3

Finally for a summary of all runs we must select a variable or variables which are to be compared across all runs. One PRINT and PLOT card must be specified for each variable to be compared as follows:

```
PLOT RAB.*
PRINT RD.*
PLOT RB.*
...
```

Any changes that you desire to make in a Rerun are made following the RERUN statement as shown in the program below in Figure 5.10.

Figure 5.10
RERUN and COMPARATIVE PLOT and PRINT

```
      PLOT STATEMENT OPTIONS                                    (C) 1978 UND
*****    S O U R C E    L I S T I N G    *****

0001  TITLE PLOT STATEMENT OPTIONS
0002  * NARROW
0003  * CHECK
0004  * EULER
0005  L RAB.K=INTGRL(RB.JK-RD.JK)
0006  N RAB=150
0007  R RB.KL=RAB.K*FERT
0008  R RD.KL=RAB.K*DEAT
0009  C FERT=.12
0010  C DEAT=.04
0011  PARM DT=.08
0012  PARM STOP=50
0013  PARM PLTPER=2
0014  PARM PRTPER=2
0015  PRINT RAB,RB,RD
0016  PRINT RAB.2,RB.2,RD.2
0017  PRINT RAB.3,RB.3,RD.3
0018  PRINT RD.*
0019  PLOT RAB(0,*),RB,RD
0020  PLOT RAB.2/RB.2
0021  PLOT RAB.3/RB.3
0022  PLOT RAB.*
0023  PLOT RB,RD//RAB
0024  NOTE RABBIT DATA GRAPHED
0025  RERUN  CHANGED DEATH RATE
0026  C DEAT=.08
0027  RERUN CHANGED DEATH RATE
0028  * RKINT
```

print program
print first rerun
print second rerun
  comparative print on variable RD.
plot program
plot first rerun
plot second rerun
  comparative plot on variable RAB.
    variable against variable plot in program

Since both the PLOT and PRINT options have already been discussed we shall consider here primarily the comparative PRINT and PLOT statements. The following points should be kept in mind:

1) One PRINT and/or PLOT card must be placed for each rerun. These PRINT and PLOT statements must be located in the MAIN program before a RERUN statement is encountered.

2) The PRINT statement on line 18 is a comparative PRINT in that it will print out the value of the indicated variable over the main program and all subsequent reruns, incorporating the effect of the changes on that variable. The result is as shown in Figure 5.11.

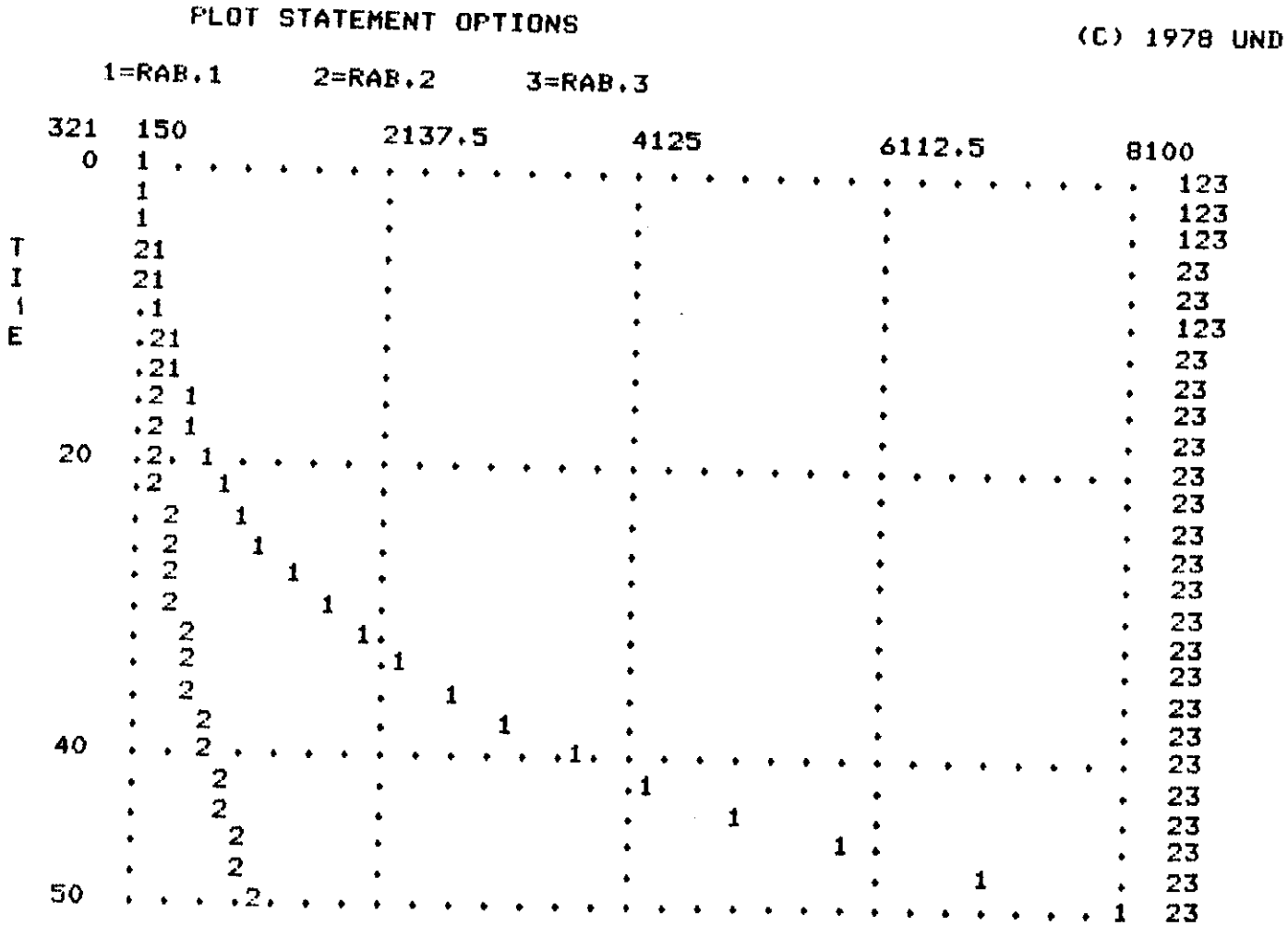
Figure 5.11
COMPARATIVE PRINT

PLOT STATEMENT OPTIONS (C) 1978 UND

TIME	RD.1	RD.2	RD.3
0.000	6.00	12.000	12.000
2.000	7.04	12.998	12.999
4.000	8.25	14.079	14.082
6.000	9.68	15.249	15.255
8.000	11.36	16.517	16.526
10.000	13.32	17.890	17.902
12.000	15.62	19.378	19.393
14.000	18.32	20.989	21.008
16.000	21.49	22.735	22.758
18.000	25.21	24.625	24.653
20.000	29.57	26.672	26.706
22.000	34.68	28.890	28.931
24.000	40.68	31.292	31.340
26.000	47.71	33.894	33.951
28.000	55.96	36.713	36.778
30.000	65.64	39.765	39.841
32.000	76.98	43.072	43.160
34.000	90.30	46.653	46.754
36.000	105.91	50.532	50.648
38.000	124.22	54.734	54.867
40.000	145.70	59.285	59.436
42.000	170.90	64.214	64.387
44.000	200.45	69.554	69.749
46.000	235.11	75.337	75.558
48.000	275.76	81.601	81.852
50.000	323.44	88.386	88.669

3) Line 22:the COMPARATIVE PLOT statement for the program, indicates that the variable RAB is to be plotted on a graph with the value of the variable calculated in the main program and in each of the rerun programs. The COMPARATIVE PLOT diagram is indicated in Figure 5.12.

Figure 5.12
COMPARATIVE PLOT OPTION



Again, the RERUN simply causes the entire program to be re-executed but with the modified values as indicated after each rerun statement. For instance line 26 in Figure 5.10 causes the program to be re-executed but with the DEAT variable changed from a value of .04 to a value of .08. Line 28 causes the last rerun to be executed with the Runge-Kutta Integration procedure instead of the Euler procedure (Line 4) which was used for the first two program runs. All changes are permanent for successive reruns unless subsequently modified. Thus DEAT had the value .08 for the last two runs. As a caveat it is usually a good idea to change only one variable at a time in successive reruns of a program, or it will be unlikely that the true impact of a change can be determined. Finally, we should note that if a comparative print or plot was not required then lines 16 through 22 would not be required.

Version 1 of NDTRAN does not have the COMPARATIVE PRINT or PLOT capability. Version 1 of NDTRAN in the RERUN mode may change the constants, tables or DT size BUT MAY NOT CHANGE INTEGRATION METHODS. Any changes in a RERUN remain for all subsequent reruns of the program. Finally, only a single PRINT or PLOT statement is required for the output from the main program and subsequent reruns. Figure 5.10.1 below shows an NDTRAN Version 1 program with a RERUN and the PRINT and PLOT statement setup for the main program and RERUN.

Figure 5.10.1
NDTRAN Version 1
RERUN Options

```
**********    S O U R C E    L I S T I N G    **********

10001   * RERUN STATEMENT OPTIONS
10002   *NOSTATS
10003   *NARROW
10004   *EULER
10005   L RAB.K=INTEGRAL(RB.JK-RD.JK)
10006   N RAB=150
10007   R RB.KL=RAB.K*FERT
10008   R RD.KL=RAB.K*DEAT
10009   C FERT=.06,DEAT=.04
10010   SPEC DT=.08,START=0,STOP=100,PRTPER=10,PLTPER=10
10011   NOTE RABBIT POPULATION
10012   PRINT RD,RB,RAB
10013   NOTE RABBIT POPULATION PLOT1
10014   PLOT RD(0,8200)/RB(0,8200)/RAB(0,8200)
10015   RERUN THIS RUN CHANGES THE RABBIT FERTILITY RATE
10016   C FERT=.08
```

B.Functions

In modeling and simulation the role of FUNCTIONS is quite important and as such are specified modifications to the normal flow of information and/or goods and services and people. A function in a simulation language (interpreter) like NDTRAN might be:

1) predefined and part of the interpreter. These types of FUNCTIONS include PULSE, NOISE, NORMRN, STEP,SINE, COSINE, and the like;
2) defined by the user from an arbitrary graph and implemented by the special function called TABLE.

A FUNCTION may appear in all equation types except level equations. Initial values may be computed using functions so long as only constants and other initial values are used as arguments. Most frequently, however, functions appear in AUXILIARY or RATE equations; always appearing on the right side of the equal sign.

1) PULSE Function

A Pulse function is theoretically designed to excite a given system at a specific point or points in time with a defined amplitude. It can be used in other ways. The form of the Pulse Function is as follows:

PULSE(VAR1,VAR2,VAR3,VAR4)

VAR1 - the height of the pulse function;

VAR2 - the width of the pulse function expressed as DT units of time for the model;

VAR3 - the period of the model in which the pulse function is first activated.

VAR4 - the interval of time between pulses. In the example shown in Figure 5.13 , VAR4 was set to five to indicate that the pulse would reactivated every sixth period.

The arguments may be input to the pulse either as constants or as variables.

Let us say that you have a modeling problem where it is necessary to inhibit the active entry of one variable in the model until several DT time periods have elapsed, and to have that variable enter at an initial value other than zero. Figure 5.13 shows such an example program. The output is printed in Figures 5.14 and 5.15. In this example pulse is activated after five periods, remains inactive for five more periods then is activated again. VAR4 would be set to zero if the pulse were to be activated only one time.

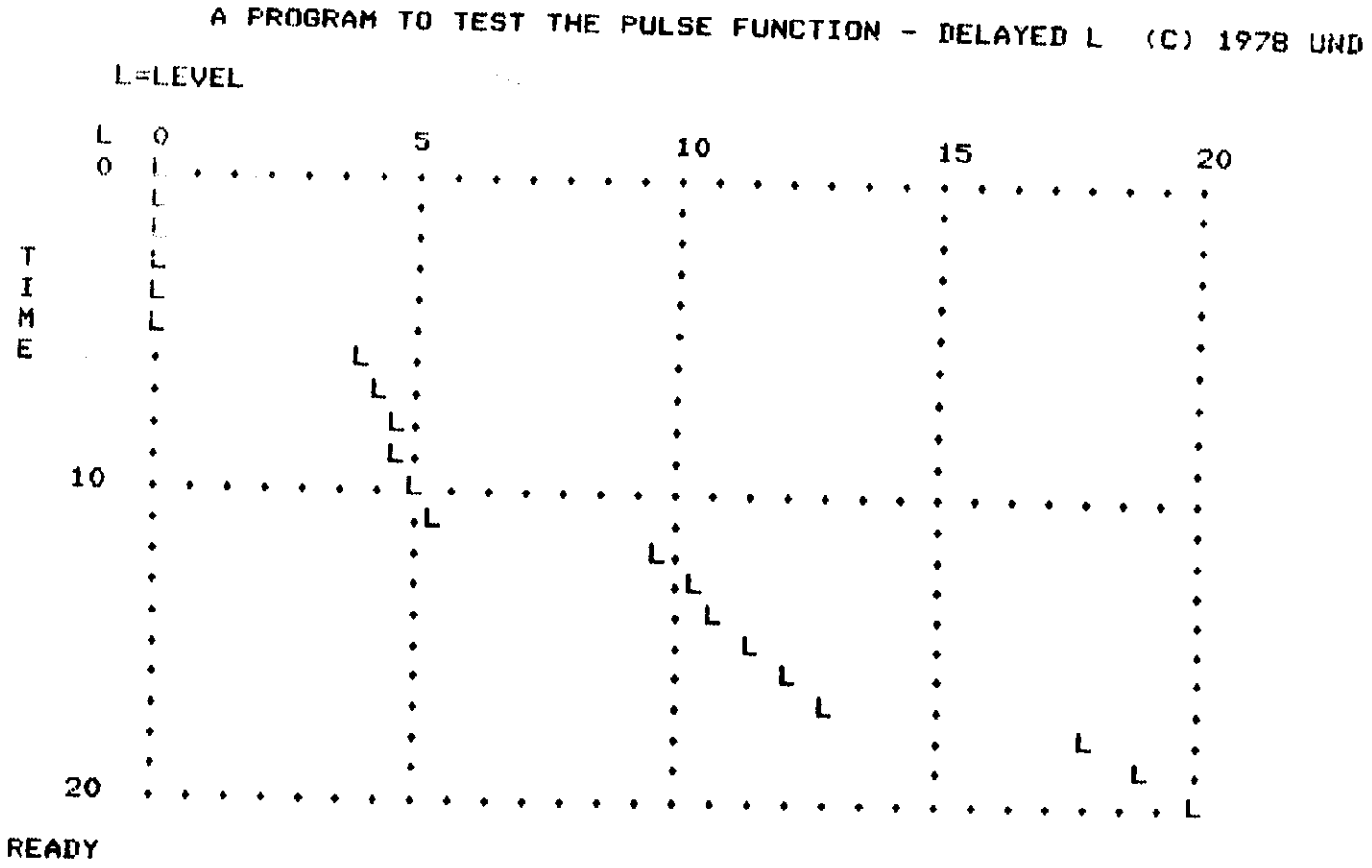
Figure 5.13
Pulse Function
Multiple Reactions

```
PAGE 1     A PROGRAM TO TEST THE PULSE FUNCTION                    (C) 1978 UND

     * * * * *    S O U R C E    L I S T I N G    * * * * *

0001  TITLE A PROGRAM TO TEST THE PULSE FUNCTION
0002  * EULER
0003  * NARROW
0004  * NOSTATS
0005  L LEVEL.K=INTGRL(R.JK)
0006  N LEVEL=0
0007  R R.KL=LEVEL.K*.06+PULSE(4,1,5,5)
0008  PARM DT=1
0009  PARM START=0
0010  PARM STOP=20
0011  PARM PLTPER=1
0012  PARM PRTPER=1
0013  NOTE DELAYED LEVEL VARIABLE
0014  PRINT LEVEL
0015  NOTE DELAYED LEVEL VARIABLE
0016  PLOT LEVEL
```

Figure 5.14
Pulse Function
Printed Output

TIME	LEVEL
0.000	0.000
1.000	0.000
2.000	0.000
3.000	0.000
4.000	0.000
5.000	0.000
6.000	4.000
7.000	4.240
8.000	4.494
9.000	4.764
10.000	5.050
11.000	5.353
12.000	9.674
13.000	10.255
14.000	10.870
15.000	11.522
16.000	12.213
17.000	12.946
18.000	17.723
19.000	18.786
20.000	19.913

Figure 5.15
Pulse Function
Plotted Output



2) CLIP Function

The Clip function can be used to:

a) limit the value of a variable ; or

b) provide a method for shifting variable values.

The general form of the Clip Function is:

CLIP(VALU2,VALU1,CNTL,TEST)

VALU2 - the value assigned to the dependent variable if the value of the CNTL variable is greater than or equal to the value of the TEST variable.

VALU1 - the value assigned to the dependent variable if the value of the CNTL variable is less than the value of the TEST variable.

CNTL - the control variable. It is common to have the control variable be the TIME variable. The Clip function may be controlled by a variable reflecting magnitude rather than time over a model run.

TEST - the reference or comparison variable to determine whether VALU1 or VALU2 will be used.

VALU1 and VALU2 may be table variables or constants. The next illustrations show the actual use of a Clip function in a given program. The Clip function becomes active for both runs when the tenth DT unit is encountered, that is after nine units have elapsed. In the first run no change occurs. In the second example a change is noted because VALU1 no longer is equal to VALU2.

Figure 5.16
Clip Function

```
      THE USE OF THE CLIP FUNCTION                          (C) 1978 UND

*****    S O U R C E    L I S T I N G    *****

0001  TITLE THE USE OF THE CLIP FUNCTION
0002  * EULER
0003  * NARROW
0004  * CHECK
0005  * NOSTATS
0006  L LEVEL.K=INTGRL(RATE.JK)
0007  N LEVEL=100
0008  R RATE.KL=LEVEL.K*FERT.K
0009  A FERT.K=CLIP(VALU1,VALU2,TIME.K,YEAR)
0010  C VALU1=.10
0011  C VALU2=.10
0012  C YEAR=10
0013  PARM DT=1
0014  PARM START=0
0015  PARM STOP=20
0016  PARM PRTPER=1
0017  PARM PLTPER=1
0018  PRINT LEVEL,FERT
0019  PRINT LEVEL.2,FERT.2
0020  PRINT LEVEL.*
0021  PLOT LEVEL/FERT
0022  PLOT LEVEL.2=L/FERT.2=F
0023  PLOT LEVEL.*
0024  RERUN
0025  C VALU1=.01
```

Figure 5.17
Clip Function and RERUN

THE USE OF THE CLIP FUNCTION (C) 1978 UND

(MAIN PROGRAM RUN)

TIME E+00	LEVEL E+00	FERT E-03
0.000	100.00	100.00
1.000	110.00	100.00
2.000	121.00	100.00
3.000	133.10	100.00
4.000	146.41	100.00
5.000	161.05	100.00
6.000	177.16	100.00
7.000	194.87	100.00
8.000	214.36	100.00
9.000	235.79	100.00
10.000	259.37	100.00
11.000	285.31	100.00
12.000	313.84	100.00
13.000	345.23	100.00
14.000	379.75	100.00
15.000	417.72	100.00
16.000	459.50	100.00
17.000	505.45	100.00
18.000	555.99	100.00
19.000	611.59	100.00
20.000	672.75	100.00

PART A
Clip Function
No change in variables
VALU1 = VALU2

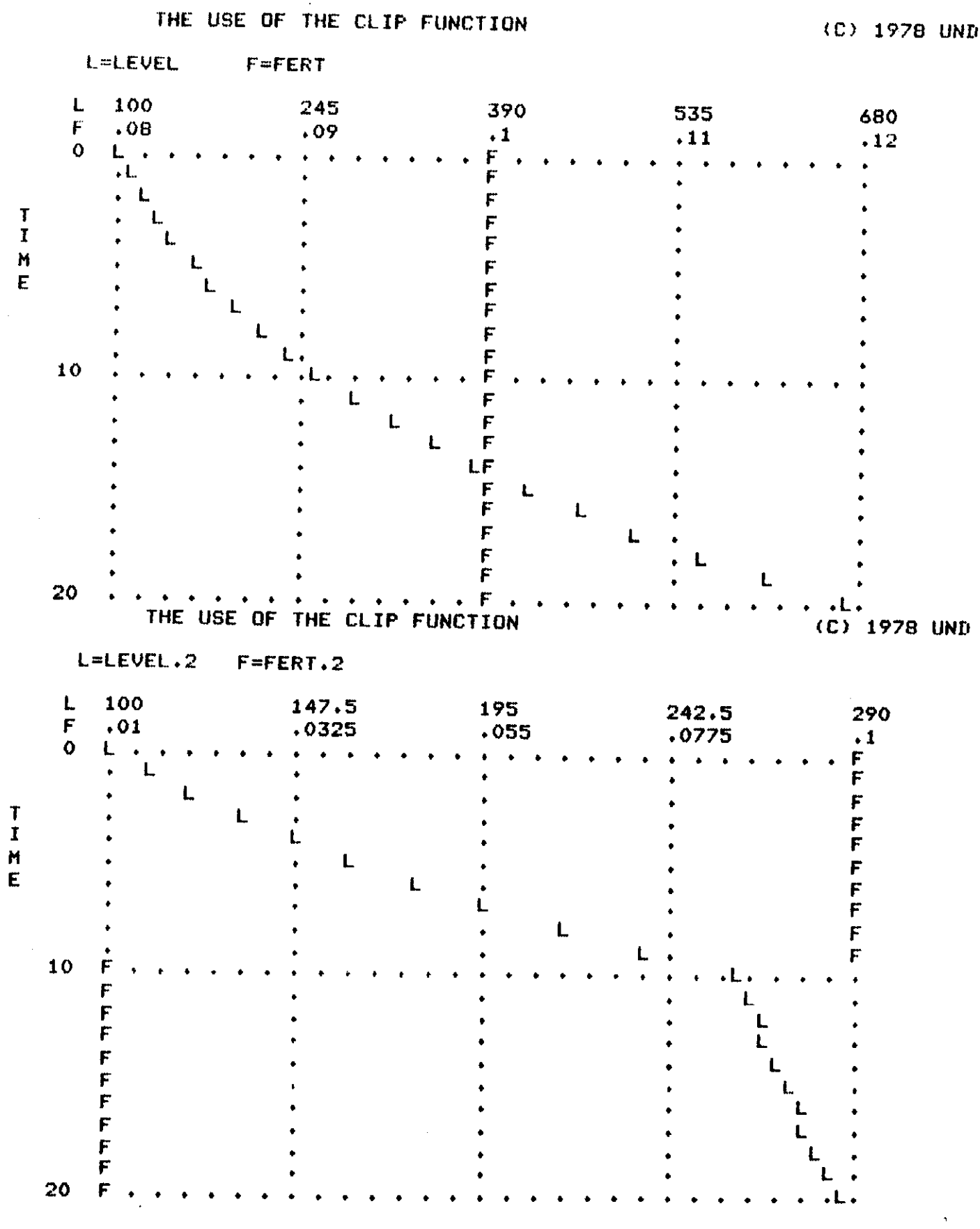
THE USE OF THE CLIP FUNCTION (C) 1978 UND

(RERUN)

TIME E+00	LEVEL.2 E+00	FERT.2 E-03
0.000	100.00	100.00
1.000	110.00	100.00
2.000	121.00	100.00
3.000	133.10	100.00
4.000	146.41	100.00
5.000	161.05	100.00
6.000	177.16	100.00
7.000	194.87	100.00
8.000	214.36	100.00
9.000	235.79	100.00
10.000	259.37	10.00
11.000	261.97	10.00
12.000	264.59	10.00
13.000	267.23	10.00
14.000	269.91	10.00
15.000	272.60	10.00
16.000	275.33	10.00
17.000	278.08	10.00
18.000	280.87	10.00
19.000	283.67	10.00
20.000	286.51	10.00

PART B
Clip Function
Variable Changes
VALU1 ≠ VALU2

Figure 5.18
Clip Function and RERUN



3. STEP Function

The form of the Step function is:

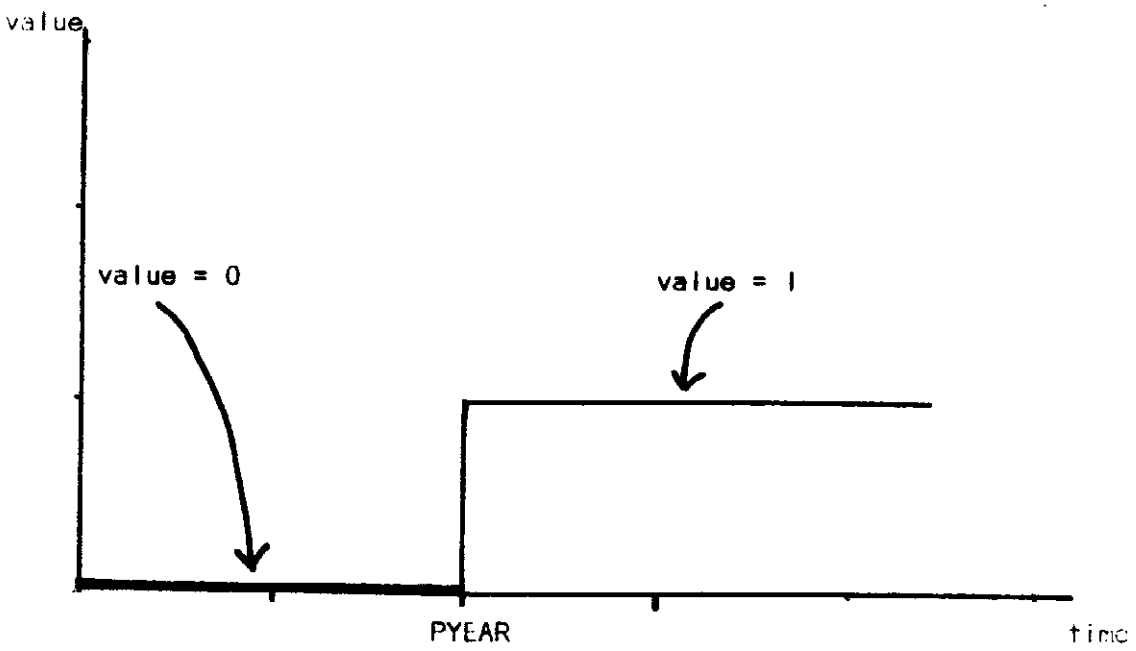
STEP(HT,PYEAR)

HT - the magnitude or height of the step

PYEAR - the value of the TIME variable at which the step becomes activated.

Schematically, a STEP function would look as shown in figure 5.19. If HT were equal to 1, then the value of the Step would be 1. The second argument shows the time at which the Step function would be activated. At all other (previous)times the Step function has a value of zero.

Figure 5.19
Step Function



In NDTRAN the Step function may be used to keep a given level initialized at some pre-determined level until some given time instant at which the level variable is activated and will begin to change. When the Step function is incorporated into a program involving a first order negative feedback loop, the program is shown in Figure 5.20 and the output in Figures 5.21 and 5.22.

Figure 5.20
Step Function Program

```
      USE OF THE STEP FUNCTION                                  (C) 1978 UND
* * * * *    S O U R C E    L I S T I N G    * * * * *

0001  TITLE USE OF THE STEP FUNCTION
0002  * EULER
0003  * CHECK
0004  * NARROW
0005  * NOSTATS
0006  L LEVEL.K=INTGRL(RATE.JK)
0007  N LEVEL=100
0008  R RATE.KL=(CON.K)(DL-LEVEL.K)/ADJTM
0009  A CON.K=STEP(HT,PYEAR)
0010  C PYEAR=10
0011  C HT=1
0012  C ADJTM=5
0013  C DL=500
0014  PARM DT=1
0015  PARM START=0
0016  PARM STOP=20
0017  PARM PRTPER=1
0018  PARM PLTPER=1
0019  NOTE OUTPUT OF STEP FUNCTION
0020  PRINT CON,LEVEL,DL
0021  PLOT CON/LEVEL/DL
```

Figure 5.21
STEP Function Program:
Printed Output

USE OF THE STEP FUNCTION - OUTPUT OF STEP FUNCTI (C) 1978 UND

TIME	CON	LEVEL	DL
0.000	0.0000	100.00	500.00
1.000	0.0000	100.00	500.00
2.000	0.0000	100.00	500.00
3.000	0.0000	100.00	500.00
4.000	0.0000	100.00	500.00
5.000	0.0000	100.00	500.00
6.000	0.0000	100.00	500.00
7.000	0.0000	100.00	500.00
8.000	0.0000	100.00	500.00
9.000	0.0000	100.00	500.00
10.000	1.0000	100.00	500.00
11.000	1.0000	180.00	500.00
12.000	1.0000	244.00	500.00
13.000	1.0000	295.20	500.00
14.000	1.0000	336.16	500.00
15.000	1.0000	368.93	500.00
16.000	1.0000	395.14	500.00
17.000	1.0000	416.11	500.00
18.000	1.0000	432.89	500.00
19.000	1.0000	446.31	500.00
20.000	1.0000	457.05	500.00

Figure 5.22
STEP Function Program
Plotted Output

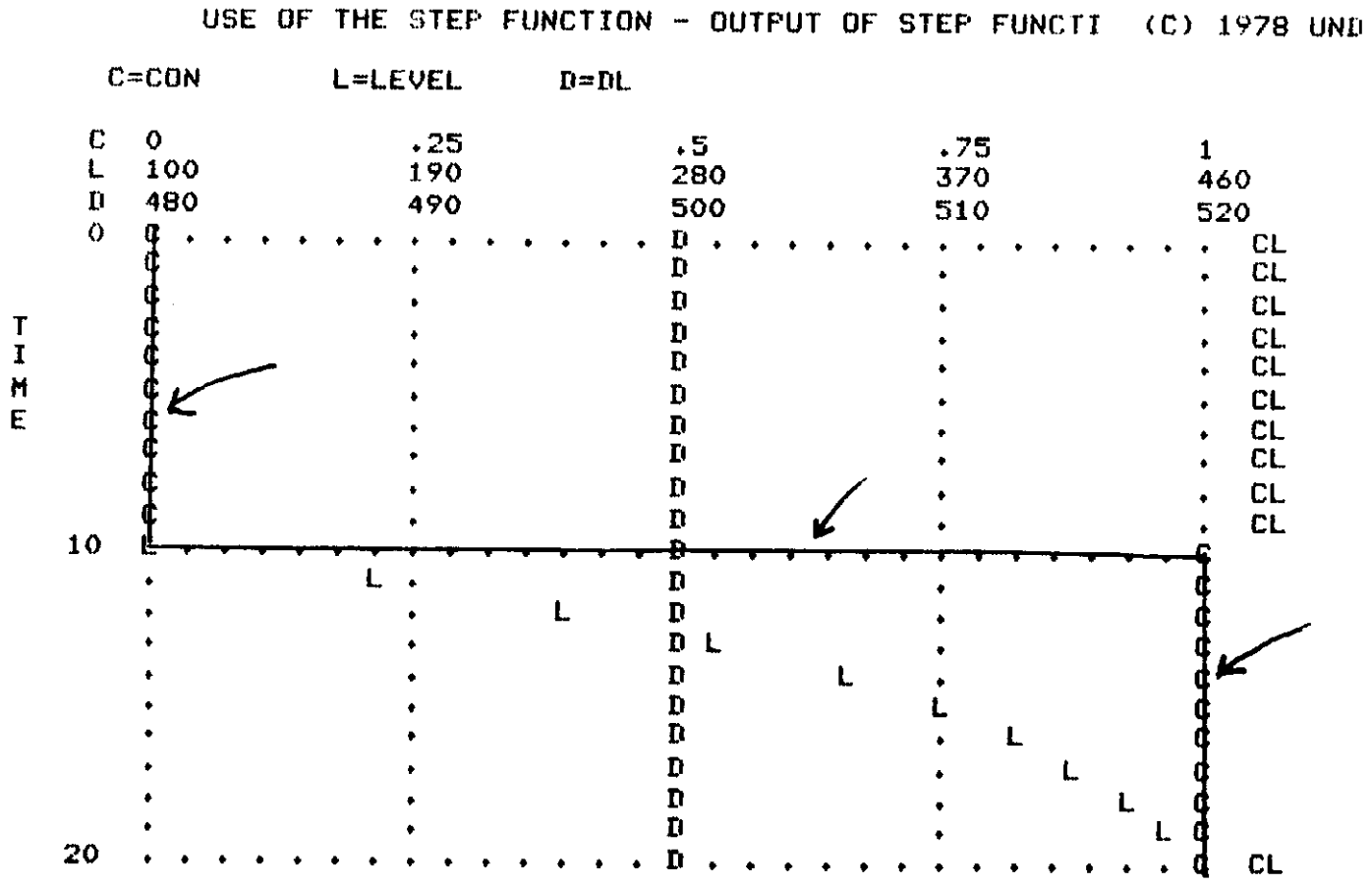


Figure 5.22 shows the plot from the Step function program shown in Figure 5.20. The black line indicated by the arrows shows that the variable CON remains at zero until the 10th year at which point it takes the value of 1. As long as the variable CON is zero, the level variable does not change. In the 10th year when the variable CON has a positive value the RATE equation becomes positive and the level variable begins to increase.

C. TABLE FUNCTIONS

NDTRAN cannot handle graphic input directly. But if the graphic input is converted to a table (table form) where each point is defined by a pair of coordinates, then NDTRAN can handle such input. Figure 5.23 illustrates a table with X and Y coordinates.

Figure 5.23
TABLE Values

X	Y
1	100
2	96
3	90
4	86
5	80
6	75
7	60
8	50

In NDTRAN, the left side of the table values or the X values are the independent or control values. The X values must be of regular interval and with a specific starting and ending point. The right side of Y values , are the dependent values, which may or may not have regular intervals. In fact, they may be increasing, decreasing or irregular. Hence the above table is input to the NDTRAN program as a TABLE ARRAY.

1) TABHL Function

There are several forms of Table functions in NDTRAN. One widely used form is the TABHL function shown in the context of an auxiliary equation as follows:

```
A Y.K=TABHL(NAME,X.K,LOW,HIGH,CHANGE)
T NAME= 100,96,90,86,80,75,60,50
```

The auxiliary function provides the name of the variable that receives a value dependent on the value of the independent variable. The variable in this case being shown as Y. Following the "=" sign, the first variable of the argument list inside the parentheses is the NAME of the table representing the "Y" values shown on Figure 5.23. The second variable or the independent variable, represents the "X" values. The independent variable will have a LOW value, a HIGH value, and an increment, CHANGE. The LOW and the HIGH define a range for the independent variable, associating the first value of the table with LOW, and the last value in the table with HIGH. Looking back to the table shown on Figure 5.23 the following would be seen:

	X
HIGH	8
LOW	1
CHANGE	1

TABHL, an acronym for Table-High-Low has two characteristics. First, it uses a linear interpolation between points within the range of the table. For values of the independent variable less than LOW, TABHL returns the first

value of the table, that is, the value associated with LOW. Similarly, for independent variable values greater than HIGH, TABHL returns the last value of the table, the value associated with HIGH.

This may be explained by using the table in Figure 5.23 as an example. For instance, if the program generated an independent variable value of 3.75, then the Y value that would be used would be 89, 3/4 of the way between the value of 90 and the value of 86 on the table. The basis would be a linear interpolation of the Y values, based on the X value.

Now, what would happen, given the table on Figure 5.23 if the program should generate a control or X value that was either greater than 8 or less than 1 . If the independent value generated were less than 1 , say it were -3, the generated Y value would still be 100. Likewise for X values greater than 8, Y would always be 50.

2) Other TABLE Functions

The general form of the other table functions is slightly different.

```
A ENDPOINT.K = TABND(TAB,INDEP.K,LOW,HIGH)
A FIRSTLST.K = TABFL(TAB,INDEP.K,LOW,HIGH)
A REGULAR.K  = TABLE(TAB,INDEP.K,LOW,HIGH)
```

TAB is the name of the table array to be used in the function; INDEP is the name of the independent variable; LOW and HIGH define the range of the independent variable and fit it to the table. These table functions

all are characterized by fourth-order curve fitting techniques for establishing the points within the existing range of the table. TABHL function is the only function using linear interpolation. Each of the table functions differs somewhat in the manner in which end-points beyond the range of the table are handled.

3) Examples

The graph in Figure 5.26 , derived from the program in Figure 5.25 illustrates the difference between linear interpolation and a fourth-order curve fit for establishing points within the existing range of the table. The function TABHL uses the linear interpolation and all of the other table functions use a fourth-order curve fit technique. In Figure 5.26 the + symbol is used to show the fourth-order curve fit function and the * symbol is used to show the linear interpolation. The table functions used here are:

```
R TEST5.KL=TABLE(TAB2,TIME.K,0,6)
R TEST6.kl=TABHL(TAB2,TIME.K,0,6,2)
T TAB2=1,2,1,2
```

Schematically the table would look as shown in Figure 5.24.

Figure 5.24
Table Function (array)

TIME.k	DEPENDENT VARIABLE
0	1
2	2
4	1
6	2

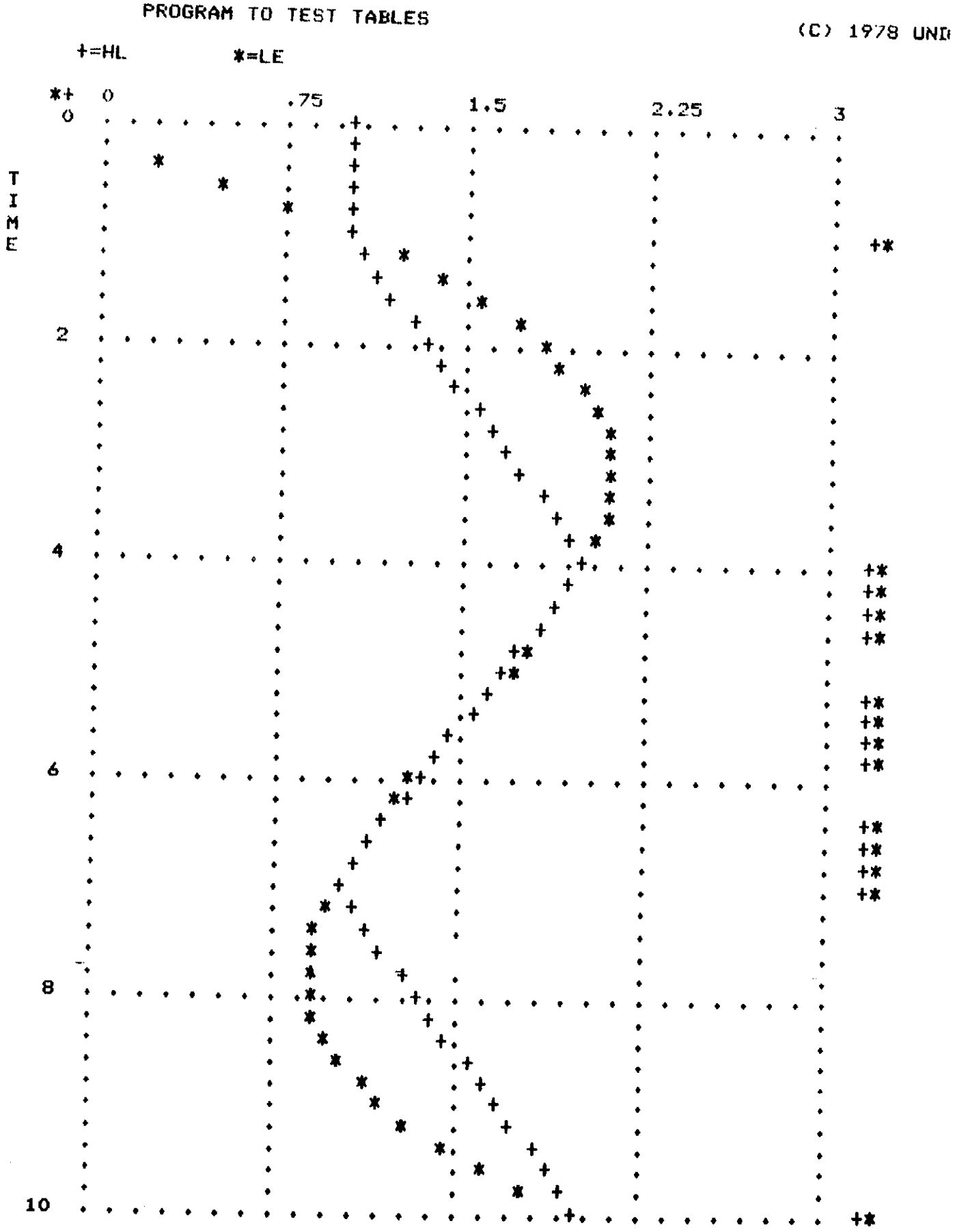
Now, the relevant question is, if TIME is 1.1 instead of 1 or 2; what is the value produced for TIME=1.1 using the two different table functions. Figure 5.25 shows the program and Figure 5.26, the output.

Figure 5.25
Table Functions:
Linear Interpolation
and
Fourth-Order Curve Fit

```
      PROGRAM TO TEST TABLES                                    (C) 1978 UND
* * * * *    S O U R C E    L I S T I N G    * * * * *
0001  TITLE PROGRAM TO TEST TABLES
0002  * EULER
0003  * CHECK
0004  * NARROW
0005  * NOSTATS
0006  L LEVEL.K=INTGRL(RATE.JK)
0007  R RATE.KL=LEVEL.K
0008  N LEVEL=0
0009  S HL.K=TABHL(TAB1,TIME.K,1,10,1)
0010  S LE.K=TABLE(TAB1,TIME.K,1,10)
0011  T TAB1=1,2,1,2
0012  PARM DT=.2
0013  PARM STOP=10
0014  PARM PLTPER=.2
0015  PLOT HL=+(0,3),LE=*
```

It is the relationships of the points within the table range that are of importance. Notice the TABHL values are calculated on the basis of a straight line or linear interpolation. The TABLE values are calculated on the basis of the fourth-order curve fit. The user may decide which method is best suited for his or her application.

Figure 5.26
Table Function Output
(Program:Figure 5.25)



The next question concerns the action of the various table functions outside of the range of the table itself. Figure 5.27 shows the program used to demonstrate this. The tabled data used in this demonstration is shown below:

TIME.K	TAB1
3	-5
4	0
5	1
6	8
7	27

Remember, in this test we wish to indicate the performance of the table functions outside of the limits of the table or form values of TIME less than 3 and greater than 7. Four table functions are used and the plot symbol for each is shown below:

Table Function	Plot Symbol
TABHL	H
TABLE	L
TABFL	F
TABND	N

The extrapolation beyond the limits or range of the table is as follows:

TABHL - a horizontal line based on the first and last value of the table;
TABLE - an extrapolation from the four points closest to the respective end points of the table;
TABFL - a straight line based on the first point of the table and the last point;
TABND - a straight line (linear interpolation) based on the first two points of the table and the last two.

The extrapolation beyond the table range is for time periods 0 - 3 and from 7 - 10 indicated by the heavy dashed line on Figure 5.28.

Figure 5.27
Table Function Behavior
Extrapolation Beyond
Table Limits

```
PAGE 1     PROGRAM TO TEST TABLES                                    (C) 1978 UNI

     * * * * *    S O U R C E    L I S T I N G    * * * * *

0001   TITLE PROGRAM TO TEST TABLES
0002   * EULER
0003   * CHECK
0004   * NARROW
0005   * NOSTATS
0006   L LEVEL.K=INTGRL(RATE.JK)
0007   R RATE.KL=LEVEL.K
0008   N LEVEL=0
0009   S HL.K=TABHL(TAB1,TIME.K,2,8,2)
0010   S LE.K=TABLE(TAB1,TIME.K,2,8)
0011   S FL.K=TABFL(TAB1,TIME.K,2,8)
0012   S ND.K=TABND(TAB1,TIME.K,2,8)
0013   T TAB1=1,2,1,2
0014   PARM DT=.2
0015   PARM STOP=10
0016   PARM PLTPER=.2
0017   PLOT HL(.5,2.5),LE,FL,ND
```

Figure 5.28
Table Function Behavior
Extrapolation Beyond
Table Limits

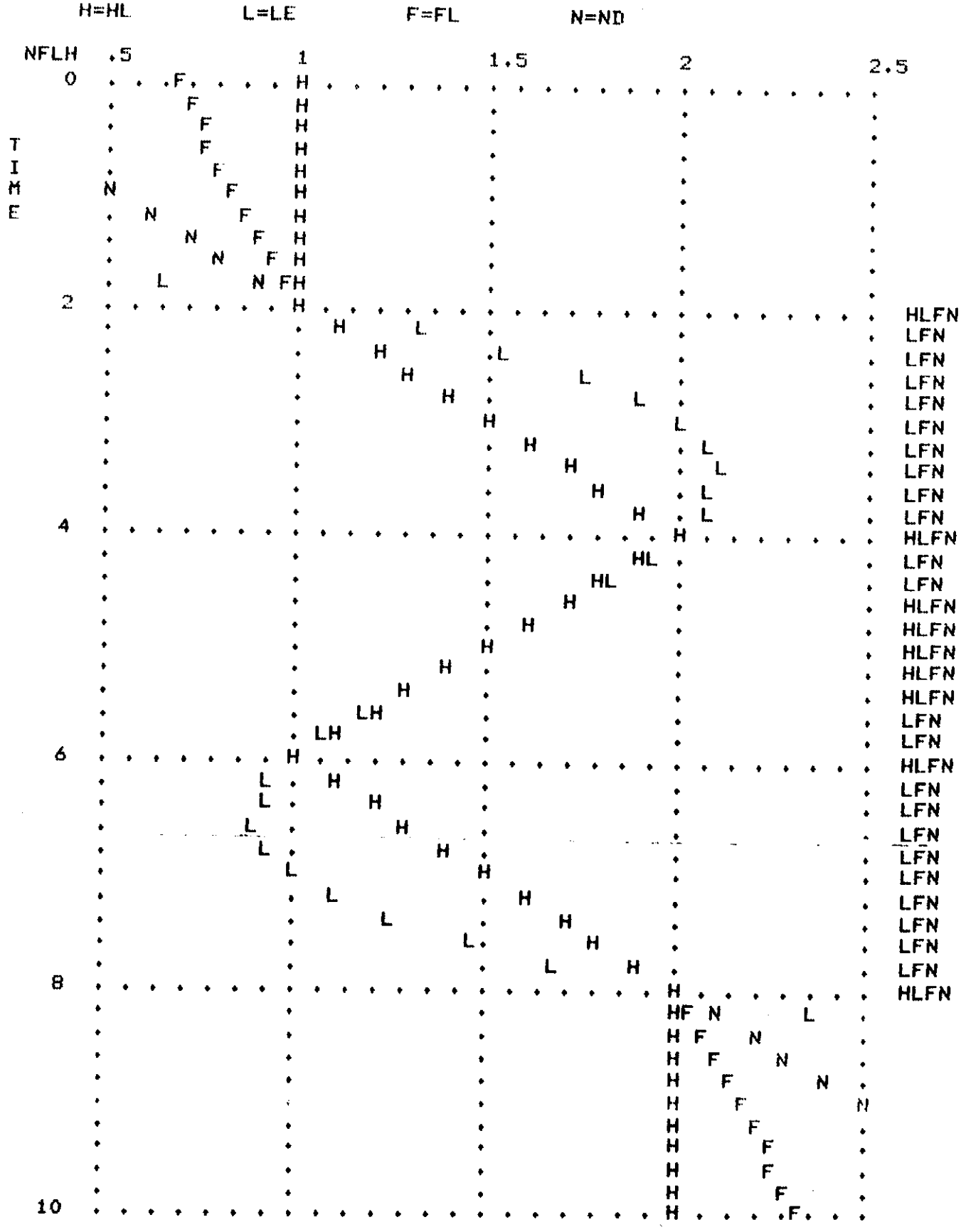


Figure 5.27.1
Table Function Behavior
Extrapolation Beyond
Table Limits

```
**********    S O U R C E    L I S T I N G    **********

10001   * THIS PROGRAM SHOWS THE USE OF THE TABLE FUNCTION
10002   *NOSTATS
10003   *EULER
10004   *NARROW
10005   *NOWARN
10006   L LEV.K=INTEGRAL(TEST1.JK)
10007   N LEV=1
10008   R TEST1.KL=LEV.K*AUX.K
10009   A AUX.K=TABND(TAB1,TIME.K,3,7)
10010   A AU1.K=TABHL(TAB1,TIME.K,3,7,1)
10011   A AU2.K=TABFL(TAB1,TIME.K,3,7)
10012   T TAB1=-5/0/1/8/27
10013   SPEC DT=.1,START=0,STOP=10,PLTPER=.25
10014   PLOT AUX=#,AU1=*,AU2=+
```

Note that the TABLE function cannot be used in this kind of test. In NDTRAN Version 1, the TABLE Function inhibits execution outside of the table range. Otherwise the behavior characteristics are the same as NDTRAN Version 2.

D) Summary

A number of commonly used functions are available on NDTRAN. Their use and definitions are similar to those previously explained. All presently available functions for NDTRAN versions 1 and 2 are shown in the table below.

Table 5.1
NDTRAN FUNCTIONS

	NAME	FORM	PROCEDURE
1.	ABS	...ABS(VAR1)	Takes the absolute value of the argument.
2.	CLIP	...CLIP(VAR1,VAR2,VAR3,VAR4)	Uses the value of VAR1 if VAR3 is greater than or equal to VAR4, otherwise uses VAR2.
3.	COS	...COS(VAR)	Generates a cosine transformation of VAR
4.	DELAY	available as procedures	See Chapter 4, pages 43 ff.
5.	EXP	...EXP(VAR1)	EXP(VAR1) means e^{VAR1} where e is the base of the natural logarithm.
6.	LOG	...LOG(VAR1)	Means, take the natural log of the value of VAR1
7.	MAX	...MAX(VAR1,VAR2)	Selects the greater of the values of VAR1 and VAR2.
8.	MIN	...MIN(VAR1,VAR2)	Selects the lesser of the values of VAR1 and VAR2.
9.	NOISE	...NOISE(SEED)	Generates a (pseudo) random number set uniformly distributed between 0 and 1. 'Seed' here and in NORMRN is any 5 digit positive number.
10.	NORMRN	...NORMRN(SEED,VAR1,VAR2)	Generates random numbers normally distributed with a specified mean and Standard Deviation, where VAR1 is the Mean and VAR2 is the Standard Deviation.

11.	PULSE	PULSE(VAR1,VAR2,VAR3,VAR4)	Output a pulse of height VAR1, width VAR2, starting at time=VAR3 with an interval of time between pulses of VAR4.
12.	RAMP	RAMP(VAR1,VAR2)	Produces a variable of slope=VAR7 beginning with value zero at time=VAR2.
13.	SAMPLE		Selects a sample at uniformly spaced sample intervals.
14.	SIN	SIN(VAR1)	Generates a sinusoid transformation .
15.	SORT	SQRT(VAR1)	Generates the square root of the value of variable shown as the argument.
16.	STEP	STEP(VAR1,VAR2)	Produces a STEP disturbance of height VAR1 at time VAR2.
17.	SWITCH	SWITCH(VAR1,VAR2,VAR3)	Selects VAR1 if VAR3 is positive. Otherwise selects VAR2.
18.	TABHL	TABHL(NAME,INDEP,LOW,HIGH,CHANGE)	Performs a linear interpolation between points in the table, LOW and HIGH used for values outside the range.
19.	TABLE	TABLE(NAME,INDEP,LOW,HIGH)	Performs a fourth order curve fit for points within the table; execution terminates outside the range.
20.	TABFL	TABFL(NAME,INDEP,LOW,HIGH)	Performs a fourth order curve fit for points within the table; linear extrapolation based on first and last values of table for values outside the range.
21.	TABND	TABND(NAME,INDEP,LOW,HIGH)	Performs a fourth order curve fit for points within the table; linear extrapolation based on two points at either end of the table for values outside the range.

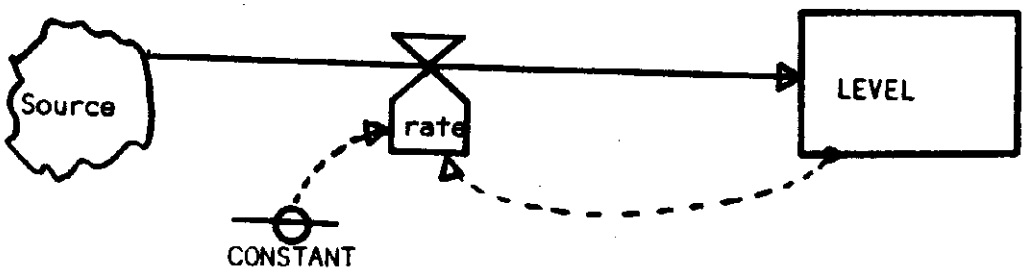
Chapter 6
Integration Methods in NDTRAN

A) Introduction

Most problems in continuous simulation techniques involve feedback, i.e. the state variables themselves directly effect the rates at which the state variables change. Such systems imply exponential change. To state it simply, the state variables (levels) themselves at a given time are an important element in the determination of the rate of change, of the levels over the coming time period (DT).

The most fundamental feedback structure in dynamic simulation can be shown in the flow diagram of Figure 6.1

Figure 6.1
Feedback Structure



The level itself is an important determinant of the rate of change of the level for the coming period. The constant determines the slope of the curve at any time instant. Theoretically, the values for the state variable that we are interested in would be the area under the curve, as shown in Figure 6.2.*

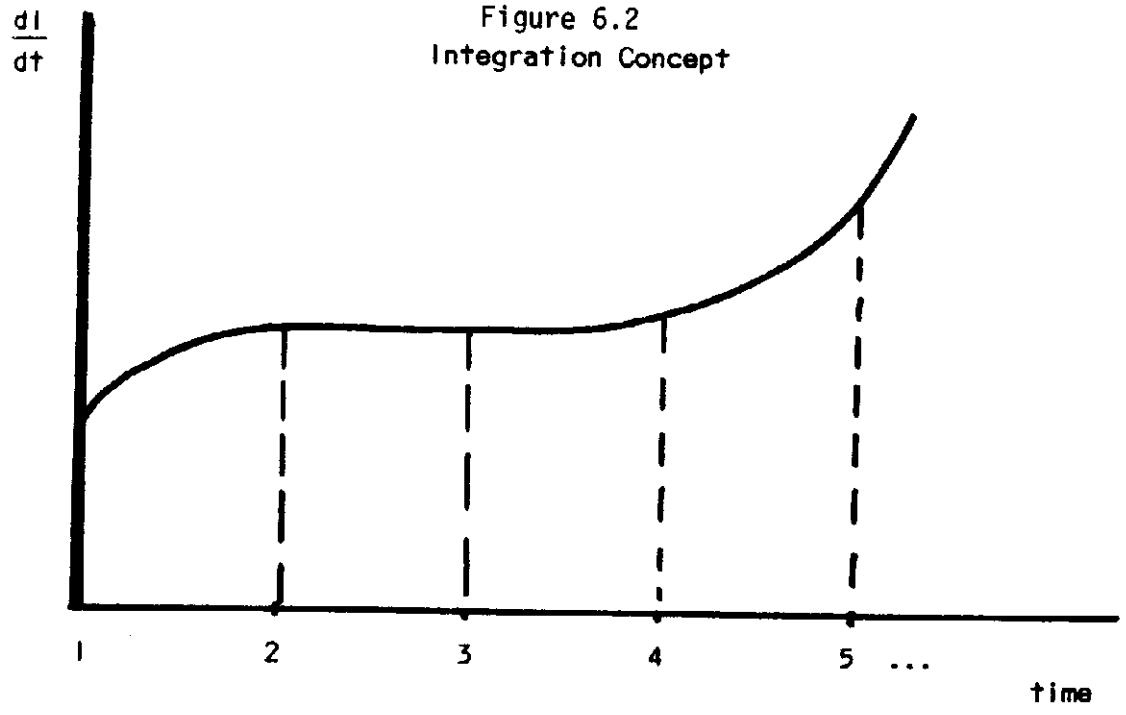


Figure 6.2
Integration Concept

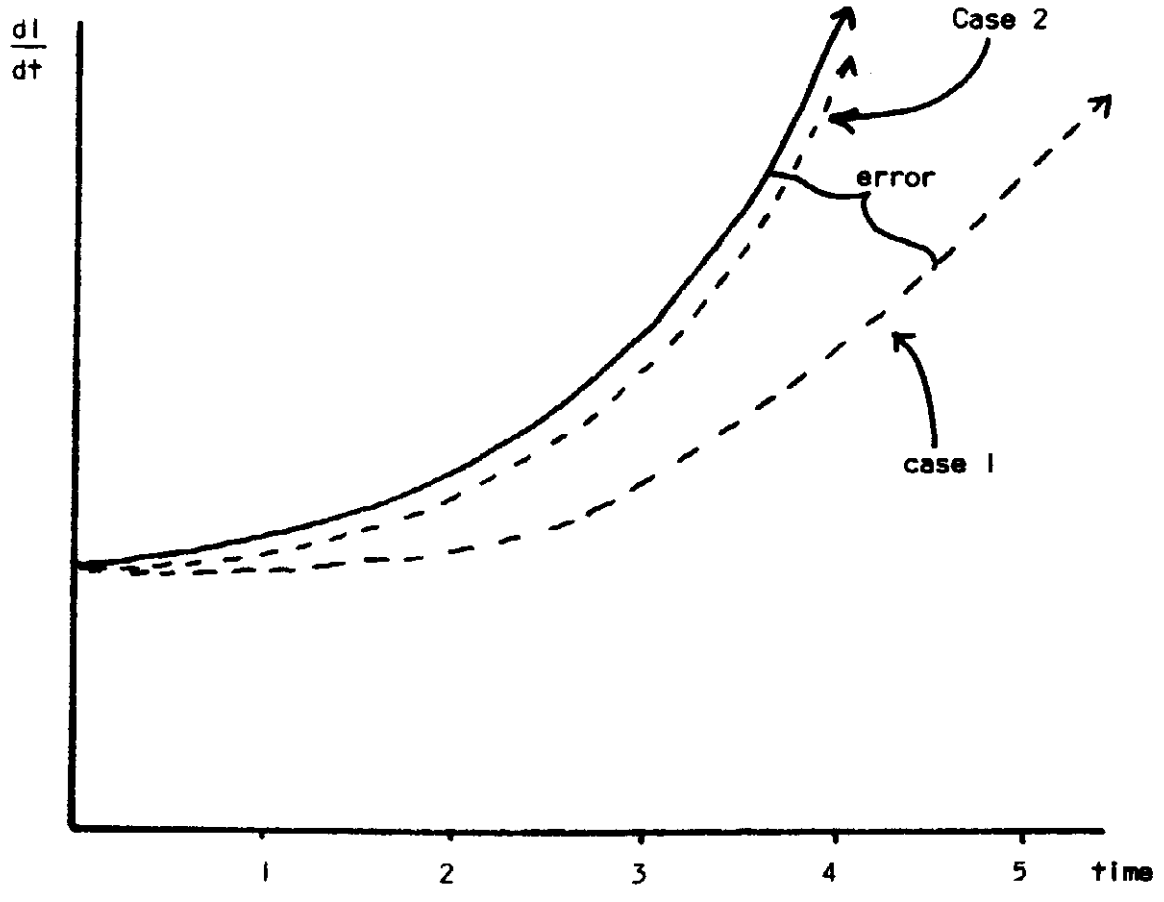
For complete accuracy we should add to the measurement of the level variable at some starting time T_1 the total area under the curve for as many time periods (DT's) as we wish to run the simulation following T_1. However, once we come to a computer numerical technique for computing or calculating the area under the curve, we find that the area can only be approximated. The field of numerical analysis has provided a variety of methods for doing this. Each of the methods has a set of properties which make it desirable

* In other words, the concept of feedback and the basic structure of our model is capable of being solved using integration.

or undesirable for the solution to a particular integration problem.

Intuitively, the object of any integration method is to approximate as closely as possible the actual curve that represents the increase or decrease in the value of a level variable. Each of the possible methods that have been devised have some "error" or difference between the actual value of the function and the calculated value. In Figure 6.3, the actual value of the level variable is shown as the solid line, with two possible variations (from the actual) of the calculated value of the level variable.

Figure 6.3
Error Concept in Integration



As shown in Figure 6.3 any methods (numerical) that we choose will vary from the actual. In case 1, the calculated value differs from the actual value by being understated a considerable amount. In case 2 the calcualted value closely approximates the actual value. Each of these cases have requirements as to necessary information and the computer time necessary to calculate the values of the level variables. Further, depending on the integration method chosen, the step size must be varied for accuracy, which will have an affect on the "roundoff" error.

B) Methods of Integration Used

1) Euler Integration

Euler's method for integration is one of the oldest, simplest, and best known methods for integrating. The method has a relatively large formula error and is often unstable. For this reason, it is usually undesirable except in instances where summation is actually preferred over integration as in simulations involving bank interest problems.

To compute the value of the next level using the Euler Method requires the past value of the level and the time rate of change of the level based on that past value. There is relatively small computer time cost since the time rate of change of the level from the previous time need be computed only one time. The error is proportional to DT^2.

2) Runge-Kutta Integration

The Runge-Kutta method is usually applied to engineering and/or physics problems where a high degree of accuracy is required. Perhaps as important, it is desirable where higher order derivates are non-zero or non-continuous.

This method requires the past level value and the value of the time rate of change for four different values of the level itself and time. The method (fourth order) essentially divides each step size into four parts and calculates the values for four different values of the level itself. The cost is relatively high, usually being four times that of the Euler method for a given step size, since the rates must be computed four times for each integration step. The method is highly accurate and stable with error terms proportional to DT^5.

3) Adams Bashforth Predictor Integration

This method is very desirable and very efficient, particularly where continous functions are concerned. Since the rates are assumed to be continuous, past rate values are used to compute present level values. This method takes three time periods to dissipate resulting transients for non-continuous rates. It produces accurate results and is used in engineering and physics problems as well as for many continuous function problems.

This method requires the past level value and the time rate of change of the level at the past four time periods. The computer time cost is relatively small as the past rate values are saved from the previous steps, and for each given step the rate is calculated only one time. The method does require a Runge-Kutta start since initially the required past four rate values are not available. For all continous equations the cost is about equal to the Euler Method and the accuracy is about equal to the Runge-Kutta method.

C) Tests of Integration Methods

While there are many tests that can be devised, we have chosen to subject the integration methods in NDTRAN (versions 1 and 2) to three:

1) Accuracy - having the interpreter integrate polynomial curves with known solutions including:

RATE = dl/dt

R = 0
R = 1
R = 2 * time
R = 3 * $time^2$
R = 4 * $time^3$

2) tracking - the property of the integration routine to remain close (over time) to the actual solution; in this case, first order growth and decline curves.

3) accuracy and stability of cyclic phenomena - the integration methods of the interpreter were used to integrate RLC circuits with known solutions.

There is a difference between NDTRAN version 1 and NDTRAN2 that ought to be noted. In NDTRAN version 1 the TIME variable was calculated (rather than integrated). In developing NDTRAN2 a number of users

contacted us about this point. In discussing this point with mathematicians familiar with this type of problem we came to the conclusion that if all of the variables in a model were integrated, but the TIME variable were calculated a transient error could develop. This problem could be corrected if in NDTRAN2 we converted the TIME variable from a calculation to an integration. We have done this. Since NDTRAN and NDTRAN2 are both double precision interpreters (on all systems with sufficient core to handle double-precision and remain within the available FORTRAN batch partition), there appears to be a high degree of accuracy in all calculations in NDTRAN and NDTRAN2. Comparisons between the output of NDTRAN and NDTRAN2 on these tests do indicate some differences. As of this point we believe the differences are because of the changes indicated in NDTRAN2. We are still considering these questions of accuracy.

1) Accuracy

The first test of accuracy was carried out using known polynomial curves. In this test the important point that we considered was the magnitude of the relative error. We would expect that Euler method would accurately integrate zero and constant curves, and that Runge-Kutta and Adams-bashforth would integrate through third order curves. Figure 6.4 shows the approach used for the integration of polynomial curves through order 5. It should be noted that the relative error for Euler was significant after the constant curve and for Runge-Kutta and Adams-Bashforth after the fifth order curves.

Figure 6.4
Accuracy Integration Tests

```
Integrate: Y' = 0, Y(0)=0,  Solution: Y=0;      Y1
Integrate: Y'=1, Y(0)=0  ,  Solution: Y=T;      Y2
Integrate: Y'=2T,  Y(0)=0,  Solution: Y=T**2;   Y3
Integrate: Y'=3T**2,Y(0)=0, Solution: Y=T**3;   Y4
Integrate: Y'=4T**3,y(0)=0, Solution: Y=T**4;   Y5
```

```
NOTE
NOTE   INTEGRATE: Y'=4T**3, Y(0)=0,  SOLUTION: Y=T**4
NOTE
L Y5=INTGRL(DY5)
R DY5=4*TIME**3
N Y5=0
S ACTLY5=TIME**4
S ABSR5=ABS(ACTLY5-Y5)
S RELR5=ABSR5/ACTLY5
```

As noted, the relative errors for the results of the integrations using the three integration methods became significant for Euler after the constant curve, and became significant for the other methods after the fifth order curve.

2) Tracking

The second test is the tracking test. Figure 6.5 shows the program segment indicating the test. The program is appropriately modified for the integration methods by changing the control option * EULER, * RKINT, * ABINT . In this test the absolute error is important as the relative error becomes meaningless as the solution approaches zero.

Figure 6.5
Integration: First Order Growth and
Decline Curves: DT=.1

```
*****    S O U R C E    L I S T I N G    *****

0001   TITLE NDTRAN TRACKING TEST
0002   * NOWARN
0003   * NOCHECK
0004   * NOSTATS
0005   * EULER
0006   C A=-1
0007   N Y=1
0008   R DY.KL=A*Y.K
0009   L Y.K=INTGRL(DY.JK)
0010   PARM DT=.1
0011   PARM START=0
0012   PARM STOP=10
0013   PARM PRTPER=1
0014   S ACTUAL.K=EXP(A*TIME.K)
0015   S ABSERR=ABS(ACTUAL.K-Y.K)
0016   S RELERR.K=ABSERR.K/ACTUAL.K
0017   PRINT Y,ABSERR,RELERR,ACTUAL
0018   PRINT Y.2,ABSERR.2,RELERR.2,ACTUAL.2
0019   RERUN
0020   C A=1
```

Figures 6.6, 6.7 and 6.8 show the results of the execution testing using the three integration methods. In these figures, the variable Y is the integrated or calculated variable as shown in line 9 of the program in Figure 6.5. Variables ABSERR, RELERR AND ACTUAL are, respectively, the absolute error, the relative error and the actual or calculated value as shown in lines 14 - 16 of the program.

Figure 6.6
EULER Integration: First Order Curves

Part A: Growth Curve

TIME E+00	Y.2 E+03	ABSERR.2 E+03	RELERR.2 E-03	ACTUAL.2 E+03
0.000	0.001	0.0000	0.00	0.001
1.000	0.003	0.0001	45.82	0.003
2.000	0.007	0.0007	89.53	0.007
3.000	0.017	0.0026	131.25	0.020
4.000	0.045	0.0093	171.05	0.055
5.000	0.117	0.0310	209.03	0.148
6.000	0.304	0.0989	245.27	0.403
7.000	0.790	0.3069	279.84	1.097
8.000	2.048	0.9326	312.84	2.981
9.000	5.313	2.7901	344.32	8.103
10.000	13.781	8.2459	374.36	22.026

Part B: Decline Curve

TIME E+00	Y E+00	ABSERR E-03	RELERR E-03	ACTUAL E+00
0.000	1.0000	0.000	0.00	1.0000
1.000	0.3487	19.201	52.19	0.3679
2.000	0.1216	13.759	101.66	0.1353
3.000	0.0424	7.396	148.55	0.0498
4.000	0.0148	3.535	192.99	0.0183
5.000	0.0052	1.584	235.11	0.0067
6.000	0.0018	0.682	275.03	0.0025
7.000	0.0006	0.285	312.87	0.0009
8.000	0.0002	0.117	348.74	0.0003
9.000	0.0001	0.047	382.73	0.0001
10.000	0.0000	0.019	414.95	0.0000

Figure 6.7
Runge-Kutta Integration: First Order Curves Tracking

Part A: Growth Curve

TIME E+00	Y.2 E+00	ABSERR.2 E-03	RELERR.2 E-06	ACTUAL.2 E+00
0.000	1	0.00	0.0000	1
1.000	3	0.00	0.7668	3
2.000	7	0.01	1.5336	7
3.000	20	0.05	2.3003	20
4.000	55	0.17	3.0671	55
5.000	148	0.57	3.8339	148
6.000	403	1.86	4.6007	403
7.000	1097	5.89	5.3675	1097
8.000	2981	18.29	6.1342	2981
9.000	8103	55.92	6.9010	8103
10.000	22026	168.89	7.6678	22026

Part B: Decline Curve

TIME E+00	Y E+00	ABSERR E-09	RELERR E-06	ACTUAL E+00
0.000	1.0000	0.00	0.0000	1.0000
1.000	0.3679	333.24	0.9058	0.3679
2.000	0.1353	245.19	1.8117	0.1353
3.000	0.0498	135.30	2.7175	0.0498
4.000	0.0183	66.36	3.6234	0.0183
5.000	0.0067	30.52	4.5292	0.0067
6.000	0.0025	13.47	5.4351	0.0025
7.000	0.0009	5.78	6.3409	0.0009
8.000	0.0003	2.43	7.2468	0.0003
9.000	0.0001	1.01	8.1526	0.0001
10.000	0.0000	0.41	9.0585	0.0000

Figure 6.8
Adams-Bashforth Integration:
First Order Curves Tracking

Part A: Growth Curve

TIME E+00	Y.2 E+00	ABSERR.2 E+00	RELERR.2 E-06	ACTUAL.2 E+00
0.000	1	0.0000	0.00	1
1.000	3	0.0001	21.11	3
2.000	7	0.0004	50.98	7
3.000	20	0.0016	80.86	20
4.000	55	0.0060	110.73	55
5.000	148	0.0209	140.60	148
6.000	403	0.0688	170.47	403
7.000	1096	0.2197	200.33	1097
8.000	2980	0.6862	230.20	2981
9.000	8101	2.1073	260.07	8103
10.000	22020	6.3862	289.93	22026

Part B: Decline Curve

TIME E+00	Y E+00	ABSERR E-06	RELERR E-06	ACTUAL E+00
0.000	1.0000	0.000	0.00	1.0000
1.000	0.3679	10.616	28.86	0.3679
2.000	0.1353	9.421	69.61	0.1353
3.000	0.0498	5.495	110.37	0.0498
4.000	0.0183	2.768	151.14	0.0183
5.000	0.0067	1.293	191.90	0.0067
6.000	0.0025	0.577	232.67	0.0025
7.000	0.0009	0.249	273.44	0.0009
8.000	0.0003	0.105	314.21	0.0003
9.000	0.0001	0.044	354.98	0.0001
10.000	0.0000	0.018	395.76	0.0000

3) Cyclic Phenomena

The final step of the integration methods is the integration of cyclic phenomena to test the accuracy and stability of the methods. The program is shown in Figure 6.9, representing a series RLC circuit shown in Figure 6.10.in NDTRAN. Figures 6.11, 6.12 and 6.13 show the results of the program execution using the three integration methods.

Figure 6.9
Series RLC Circuit
Program DT=.02

```
*****   S O U R C E   L I S T I N G   *****

0001   TITLE STEP RESPONSE OF RLC CIRCUIT
0002   * NOSTATS
0003   * CHECK
0004   * EULER
0005   N VC=0
0006   N IL=0
0007   L VC.K=INTGRL(ILR.JK)
0008   L IL.K=INTGRL(VIR.JK)
0009   R ILR.KL=IL.K/C
0010   R VIR.KL=10/L-R*IL.K/L-VC.K/L
0011   C C=4.94E-2
0012   C R=1
0013   C L=.5
0014   C A=1
0015   C B=6.2832853
0016   S VCS.K=10*(1+EXP(-A*TIME.K)*(-A/B*SIN(B*TIME.K)
       X -COS(B*TIME.K)))
0017   PARM DT=.02
0018   PARM START=1
0019   PARM STOP=3
0020   PARM PRTPER=.1
0021   S ABSERR.K=VCS.K-VC.K
0022   S RELERR.K=ABS(ABSERR.K)/VCS.K
0023   PRINT VC,VCS,ABSERR,RELERR
```

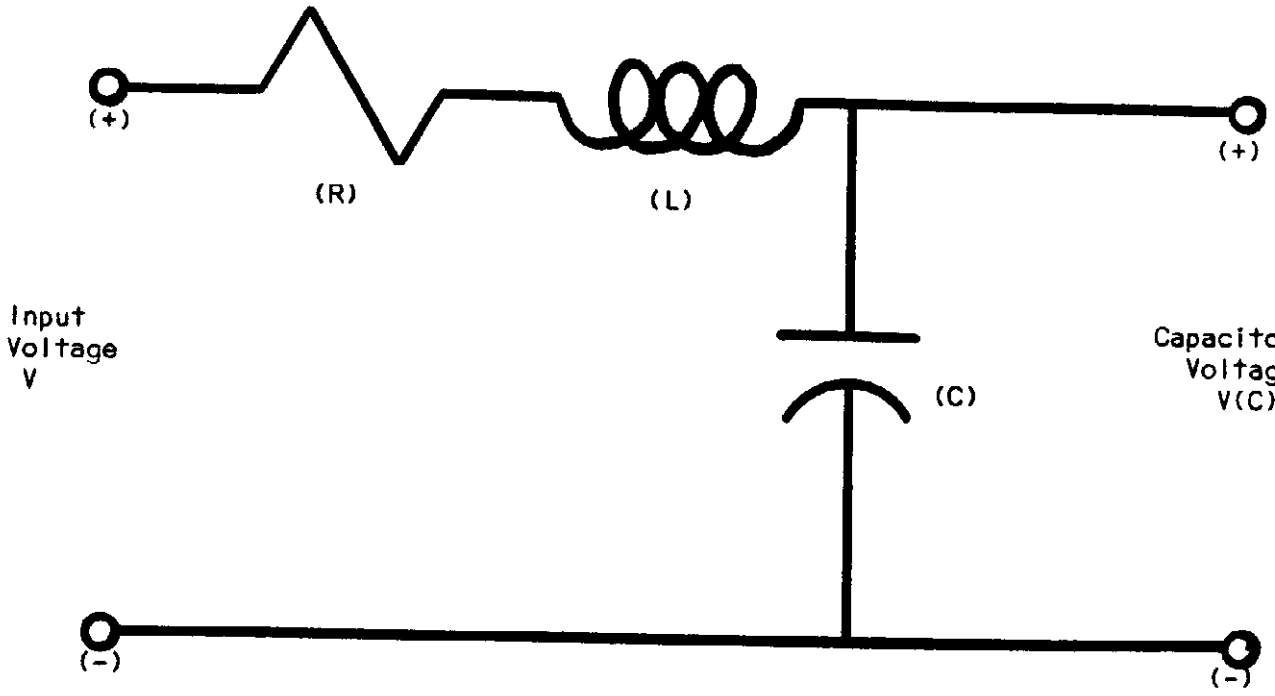


Figure 6.10
Step Response of Series RLC Circuit

R = 1 ohm
L = .5 henries
C = .0494 farads
V = 10 volt step

Figure 6.11
Series RLC Circuit
Euler Integration
DT=.02

STEP RESPONSE OF SERIES RLC CIRCUIT

TIME E+00	VC E+00	VCS E+00	ABSERR E+00	RELERR E-03
0.0000	0.000	0.000	0.0000	0.00
0.1000	1.543	1.833	0.2902	158.30
0.2000	6.070	6.231	0.1608	25.81
0.3000	11.551	11.168	-0.3830	34.30
0.4000	15.831	14.796	-1.0347	69.93
0.5000	17.447	16.065	-1.3814	85.99
0.6000	16.098	14.953	-1.1444	76.53
0.7000	12.624	12.286	-0.3381	27.52
0.8000	8.565	9.292	0.7269	78.23
0.9000	5.502	7.091	1.5891	224.09
1.0000	4.467	6.321	1.8541	293.32
1.1000	5.616	6.996	1.3793	197.17
1.2000	8.273	8.613	0.3408	39.57
1.3000	11.273	10.430	-0.8429	80.82
1.4000	13.457	11.764	-1.6924	143.86
1.5000	14.102	12.231	-1.8706	152.93
1.6000	13.142	11.822	-1.3193	111.60
1.7000	11.117	10.841	-0.2758	25.44
1.8000	8.905	9.739	0.8347	85.70
1.9000	7.353	8.930	1.5771	176.61
2.0000	6.966	8.647	1.6809	194.39

Figure 6.12
Series RLC Circuit
Runge-Kutta Integration
DT=.02

STEP RESPONSE OF SERIES RLC CIRCUIT (C) 1978 UND

TIME E+00	VC E+00	VCS E+00	ABSERR E-03	RELERR E-06
0.0000	0.000	0.000	0.0000	0.00
0.1000	1.834	1.833	-0.3126	170.52
0.2000	6.232	6.231	-0.9645	154.80
0.3000	11.169	11.168	-1.3952	124.93
0.4000	14.797	14.796	-1.1880	80.30
0.5000	16.066	16.065	-0.3062	19.06
0.6000	14.952	14.953	0.8969	59.98
0.7000	12.284	12.286	1.8713	152.31
0.8000	9.289	9.292	2.1502	231.41
0.9000	7.090	7.091	1.5765	222.32
1.0000	6.321	6.321	0.3712	58.72
1.1000	6.997	6.996	-0.9733	139.13
1.2000	8.615	8.613	-1.9153	222.36
1.3000	10.432	10.430	-2.0950	200.87
1.4000	11.766	11.764	-1.4752	125.39
1.5000	12.232	12.231	-0.3374	27.59
1.6000	11.821	11.822	0.8509	71.97
1.7000	10.839	10.841	1.6350	150.82
1.8000	9.738	9.739	1.7504	179.72
1.9000	8.929	8.930	1.2094	135.44
2.0000	8.646	8.647	0.2726	31.53

Figure 6.13
Series RLC Circuit
Adams-Bashforth Integration
DT=.02

STEP RESPONSE OF SERIES RLC CIRCUIT

TIME E+00	VC E+00	VCS E+00	ABSERR E-03	RELERR E-06
0.0000	0.000	0.000	0.0000	0.00
0.1000	1.833	1.833	-0.1205	65.74
0.2000	6.231	6.231	-0.2902	46.58
0.3000	11.169	11.168	-0.6225	55.74
0.4000	14.797	14.796	-0.9260	62.58
0.5000	16.066	16.065	-0.9694	60.34
0.6000	14.954	14.953	-0.6353	42.49
0.7000	12.286	12.286	0.0024	0.19
0.8000	9.291	9.292	0.7058	75.96
0.9000	7.090	7.091	1.1845	167.04
1.0000	6.320	6.321	1.2311	194.76
1.1000	6.995	6.996	0.8170	116.79
1.2000	8.613	8.613	0.1039	12.07
1.3000	10.430	10.430	-0.6273	60.14
1.4000	11.765	11.764	-1.0963	93.19
1.5000	12.232	12.231	-1.1369	92.95
1.6000	11.823	11.822	-0.7574	64.06
1.7000	10.841	10.841	-0.1270	11.71
1.8000	9.739	9.739	0.5013	51.47
1.9000	8.929	8.930	0.8941	100.13
2.0000	8.646	8.647	0.9262	107.12

The final test on cyclic phenomena showed greater differences between NDTRAN version 1 and version 2 than in any of the other tests that me made (many of which are not shown in this chapter). Therefore we are showing the programand the three outputs from the same RLC circuit done using NDTRAN version 1.

Figure 6.13
Series RLC Circuit
Program

```
STEP RESPONSE OF SERIES RLC CIRCUIT

**********    S O U R C E    L I S T I N G    **********

10001   * STEP RESPONSE OF SERIES RLC CIRCUIT
10002   *NOWARN
10003   *NOSTATS
10004   *EULER
10005   N VC=0
10006   N IL=0
10007   L VC=INTEGRAL(ILR)
10008   L IL=INTEGRAL(VIR)
10009   R ILR=IL/C
10010   R VIR=10/L-R*IL/L-VC/L
10011   C C=4.94E-2
10012   C R=1
10013   C L=.5
10014   C A=1
10015   C B=6.2831853
10016   S VCS=10*(1+EXP(-A*TIME)*(-A/B*SIN(B*TIME)-COS(B*TIME)))
10017   SPEC DT=.02,LENGTH=2,PRTPER=.1
10018   S ABSERR=VCS-VC
10019   S RELERR=ABS(ABSERR)/VCS
10020   PRINT VC,VCS,ABSERR,RELERR
```

Figure 6.14
Series RLC Circuit
EULER Output

TIME	VC	VCS	ABSERR	RELERR
0.0000	0.000	0.000	0.0000	.00000
0.1000	1.543	1.833	0.2902	.15830
0.2000	6.070	6.231	0.1608	.02581
0.3000	11.551	11.168	-0.3830	.03430
0.4000	15.831	14.796	-1.0347	.06993
0.5000	17.447	16.065	-1.3814	.08599
0.6000	16.098	14.953	-1.1444	.07653
0.7000	12.624	12.286	-0.3381	.02752
0.8000	8.565	9.292	0.7269	.07823
0.9000	5.502	7.091	1.5891	.22409
1.0000	4.467	6.321	1.8541	.29332
1.1000	5.616	6.996	1.3793	.19717
1.2000	8.273	8.613	0.3408	.03957
1.3000	11.273	10.430	-0.8429	.08082
1.4000	13.457	11.764	-1.6924	.14386
1.5000	14.102	12.231	-1.8706	.15293
1.6000	13.142	11.822	-1.3193	.11160
1.7000	11.117	10.841	-0.2758	.02544
1.8000	8.905	9.739	0.8347	.08570
1.9000	7.353	8.930	1.5771	.17661
2.0000	6.966	8.647	1.6809	.19439

Figure 6.15
Series RLC Circuit
Runge-Kutta Output

TIME E+00	VC E+00	VCS E+00	ABSERR E-04	RELERR E-05
0.0000	0.000	0.000	0.000	0.000
0.1000	1.834	1.833	-3.130	17.075
0.2000	6.232	6.231	-9.657	15.499
0.3000	11.169	11.168	-13.969	12.508
0.4000	14.797	14.796	-11.894	8.039
0.5000	16.066	16.065	-3.065	1.908
0.6000	14.952	14.953	8.982	6.007
0.7000	12.284	12.286	18.736	15.250
0.8000	9.289	9.292	21.528	23.170
0.9000	7.090	7.091	15.783	22.258
1.0000	6.321	6.321	3.715	5.877
1.1000	6.997	6.996	-9.746	13.932
1.2000	8.615	8.613	-19.177	22.264
1.3000	10.432	10.430	-20.976	20.112
1.4000	11.766	11.764	-14.769	12.554
1.5000	12.232	12.231	-3.377	2.761
1.6000	11.821	11.822	8.520	7.207
1.7000	10.839	10.841	16.371	15.101
1.8000	9.738	9.739	17.525	17.994
1.9000	8.929	8.930	12.108	13.559
2.0000	8.646	8.647	2.729	3.156

Figure 6.16
Series RLC Circuit
Adams-Bashforth Output

TIME E+00	VC E+00	VCS E+00	ABSERR E-06	RELERR E-07
0.0000	0.000	0.000	0.0	0.0
0.1000	1.833	1.833	-120.9	659.6
0.2000	6.231	6.231	-291.4	467.7
0.3000	11.169	11.168	-624.3	559.0
0.4000	14.797	14.796	-927.3	626.8
0.5000	16.066	16.065	-969.7	603.6
0.6000	14.954	14.953	-634.1	424.1
0.7000	12.286	12.286	4.8	3.9
0.8000	9.291	9.292	708.5	762.5
0.9000	7.090	7.091	1186.3	1673.0
1.0000	6.320	6.321	1231.4	1948.1
1.1000	6.995	6.996	815.7	1166.0
1.2000	8.613	8.613	101.5	117.8
1.3000	10.430	10.430	-629.8	603.9
1.4000	11.765	11.764	-1098.0	933.3
1.5000	12.232	12.231	-1137.2	929.7
1.6000	11.823	11.822	-756.2	639.7
1.7000	10.841	10.841	-124.9	115.2
1.8000	9.739	9.739	503.4	516.9
1.9000	8.929	8.930	895.6	1002.9
2.0000	8.646	8.647	926.5	1071.5

DY

Chapter 7
NDTRAN Simulation: Applications

A) Introduction

While the previous chapters had a number of application problems, the discussions there were specifically designed to illustrate the manner in which NDTRAN statement types were used. The present chapter is specifically concerned with various aspects of the application of NDTRAN in various areas. Many simulation studies in the social sciences as well as in hard sciences involve feedback of one kind or another, a concept essential to system science studies.

The discussion below shows how NDTRAN solved three simple applications problems as well as one more complex environmental study. For users obtaining NDTRAN2, the magnetic tape containing the NDTRAN2 interpreter contains, among other files, the complete WORLD3 model by Dennis Meadows along with each of the separate sectors in the model including Population Sector, Capital Sector and so on. The programs are exported with Meadow's consent. The documents for the WORLD3 model may be obtained either from MIT Press or from Meadows at Dartmouth College.

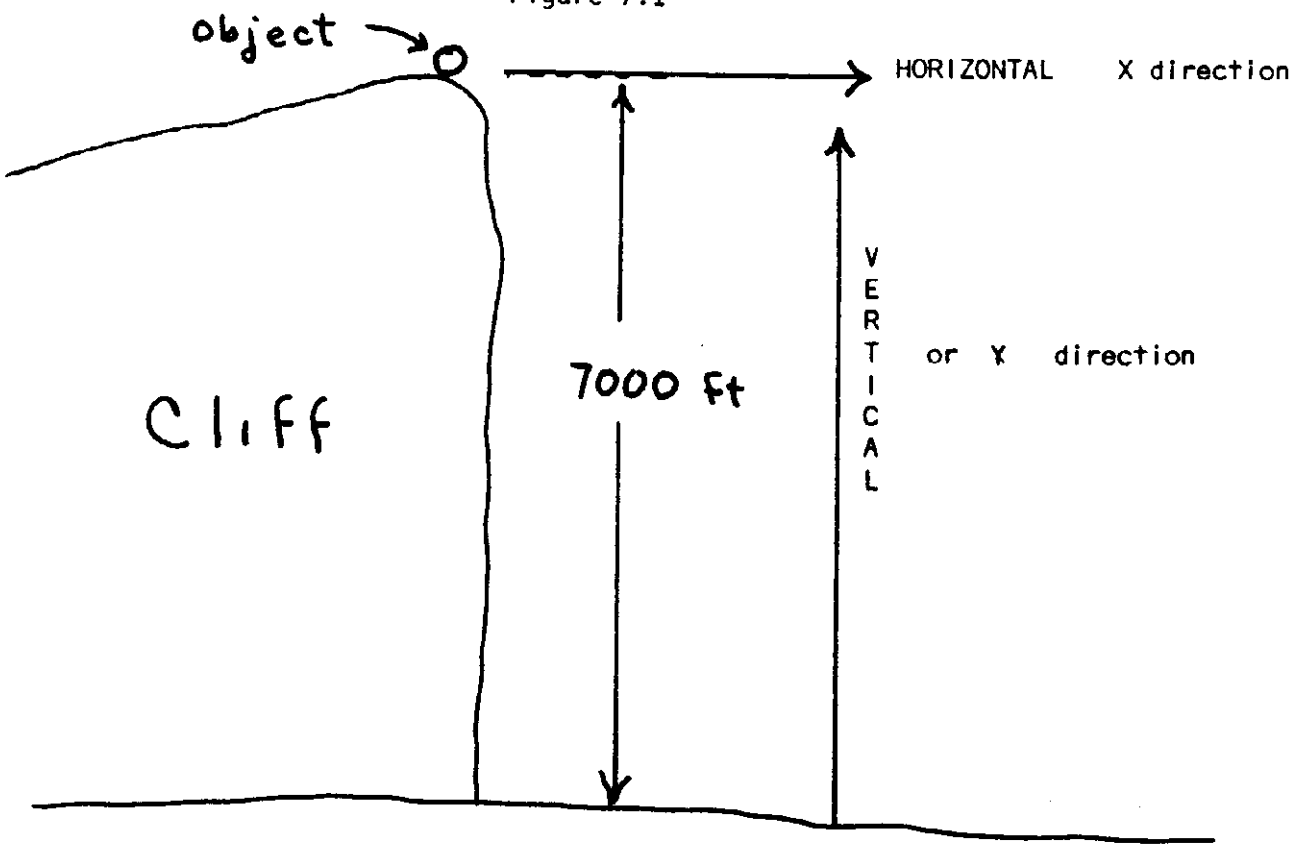
B) Simple Problems

1) Mechanics Problem: Falling Body

Let us now examine the solution to a "falling body " problem. We wish to model the path of an object hurled from a 7000 ft cliff that has an initial velocity of 2000 ft. per second. We neglect air disturbance and assume that gravity is the only force exerted on the object (32.3 ft./second^2). As it is subject to the

horizontal thrust, it is released. The problem is to determine where the object will strike the ground, given the above conditions. Figure 7.1 shown below illustrates the basic problem and its parameters. The coordinate system is centered at the base of the cliff.

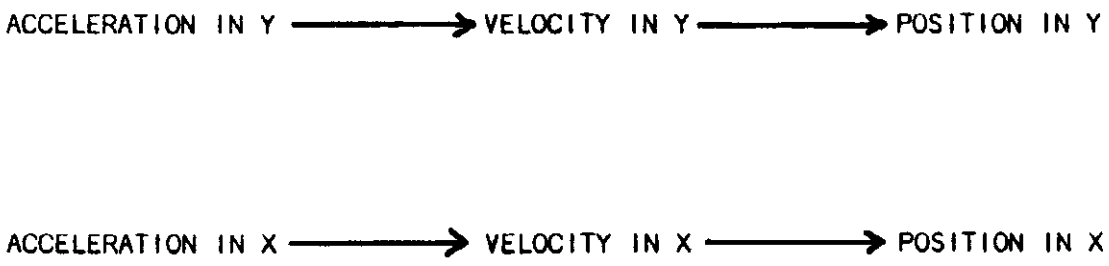
Figure 7.1



In terms of a cartesian coordinate system, the velocity of the object relative to the abscissa (x-axis) is constant at 2000 ft. per second. The velocity of the object increases because of the impact of gravity (over time). Thus the height (linear position) of the object therefore decreases toward zero. The causal loop diagram in Figure 7.2 reflects

the variable interactions as the object falls in space.

Figure 7.2
Causal Loop Diagram

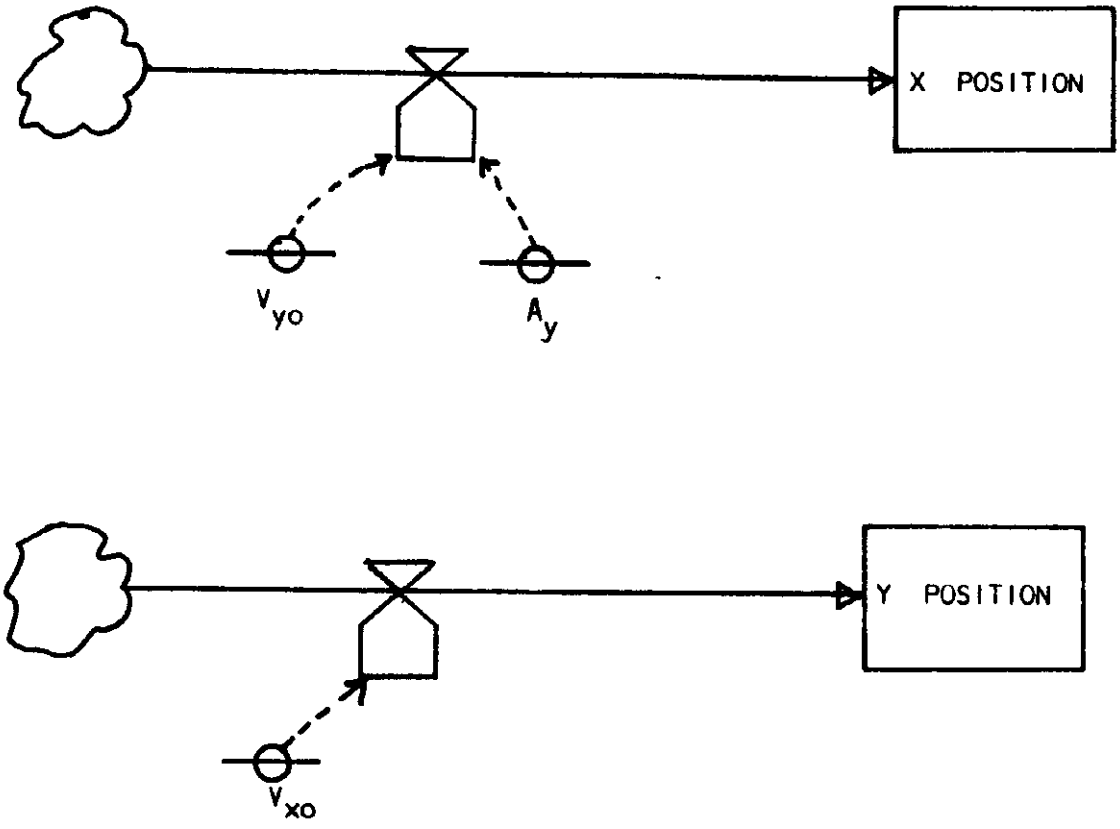


Knowledge of the problem can afford two methods for solution in this situation. The movement in the X and Y directions are independent and can be found by integrating the respective velocities. Since the X velocity is initially 2000 ft. per second and no other forces act in this direction, we simply integrate it. The Y direction has a constant acceleration and therefore linear velocity is:

$$V_y = V_{y_0} + A_y * T \qquad (7.1)$$

Thus from these ideas and the causal loop diagram of Figure 7.2, the flow diagram of Figure 7.3 is developed.

Figure 7.3
Flow Diagram



The program which will properly simulate this model in NDTRAN is given in Figure 7.4

Figure 7.4
Cliff Model

```
PAGE 1     AN OBJECT THROWN FROM A CLIFF                    (C) 1978 UND

     * * * * *    S O U R C E    L I S T I N G    * * * * *

0001   TITLE AN OBJECT THROWN FROM A CLIFF
0002   * NOSTATS
0003   * NARROW
0004   * CHECK
0005   * EULER
0006   L Y.K=INTGRL(VYR.JK)
0007   R VYR.KL=VYO+AY.K*TIME.K
0008   A AY.K=GRAV
0009   L X.K=INTGRL(VXR.JK)
0010   R VXR.KL=VXO
0011   N Y=7000
0012   N X=0
0013   C VYO=0
0014   C VXO=2000
0015   C GRAV=-32.2
0016   PARM DT=.01
0017   PARM STOP=21
0018   PARM START=0
0019   PARM PRTPER=1
0020   PARM PLTPER=1
0021   NOTE IMPACT OF GRAVITY
0022   PRINT X,Y
0023   PLOT X=X/Y=Y
```

As can be seen from the table in Figure 7.5 and the graph on Figure 7.6, the movement in the horizontal direction will be linear with time. The movement along the vertical direction will be parabolic because of the integration of a straight line.

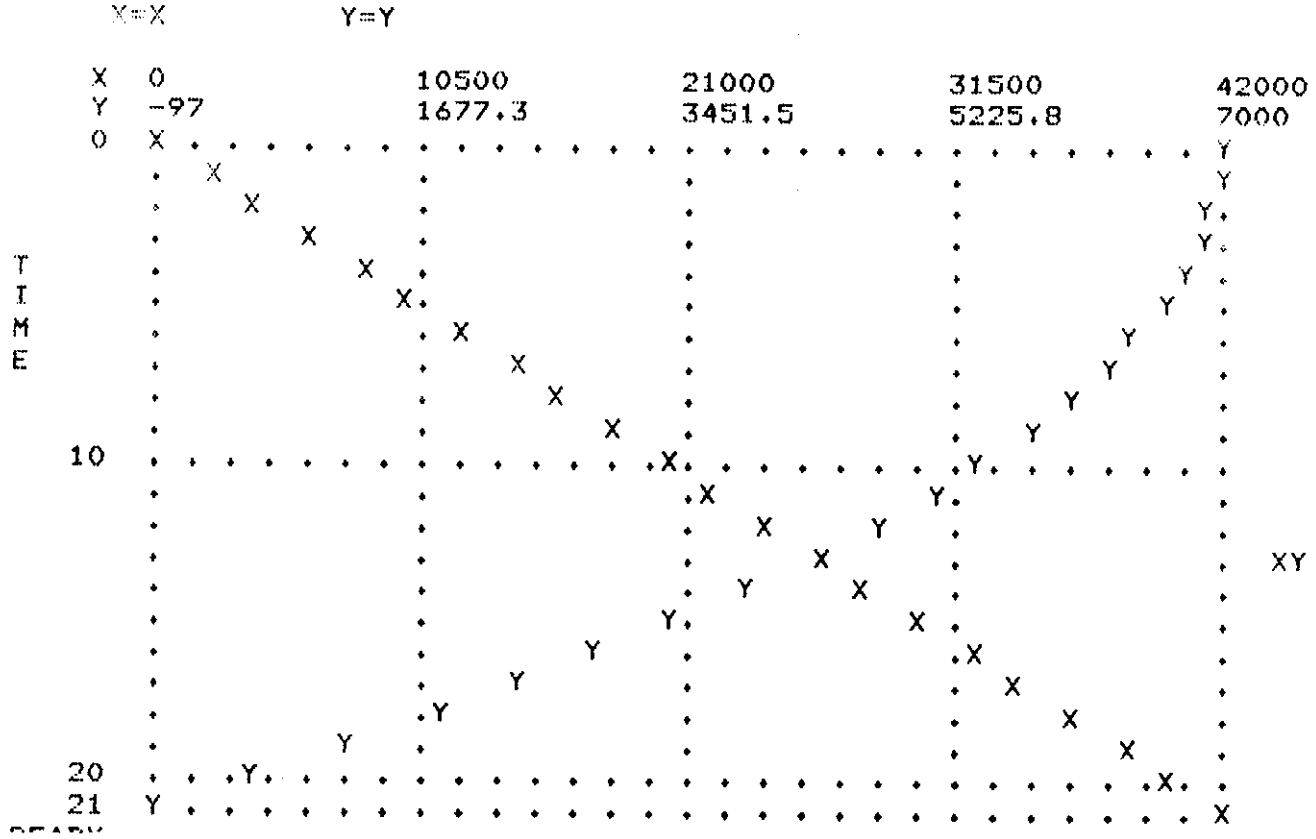
Figure 7.5
Program Output

TIME E+00	X E+03	Y E+03
0.000	0.000	7.0000
1.000	2.000	6.9841
2.000	4.000	6.9359
3.000	6.000	6.8556
4.000	8.000	6.7430
5.000	10.000	6.5983
6.000	12.000	6.4214
7.000	14.000	6.2122
8.000	16.000	5.9709
9.000	18.000	5.6973
10.000	20.000	5.3916
11.000	22.000	5.0537
12.000	24.000	4.6835
13.000	26.000	4.2812
14.000	28.000	3.8467
15.000	30.000	3.3799
16.000	32.000	2.8810
17.000	34.000	2.3498
18.000	36.000	1.7865
19.000	38.000	1.1910
20.000	40.000	0.5632
21.000	42.000	-0.0967

There is no way to stop the program dynamically, thus one will note that the Y value will go negative implying impact with the ground. Impact with the ground will occur between the 20th and the 21st second after being thrown.

It is possible to do this problem a second way. If it was not desirable to use Equation 7.1 then the velocity in the Y direction would have to be calculated before position. But integral relationships still exist between the acceleration and the velocity of the object.

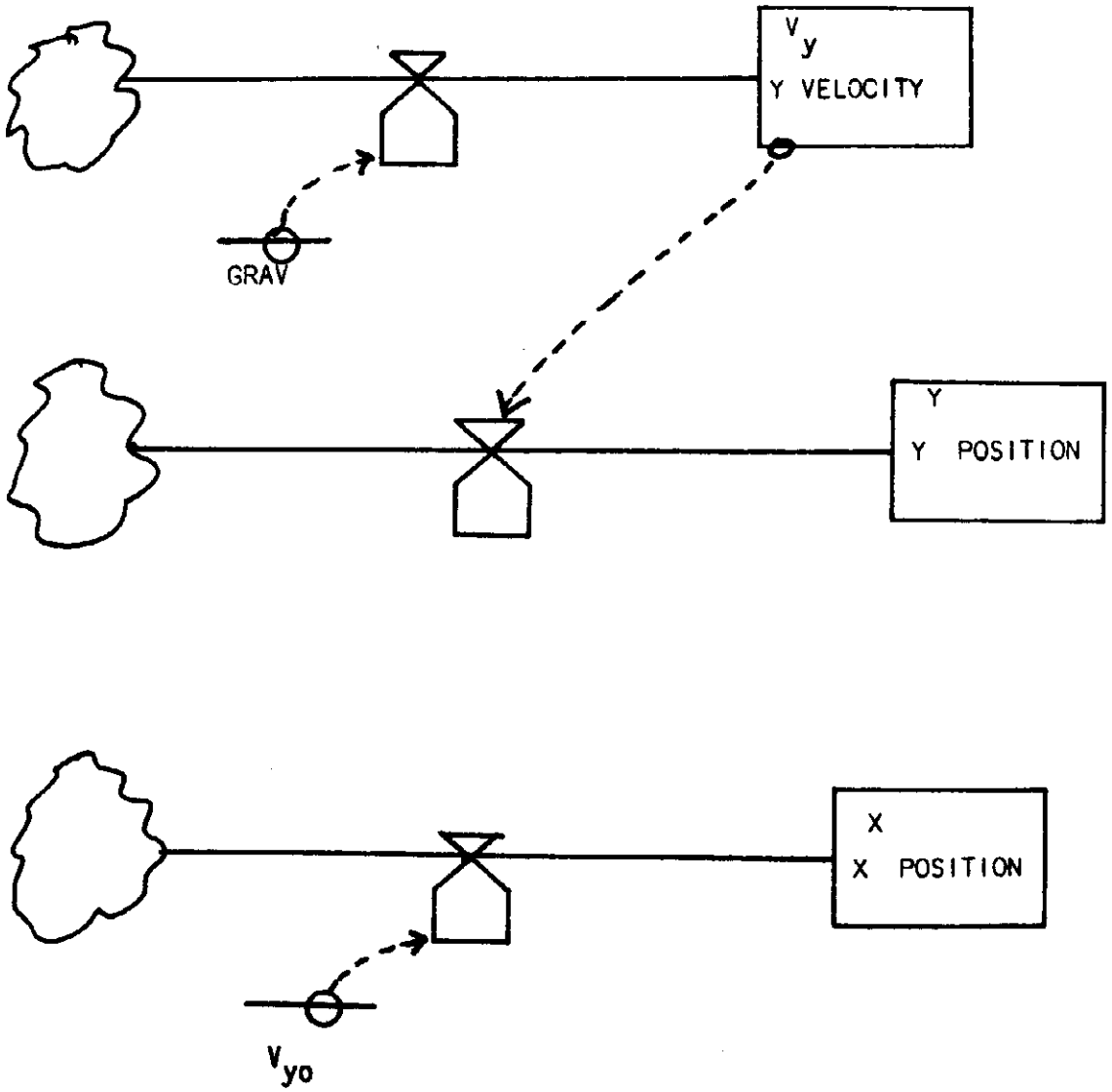
Figure 7.6
Cliff Problem
Plot of X and Y



Thus we now obtain the flow diagram of Figure 7.7

Three integrations are now required; two for the Y position and one for the X position. In the case of the Y variable, the velocity is obtained by integrating the acceleration and the position is obtained by integrating the velocity. In this type of problem some care has to be extended <u>since the velocity variable appears as both a rate and a variable</u>.

Figure 7.7
Flow Diagram



Avoiding this can only be accomplished by some linear transformation of the states which would cause a loss of the desired defined variables.

A source listing and the corresponding results for this problem variations are given in Figures 7.8, 7.9 and 7.10 respectively. One should note that the tables and graphs for both problem approaches are identical.

Figure 7.8
Cliff Model

```
      AN OBJECT THROWN FROM A CLIFF                          (C) 1978 UND

*****    S O U R C E    L I S T I N G    *****

0001  TITLE AN OBJECT THROWN FROM A CLIFF
0002  * NOSTATS
0003  * NOWARN
0004  * NARROW
0005  * CHECK
0006  * EULER
0007  L VY.K=INTGRL(AY.JK)
0008  R AY.KL=GRAV
0009  L Y.K=INTGRL(VYR.JK)
0010  R VYR.KL=VY.K
0011  L X.K=INTGRL(VXR.JK)
0012  R VXR.KL=VX0
0013  N Y=7000
0014  N X=0
0015  N VY=0
0016  C VY0=0
0017  C VX0=2000
0018  C GRAV=-32.2
0019  PARM DT=.01
0020  PARM STOP=21
0021  PARM PLTPER=1
0022  PARM PRTPER=1
0023  NOTE IMPACT OF GRAVITY
0024  PRINT X,Y
0025  PLOT X=X/Y=Y
```

Figure 7.9
Table

TIME	X	Y
0.000	0	7000.0
1.000	2000	6984.1
2.000	4000	6935.9
3.000	6000	6855.6
4.000	8000	6743.0
5.000	10000	6598.3
6.000	12000	6421.4
7.000	14000	6212.2
8.000	16000	5970.9
9.000	18000	5697.3
10.000	20000	5391.6
11.000	22000	5053.7
12.000	24000	4683.5
13.000	26000	4281.2
14.000	28000	3846.7
15.000	30000	3379.9
16.000	32000	2881.0
17.000	34000	2349.8
18.000	36000	1786.5
19.000	38000	1191.0
20.000	40000	563.2
21.000	42000	-96.7

Figure 7.10
Plot

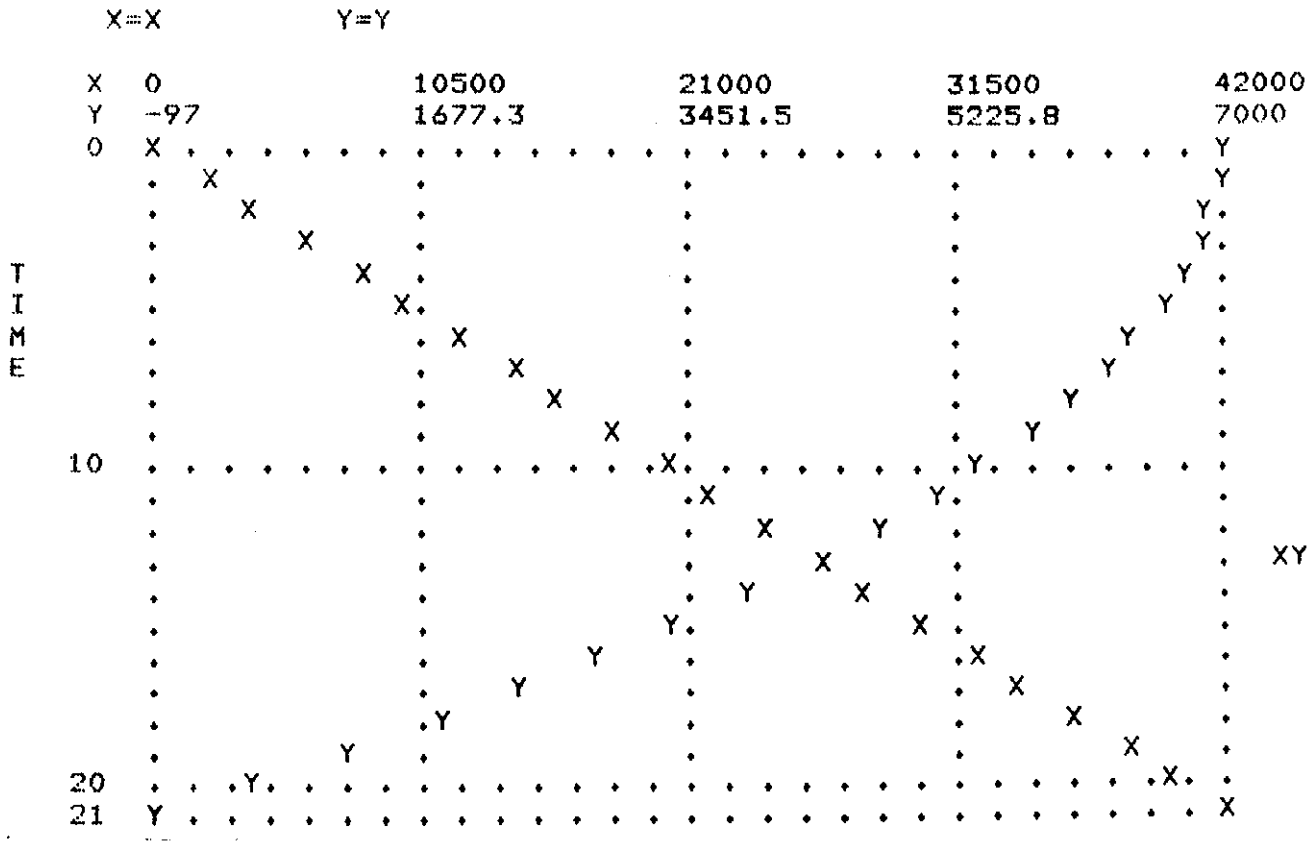


Figure 7.4.1
Cliff Model

```
**********    S O U R C E    L I S T I N G    **********

10001   * AN OBJECT THROWN FROM A CLIFF
10002   *NOSTATS
10003   *NARROW
10004   *SOURCE
10005   *XREF
10006   *EULER
10007   *WARN
10008   L Y.K=INTEGRAL(VYR.JK)
10009   R VYR.KL=VY0+AY.K*TIME.K
10010   A AY.K=GRAV
10011   L X.K=INTEGRAL(VXR.JK)
10012   R VXR.KL=VX0
10013   N Y=7000
10014   N X=0
10015   C VY0=0,VX0=2000,GRAV=-32.2
10016   SPEC DT=.01,START=0,STOP=21,PRTPER=1,PLTPER=1
10017   NOTE IMPACT OF GRAVITY
10018   PRINT X,Y
10019   PLOT X=X/Y=Y
```

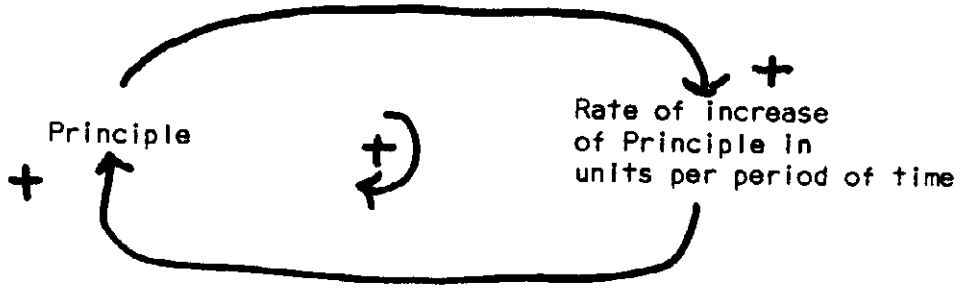
Figure 7.8.1
Cliff Model

```
**********    S O U R C E    L I S T I N G    **********

10001   * AN OBJECT THROWN FROM A CLIFF
10002   *NOSTATS
10003   *NARROW
10004   *SOURCE
10005   *XREF
10006   *EULER
10007   *WARN
10008   L VY.K=INTEGRAL(AY.JK)
10009   R AY.KL=GRAV
10010   L Y.K=INTEGRAL(VYR.JK)
10011   R VYR.KL=VY.K
10012   L X.K=INTEGRAL(VXR.JK)
10013   R VXR.KL=VX0
10014   N Y=7000
10015   N X=0
10016   N VY=0
10017   C VY0=0,VX0=2000,GRAV=-32.2
10018   SPEC DT=.01,START=0,STOP=21,PRTPER=1,PLTPER=1
10019   NOTE IMPACT OF GRAVITY
10020   PRINT X,Y
10021   PLOT X=X/Y=Y
```

2) Savings Account Problem

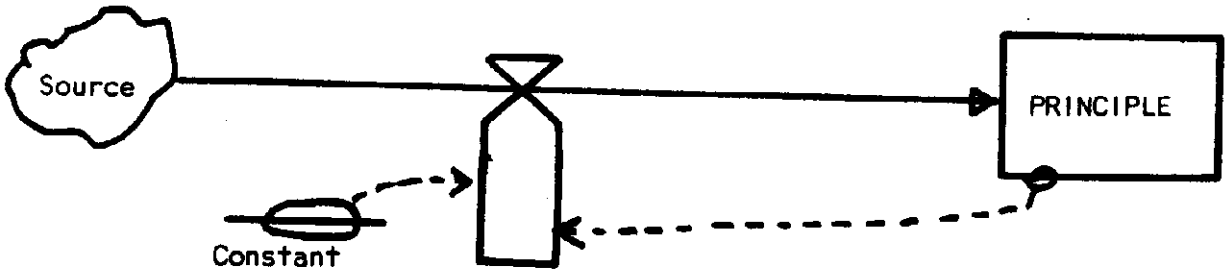
The growth of a savings account illustrates a first order (positive) feedback system. The causal loop diagram that is appropriate appears in Figure 7.11.

Figure 7.11
Causal Loop Diagram: Interest Problem



The Flow Chart to illustrate this is shown in Figure 7.12. Remember, any system involving feedback will exhibit exponential changes in a level or state variable.

Figure 7.12
Flow Diagram: Interest Accumulation



Exponential rise or decay implies that the next value of a level is a defined percentage of the current value. That is the level (quantity) at time $t+1$ is a defined percentage of the level (quantity) at time t .

In the case of exponential change, the level is given an initial value and increases or decreases over time. The shape of the rise or decline curve depends on the size of the constant involved. For exponential rise or decay, the larger the constant, the steeper the curve.

The sum of $1000.00 is deposited in a bank account which pays 5 percent per annum. Determine the amount accumulated in the account if it is compounded:

a) annually

b) quarterly

c) daily

The basic idea of compounding interest is fairly simple. The money remains in the savings account for a given period (say one year) and at the end of the period the sum in the bank is multiplied by the rate of interest for the given period. An example may help:

Amount in bank for one year	$1000.00	
rate at which earnings occur	.05	(5 percent per year)
interest earned in dollars/year	50.00	
Total in bank at end of year	$1050.00	
Amount in bank for second year	$1050.00	
rate at which earnings occur	.05	(5 percent per year)
interest earned in dollars/year for the second year	52.50	
Total in bank at end of second year	$1102.50-	

The process is continued for as many years (periods) as is desired.

There is only one caveat that must be mentioned. That is, what if the rate of interest is given in terms of percent per year on a savings account, and the point is also noted that the bank account is compounded quarterly or daily. This raises a very important point not only for persons with a savings account, but also for those wishing to do dynamic modeling. The point is that the rates at which the level or state variables change must be compatible with the solution interval or calculation time period for the model.

In the example given above, the calculation period was one year. That is, the interest was added to the account (compounded) annually. Thus if the interest earnings are given at 5 percent per year and the interest is calculated once each year; the rate at which the savings account accumulates interest is compatible with the calculation period for adding the savings earned to the original balance.

What if, however, the account earns interest at 5 percent per year (.05) and the interest is added to the account every three months (compounded quarterly). In this case one would not multiply the balance of the account by .05 every three months. If one did, that would amount to an effective earning power of 20 percent per year. The earning power is 5 percent per year compounded quarterly. Thus, one would need to divide the annual rate by 4 to obtain the nominal quarterly rate of .0125 or 1.25 percent per quarter which is compatible with the 5 percent per year. However, the DT period has been modified to 3 months rather than to 12 months and the nominal interest remains at 5 percent per year. The programs for annual and quarterly compounding of nominal 5 percent annual interest on accounts are shown below.

Figure 7.13
Interest Compounded Annually

```
*****   S O U R C E   L I S T I N G   *****

0001  TITLE INTEREST PROBLEM
0002  * EULER
0003  * CHECK
0004  * NOWARN
0005  * NOSTATS
0006  * NARROW
0007  L P.K=INTGRL(INTR.JK)
0008  N P=1000
0009  R INTR.KL=P.K*IR
0010  C CON=1
0011  C IR=.05
0012  PARM DT=1
0013  PARM STOP=20
0014  PARM PRTPER=1
0015  PARM PLTPER=1
0016  NOTE INTEREST ACCUMULATION
0017  PRINT P
0018  PLOT P
```

Figure 7.13.1
Interest Compounded Annually

```
**********   S O U R C E   L I S T I N G   **********

10001  * INTEREST PROBLEM
10002  *EULER
10003  *NOSTATS
10004  *NARROW
10005  L P.K=INTEGRAL(INTR.JK)
10006  N P=1000
10007  R INTR.KL=P.K*IR
10008  C CON=1
10009  C IR=.05
10010  S PF.K=1000*(1+IR)**TIME.K
10011  SPEC DT=1,START=0,STOP=20,PRTPER=1,PLTPER=1
10012  NOTE INTEREST ACCUMULATION
10013  PRINT P,PF
10014  PLOT P
```

Figure 7.14
Interest Compounded Quarterly

```
*****   S O U R C E   L I S T I N G   *****

0001  TITLE INTEREST PROBLEM
0002  * EULER
0003  * CHECK
0004  * NOWARN
0005  * NOSTATS
0006  * NARROW
0007  L P.K=INTGRL(INTR.JK)
0008  N P=1000
0009  R INTR.KL=P.K*IR
0010  C CON=1
0011  C IR=.05
0012  PARM DT=.25
0013  PARM STOP=20
0014  PARM PRTPER=1
0015  PARM PLTPER=1
0016  NOTE INTEREST ACCUMULATION
0017  PRINT P
0018  PLOT P
PAGE 2    INTEREST PROBLEM - INTEREST ACCUMULATION          (C) 1978 UND
```

Figure 7.14.1
Interest Compounded Quarterly

```
**********   S O U R C E   L I S T I N G   **********

10001  * INTEREST PROBLEM
10002  *EULER
10003  *NOSTATS
10004  *NARROW
10005  L P.K=INTEGRAL(INTR.JK)
10006  N P=1000
10007  R INTR.KL=P.K*IR
10008  C CON=1
10009  C IR=.05
10010  SPEC DT=.25,START=0,STOP=20,PRTPER=1,PLTPER=1
10011  NOTE INTEREST ACCUMULATION
10012  PRINT P
10013  PLOT P
```

Figure 7.15
Interest Compounded Annually

TIME E+00	P E+03
0.000	1.0000
1.000	1.0500
2.000	1.1025
3.000	1.1576
4.000	1.2155
5.000	1.2763
6.000	1.3401
7.000	1.4071
8.000	1.4775
9.000	1.5513
10.000	1.6289
11.000	1.7103
12.000	1.7959
13.000	1.8856
14.000	1.9799
15.000	2.0789
16.000	2.1829
17.000	2.2920
18.000	2.4066
19.000	2.5270
20.000	2.6533

Figure 7.16
Interest Compounded Quarterly

TIME	P
0.000	1000.0
1.000	1050.9
2.000	1104.5
3.000	1160.8
4.000	1219.9
5.000	1282.0
6.000	1347.4
7.000	1416.0
8.000	1488.1
9.000	1563.9
10.000	1643.6
11.000	1727.4
12.000	1815.4
13.000	1907.8
14.000	2005.0
15.000	2107.2
16.000	2214.5
17.000	2327.4
18.000	2445.9
19.000	2570.5
20.000	2701.5

The fact of compounding quarterly means that the interest earned is entered into the account more rapidly. Thus the earned interest itself begins to earn interest at an earlier period. As noted in the figures above the only program change to move from annual compounding to quarterly compounding is the DT size.

3) Poisson Distribution

Certain types of simulations involve a random arrival of events at a particular station. This may be obtained in NDTRAN using a random number generator and two table functions. The particular group of students who needed this type of distribution were developing a model of plane arrivals and departures from an airport. The arrivals were considered to be random. The program below shows the method used for calculating the arrivals based on a poisson inter-arrival time.

The first step involved a uniform random number generator (available in NDTRAN) with a value generated between 0 and 1. This function was used with two TABND functions. One of the TABND functions yields an average of one arrival per DT for normal hours. The second TABND function yields 2.5 arrivals per DT for rush hours. A PULSE function was used to signal the rush hours condition. In the example shown below only one TABND is shown. The program for the generation of the Poisson Distribution is shown in Figure 7.17 with the tabular output in Figure 7.18 and the graphic output in Figure 7.19. The TABND function represents the inverse of the cumulative distribution function for the specified average arrival time.

Figure 7.17

Poisson Distribution:
Program

```
PAGE 1     THIS PROGRAM GENERATES POISSON DISTRIBUTED RANDO  (C) 1978 UND

     * * * * *    S O U R C E    L I S T I N G    * * * * *

     0001  TITLE THIS PROGRAM GENERATES POISSON DISTRIBUTED RANDOM VARIAB
LES
     0002  * NOSTATS
     0003  * NARROW
     0004  * CHECK
     0005  * EULER
     0006  L LEV.K=INTGRL(TEST.JK)
     0007  N LEV=1
     0008  R TEST.KL=LEV.K*RND.K
     0009  NOTE
     0010  NOTE GENERATE PSEUDO RANDOM NUMBER BETWEEN 0 AND 1
     0011  NOTE
     0012  A RND.K=NOISE(SEED)
     0013  C SEED=12345
     0014  NOTE
     0015  NOTE GENERATE POISSON DISTRIBUTED RANDOM VARIABLE
     0016  NOTE
     0017  S POISON.K=TABND(TAB1,RND.K,0,1)
     0018  T TAB1=0.0,1.2,1.75,2.3,2.7,3.0,3.25,3.50,3.8,4.05,4.4,4.65,5.
0,
           X 5.3,5.65,6.0,6.5,7.0,7.5,8.5,20.0
     0019  PARM DT=1
     0020  PARM START=0
     0021  PARM STOP=20
     0022  PARM PRTPER=1
     0023  PARM PLTPER=1
     0024  NOTE POISSON PRINTED OUTPUT
     0025  PRINT RND,POISON
     0026  NOTE PLOTTED OUTPUT FOR POISSON
     0027  PLOT RND=R(0,1)/POISON=+(0,20)
     0028  PLOT RND=R(0,1)//POISON
```

Figure 7.18

Poisson Distribution:
Tabular Output

THIS PROGRAM GENERATES POISSON DISTRIBUTED RANDO -(P) 1978 UND

TIME E+00	RND E-03	POISON E+00
0.000	588.26	4.917
1.000	842.90	6.929
2.000	763.06	6.122
3.000	992.31	17.147
4.000	86.30	1.627
5.000	587.04	4.908
6.000	745.48	5.964
7.000	189.58	2.627
8.000	428.10	3.937
9.000	862.43	7.105
10.000	321.66	3.355
11.000	168.09	2.459
12.000	113.65	1.906
13.000	169.07	2.468
14.000	991.58	16.900
15.000	427.86	3.936
16.000	642.94	5.257
17.000	6.96	0.230
18.000	255.25	3.028
19.000	468.87	4.181
20.000	515.99	4.481

Figure 7.19
Poisson Distribution:
Graphic Output

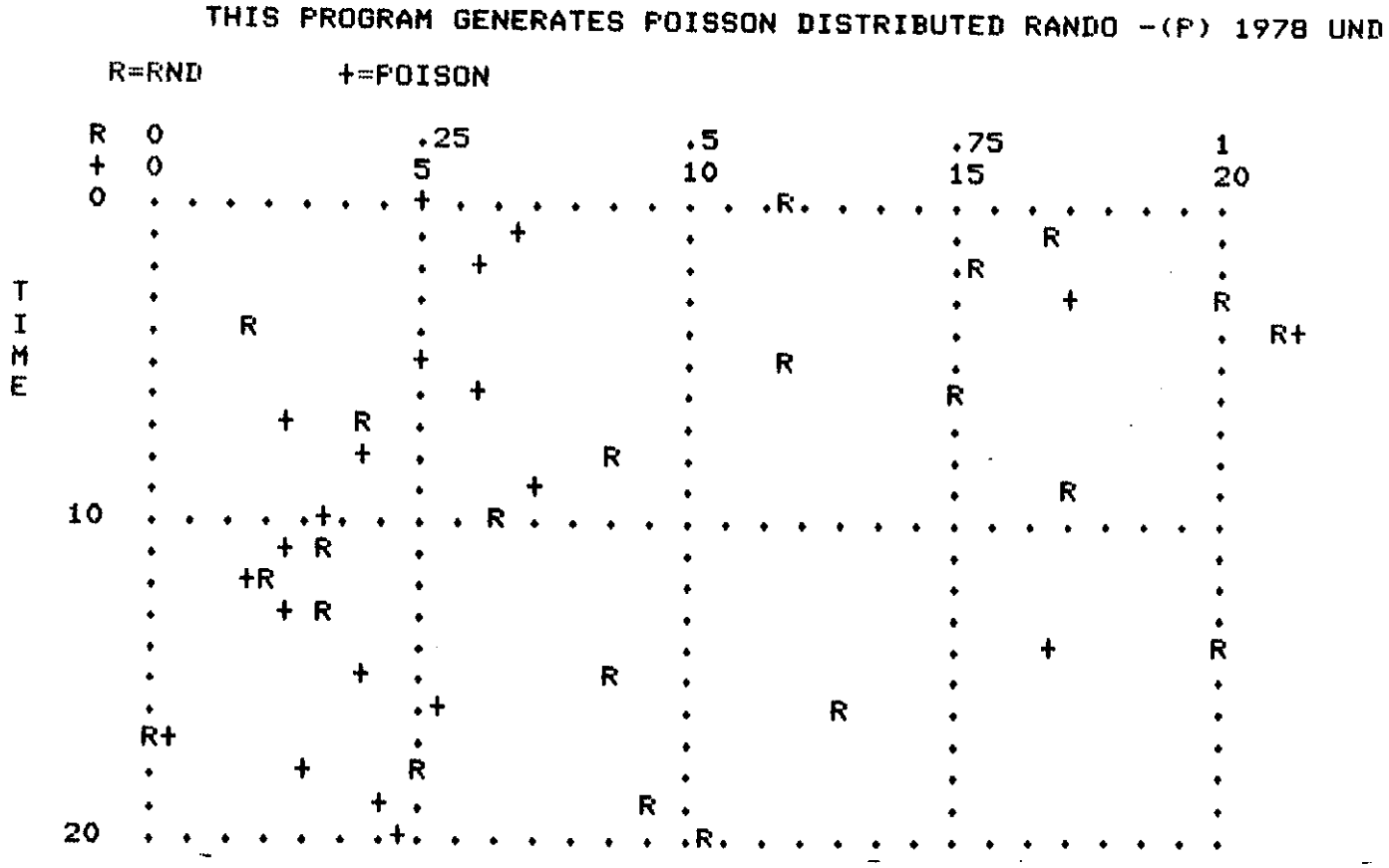


Figure 7.17.1

Poisson Distribution:
Program

```
THIS PROGRAM GENERATES POISSON DISTRIBUTED RANDO  C 1976 UND

**********    S O U R C E    L I S T I N G    **********

10001  * THIS PROGRAM GENERATES POISSON DISTRIBUTED RANDOM V
ARIABLES
10002  *NOSTATS
10003  *NARROW
10004  *EULER
10005  L LEV.K=INTEGRAL(TEST.JK)
10006  N LEV=1
10007  R TEST.KL=LEV.K*RND.K
10008  NOTE
10009  NOTE GENERATE PSEUDO RANDOM NUMBER BETWEEN 0 AND 1
10010  NOTE
10011  A RND.K=NOISE(SEED)
10012  C SEED=12345
10013  NOTE
10014  NOTE GENERATE POISSON DISTRIBUTED RANDOM VARIABLE
10015  NOTE
10016  A POISSON.K=TABND(TAB1,RND.K,0,1) ,
10017  T TAB1=0.0/1.2/1.75/2.3/2.7/3.0/3.25/3.50/3.8/4.05/4.
4/4.65/5.0/
10018  X 5.3/5.65/6.0/6.5/7.0/7.5/8.5/20.0
10019  SPEC DT=1,START=0,STOP=20,PRTPER=1,PLTPER=1
10020  NOTE POISSON PRINTED OUTPUT
10021  PRINT RND,POISSON
10022  NOTE PLOTTED OUTPUT FOR POISSON
10023  PLOT RND=R(0,1)/POISSON=+(0,20)
10024  PLOT RND=R(0,1)//POISSON
```

C) Complex Problem

Cedar Bog Lake, a lake in Minnesota has been modeled by R.B. Williams, and the results have been published.* A.A.B. Pritsker, of Purdue University, has a problem in his book concerning the Cedar Bog Lake Model. **

---

* B.C. Patten(ed.), SYSTEMS ANALYSIS AND SIMULATION IN ECOLOGY, vol. 1, Academic Press, NY, 1971, pp. 543 - 582.

** Pritsker, A.A.B., The GASP IV Simulation Language, John Wiley & Sons, NY, p. 324.

The problem worked out below is a modification of the problem as originally set up by Williams, and also as suggested by Pritsker.

There are five basic levels in the ecosystem represented by the lake which are:

1) ENVIRO -Environmental losses of energy to the lake
2) CARN -Carnivores in the lake
3) HERB -Herbivores in the lake
4) PLANT -Plants in the lake
5) ORGAN -Organic buildup in the lake

These five level variables may be described as the state variables whose behavior over time describe the system. These variables, together with SUNLIGHT, an exogenous variable, combine to determine the overall energy in the lake in terms of calories/centimeter^2. The lake entities are modeled in terms of their energy content, the energy transfers among the lake entities, and energy losses to the environment.

The annual cycle of solar radiation was simulated using the following equation:

$$SUN=95.9 * (1.0 + .635 * \sin(t))$$

t = 0 at the vernal equinox.

The equations of the model are:

PLANT = 95.9 - (1.00 + 2.55 + .48)PLANT

HERB = .48(Plant) - (6.90 + 6.12 + 4.85)HERB

CARN = 4.85(HERB) - (2.70 + 1.95)CARN

ORGAN = 2.55(PLANT) + 6.12 (Herb) + 1.95(Carn)

ENVIRO = 1.00(Plant) + 6.90(Herb) + 2.70(Carn)

The table shown by Williams shows the coefficients:

VARIABLE	Photo-Synthesis cal cm^{-2} yr^{-1}	Transfer Rate	Respiration Rate	Loss Rate
PLANT	95.9	.48	1.00	2.55
HERB		4.85	6.90	6.12
CARN			2.70	1.95

For instance, the sun shining on the lake adds energy to the lake. For the given area it adds 95.9 calories per cm^2. That is the sun adds energy to the lake at the rate of 95.9 calories per cm^2 . The other variables are energy measured in calories, the unit reference being the cm^2 . Employing these coefficients, the ecosystem may be represented by the causal loop diagram in Figure 7.20 and the Flow Diagram in Figure 7.21.

For a moment let us digress. In many types of modeling it is usual to associate two or more rates with a single level variable. For instance, for a particular age group of the population there may be three rates:

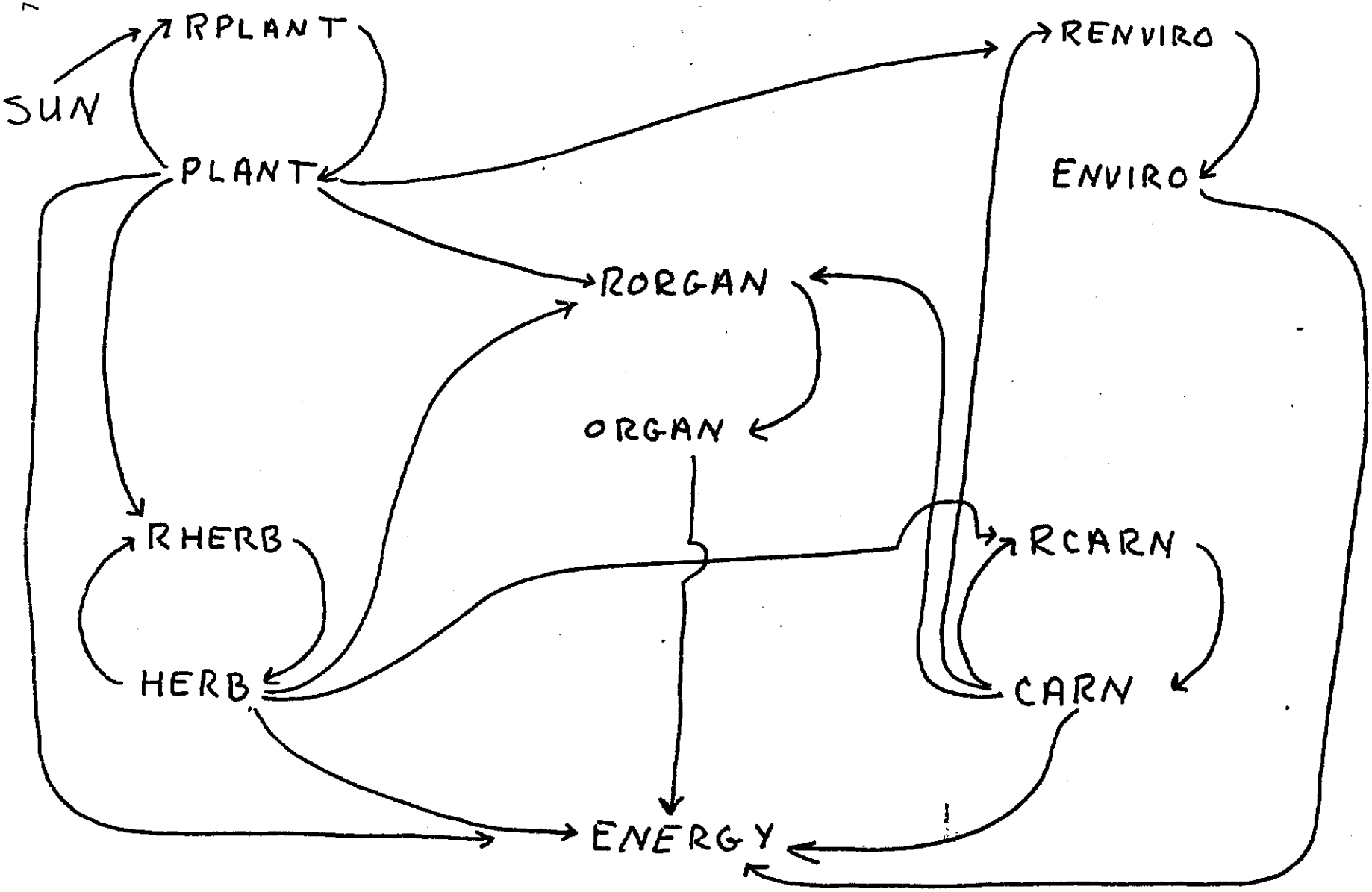
For the population age group 0 - 18 years;

1) a rate to represent births per year;

2) a rate to represent deaths per year;

3) a rate to represent those maturing to the next age group--i.e., 19 - 24 years.

Figure 7.20
Cedar Lake Model:
Causal Loop Diagram



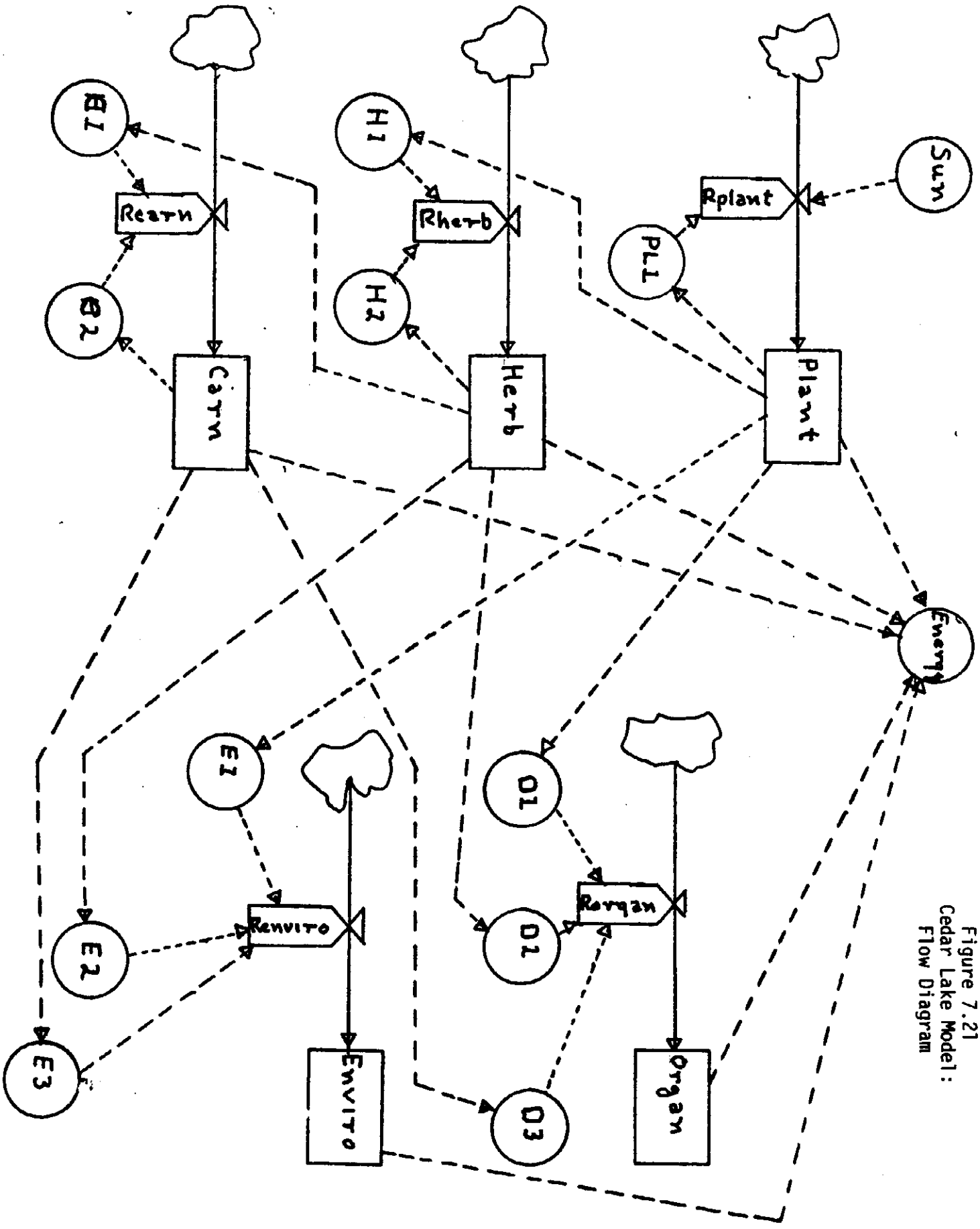
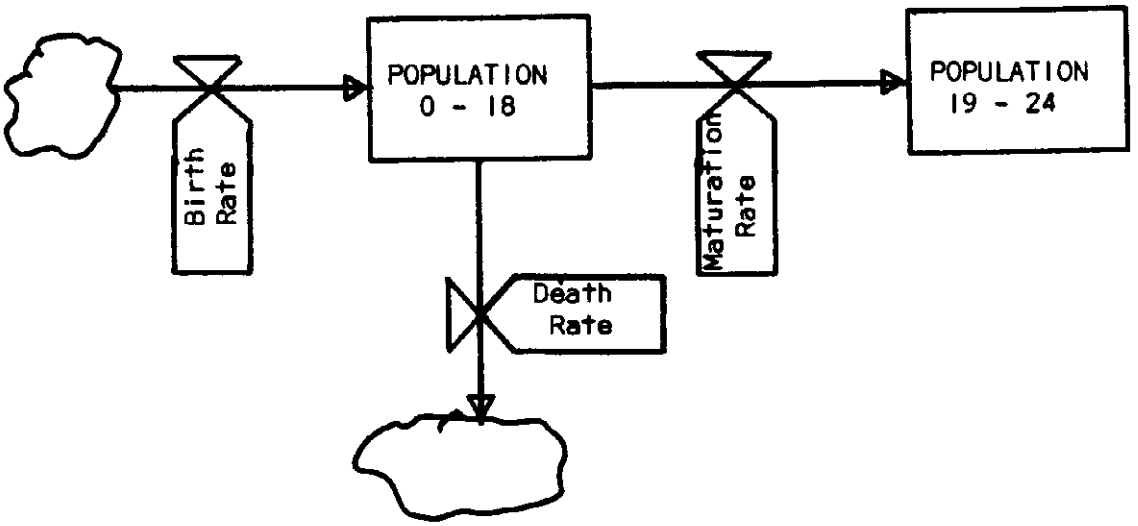


Figure 7.21
Cedar Lake Model:
Flow Diagram

The flow chart for this population level would look as shown below:

Figure 7.22



The equations involved are as follows:

L POP.K=INTEGRAL(BR.JK-DR.JK-MAT.JK)

R BR.KL=...

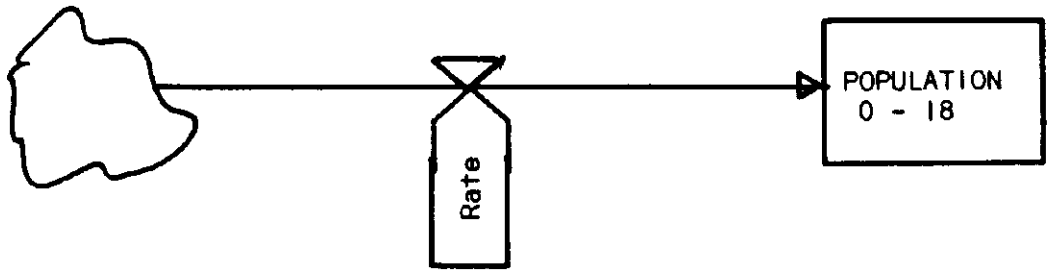
R DR.KL=...

R MAT.KL=...

It is perfectly possible to handle this problem differently, by using only a single rate equation, i.e. by lumping all rates with a composite

variable. The flow chart would be as shown below:

Figure 7.23



And the required equations would be:

L POP.K=EULER(RATE.JK)

R RATE.KL=BR.KL-DR.KL-MAT.KL

In the existing rate equation all of the necessary conditions would need to be considered. It is this latter approach that is generally used in the CEDAR LAKE model.

From both the Causal Loop Diagram of Figure 7.20 and the Flow Diagram of Figure 7.21 it is apparent that there are five levels in the model. Three of the levels involve exponential change as they are involved with feedback. The other two, the Environmental and the Organic material levels are residuals. All five of the level variables feed the Energy variable, which measures the energy in the lake.

The technique of programming the model is to use only a single rate variable for each level variable, with that variable reflecting the

total change in the state variable. The program is developed in a modular form, with each module representing a specific subsystem of the lake. The constants are generally defined at the end of the program in the section noted as CONSTANTS. The data coefficients appropriate for the gains and losses of energy for each variable are shown as numeric constants in the given equations, and are taken from the before mentioned section, where the equations of the model were discussed.

Essentially the structure of the model is as follows:

A) The basic or initial form of energy to the lake is by means of the sun. The sun has an impact on the PLANTS that produce energy by means of photosynthesis.

B) The second level in the lake is the herbivorous animals or those that use the plants as food. The coefficients for TRANSFER RATE reflect the rate at which one variable is consumed by another--i.e., plants are consumed by herbivores. The Respiration Rate is the oxygen exchange rate for the plants and animals, and the loss rate is the rate at which the plants and animals are losing energy to the environment. Experimental runs indicate that the lake environment changes little except with respect to changes in the level of plants. Changes at the level of plants causes significant changes in the energy levels in the lake.

Output is measured as energy in the lake, which is given as calories per centimeter-squared. The program is shown in Figure 7.24 and the table output is in Figure 7.25 with the plot output in Figure 7.26.

Figure 7.24
Cedar Lake Model

```
0001  TITLE CEDAR LAKE ENERGY BALANCE
0002  * SOURCE
0003  * CHECK
0004  * NOWARN
0005  * NARROW
0006  * EULER
0007  NOTE
0008  NOTE
0009  NOTE PLANTS
0010  N PLANT=24.0
0011  A PL1.K=(CC1+C255+C48)*PLANT.K
0012  A P2.K=CLIP(P65,ONE,PLANT.K,N28)*PLANT.K
0013  R RPLANT.KL=SUN.K-PL1.K
0014  L PLANT.K=INTGRL(RPLANT.JK)
0015  NOTE
0016  NOTE
0017  NOTE HERBEVIORS
0018  N HERB=.6
0019  A H1.K=C48*PLANT.K
0020  A H2.K=(C690+C612+C485)*HERB.K
0021  R RHERB.KL=(CC1)(H1.K-H2.K)
0022  L HERB.K=INTGRL(RHERB.JK)
0023  NOTE
0024  NOTE
0025  NOTE CARNIVORS
0026  N CARN=.6
0027  A C1.K=C485*HERB.K
0028  A C2.K=(C270+C195)(CARN.K)
0029  R RCARN.KL=C1.K-C2.K
0030  L CARN.K=INTGRL(RCARN.JK)
0031  NOTE
0032  NOTE
0033  NOTE ORGANIC
0034  N ORGAN=0
0035  A O1.K=C255*PLANT.K
0036  A O2.K=C612*HERB.K
0037  A O3.K=C270*CARN.K
0038  R RORGAN.KL=O1.K+O2.K+O3.K
0039  L ORGAN.K=INTGRL(RORGAN.JK)
0040  NOTE
0041  NOTE
0042  NOTE ENVIRONMENT
0043  N ENVIRO=0
0044  A E1.K=CC1*PLANT.K
0045  A E2.K=C690*HERB.K
0046  A E3.K=C270*CARN.K
0047  R RENVRO.KL=E1.K+E2.K+E3.K
0048  L ENVIRO.K=INTGRL(RENVRO.JK)
0049  NOTE
0050  NOTE
0051  NOTE SUN'S ENERGY INPUT TO THE LAKE
0052  A SUN.K=C959*(CC1+C0635*SIN(TIME.K*CC2*C314))
0053  NOTE AVERAGE ENERGY IN THE LAKE
0054  A ENERGY.K=(PLANT.K+HERB.K+CARN.K+ORGAN.K-ENVIRO.K+SUN.K)/TIME
.K
```

Figure 7.24-continued

```
 2     CEDAR LAKE ENERGY BALANCE
0055   NOTE CONSTANTS
0056   C P003=.003
0057   C P0416=.0416
0058   C P1=.1
0059   C P3=.3
0060   C P5=.5
0061   C P6=.6
0062   C P65=.65
0063   C P7=.7
0064   C ONE=1
0065   C ZERO=0
0066   C N28=28
0067   C CC1=1.00
0068   C C255=2.55
0069   C C48=.48
0070   C C690=6.90
0071   C C612=6.12
0072   C C485=4.85
0073   C C270=2.70
0074   C C959=95.9
0075   C C0635=.0635
0076   C CC2=2.00
0077   C C314=3.14
0078   C C195=1.95
0079   NOTE CONTROL CARDS
0080   PARM DT=.01
0081   PARM START=1
0082   PARM STOP=51
0083   PARM PRTPER=2
0084   PARM PLTPER=2
0085   PRINT PLANT,CARN,ORGAN,ENVIRO,ENERGY
0086   PLOT PLANT/CARN/ORGAN/ENVIRO/ENERGY
```

Figure 7.24.1
Cedar Lake Model:
Systems Dynamics Program

```
CEDAR LAKE ENERGY BALANCE

**********    S O U R C E    L I S T I N G    **********

10001  * CEDAR LAKE ENERGY BALANCE
10002  *SOURCE
10003  *NOOBJECT
10004  *WARN
10005  NOTE
10006  NOTE
10007  NOTE PLANTS
10008  N PLANT=24.0
10009  A PL1.K=(1.00+2.55+.48)*PLANT.K
10010  A P2.K=CLIP(P65,ONE,PLANT.K,N28)*PLANT.K
10011  R RPLANT.KL=SUN.K-PL1.K
10012  L PLANT.K=INTEGRAL(RPLANT.JK)
10013  NOTE
10014  NOTE
10015  NOTE HERBEVIORS
10016  N HERB=.6
10017  A H1.K=.48*PLANT.K
10018  A H2.K=(6.90+6.12+4.85)*HERB.K
10019  R RHERB.KL=(1)(H1.K-H2.K)
10020  L HERB.K=INTEGRAL(RHERB.JK)
10021  NOTE
10022  NOTE
10023  NOTE CARNIVORS
10024  N CARN=.6
10025  A C1.K=4.85*HERB.K
10026  A C2.K=(2.70+1.95)(CARN.K)
10027  R RCARN.KL=C1.K-C2.K
10028  L CARN.K=INTEGRAL(RCARN.JK)
10029  NOTE
10030  NOTE
10031  NOTE ORGANIC
10032  N ORGAN=0
10033  A O1.K=2.55*PLANT.K
10034  A O2.K=6.12*HERB.K
10035  A O3.K=2.70*CARN.K
10036  R RORGAN.KL=O1.K+O2.K+O3.K
10037  L ORGAN.K=INTEGRAL(RORGAN.JK)
```

Figure 7.24.1 Continued

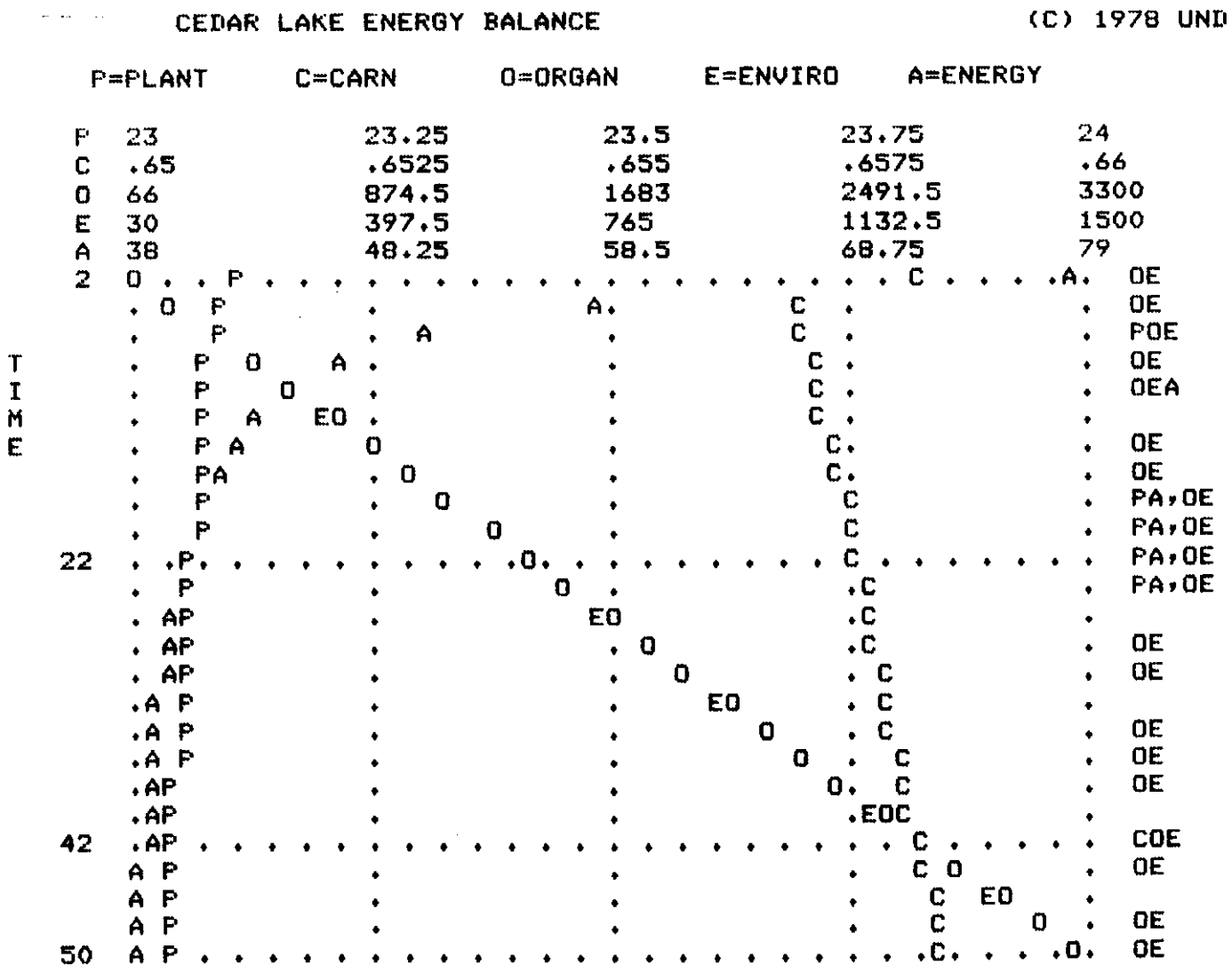
```
10038  NOTE
10039  NOTE
10040  NOTE ENVIORONMENT
10041  N ENVIRO=0
10042  A E1.K=1.00*PLANT.K
10043  A E2.K=6.90*HERB.K
10044  A E3.K=2.70*CARN.K
10045  R RENVIRO.KL=E1.K+E2.K+E3.K
10046  L ENVIRO.K=INTEGRAL(RENVIRO.JK)
10047  NOTE
10048  NOTE
10049  NOTE SUN'S ENERGY INPUT TO THE LAKE
10050  A SUN.K=95.9*(1.0+.0635*SIN(TIME.K*2.0*3.14))
10051  NOTE AVERAGE ENERGY IN THE LAKE
CEDAR LAKE ENERGY BALANCE

10052  A ENERGY.K=(PLANT.K+HERB.K+CARN.K+ORGAN.K-ENVIRO.K+SUN.K)/TIME.K
10053  NOTE CONSTANTS
10054  C P003=.003
10055  C P0416=.0416
10056  C P1=.1
10057  C P3=.3
10058  C P5=.5
10059  C P6=.6
10060  C P65=.65
10061  C P7=.7
10062  C ONE=1
10063  C ZERO=0
10064  C N28=28
10065  NOTE CONTROL CARDS
10066  SPEC DT=.01,PRTPER=5,START=1,STOP=51
10067  PRINT PLANT,HERB,CARN,ORGAN,ENVIRO,SUN,ENERGY
```

Figure 7.25
Cedar Lake Model:
Table Output

CEDAR LAKE ENERGY BALANCE (C) 1978 UND

TIME E+00	PLANT E+00	CARN E-03	ORGAN E+03	ENVIRO E+03	ENERGY E+00
2.000	23.102	658.22	0.0670	0.0302	78.484
4.000	23.084	657.00	0.1998	0.0903	57.422
6.000	23.082	657.05	0.3326	0.1503	50.403
8.000	23.079	657.11	0.4653	0.2103	46.894
10.000	23.076	657.17	0.5981	0.2703	44.788
12.000	23.074	657.23	0.7309	0.3303	43.385
14.000	23.071	657.29	0.8637	0.3903	42.382
16.000	23.069	657.35	0.9965	0.4504	41.630
18.000	23.066	657.41	1.1293	0.5104	41.045
20.000	23.064	657.48	1.2621	0.5704	40.577
22.000	23.061	657.54	1.3949	0.6304	40.194
24.000	23.059	657.60	1.5277	0.6904	39.875
26.000	23.057	657.66	1.6604	0.7504	39.605
28.000	23.054	657.73	1.7932	0.8104	39.374
30.000	23.052	657.79	1.9260	0.8705	39.174
32.000	23.050	657.85	2.0588	0.9305	38.998
34.000	23.047	657.92	2.1916	0.9905	38.843
36.000	23.045	657.98	2.3244	1.0505	38.706
38.000	23.043	658.05	2.4572	1.1105	38.583
40.000	23.041	658.11	2.5900	1.1705	38.472
42.000	23.039	658.18	2.7227	1.2305	38.371
44.000	23.036	658.24	2.8555	1.2906	38.280
46.000	23.034	658.31	2.9883	1.3506	38.197
48.000	23.032	658.37	3.1211	1.4106	38.121
50.000	23.030	658.44	3.2539	1.4706	38.051

Figure 7.26
Cedar Lake Model:
Graphic Output



NDTRAN2

********** ERROR MESSAGES **********

100 --- The card type indicator which begins in column one is not recognized by NDTRAN. That card will not be processed.

101 --- The previous card may not be continued. This continuation card will not be processed.

102 --- The indicated card exceeds the limit of one continuation per statement and will not be processed.

103 --- A MEND statement was encountered but a MACRO was not being processed. The MEND statement will be ignored.

104 --- A RERUN card may not appear within a MACRO group. The card will be ignored.

105 --- A card type other than T, C, PARM, or * has appeared in RERUN mode and cannot be processed.

106 --- A program may not start with a continuation statement. The card will be ignored.

107 --- All control cards must immediately follow the TITLE card. If no TITLE card is provided, they must precede any other type of statement.

108 --- Only one TITLE card may be specified and it must be the first card of the program.

109 --- a macro may not be defined within another MACRO group. The MACRO card will be ignored.

110 --- The indicated card contains no equation or data and therefore, cannot be processed.

111 --- An EXPND card may not appear within a MACRO group. The card will be ignored.

112 --- The end of the program was encountered before the MEND in the MACRO mode.

113 --- The indicated RERUN statement is not followed by a valid variable change. This RERUN will not be executed.

114 --- A new integration technique has already been requested for this RERUN. This card will be ignored.

115 --- Integration type is the only control card allowed in RERUN.

116 --- Too many changes have been requested in the RERUN. This card and all further ones in this RERUN will be ignored.

152 --- There is already a DEF card for this variable.

153 --- No definition occurs on this DEF card. It is ignored.

204 --- The indicated control card specifies an option which is not recognized by NDTRAN.

205 --- The indicated control card requests an option which was previously specified. This request will be ignored.

206 --- 'No' may not be specified before an integration option.

302 --- a m a c r o contains more than 18 arguments.

303 --- Missing parenthese delimiter for argument list.

304 --- Argument list for MACRO is missing.

305 --- The user has already defined a MACRO with this name.

308 --- The same dummy argument name occurs twice.

309 --- Excess information occurs after indicated end of MACRO or EXPND argument list.

310 --- There was an EXPND statement encountered for an undefined MACRO, i.e., MACRO name encountered in EXPND statement is neither a built-in MACRO nor a user-defined MACRO.

311 --- A MACRO in an EXPND statement does not have the number of arguments required by either the MACRO definition or by the built-in MACRO requirements.

312 --- A data field of an expanded statement has exceeded the 72 character length of one card.

313 --- There is a missing MACRO name.

314 --- An arithmetic expression occurs in the MACRO name field.

315 --- Expansion has caused the comment field to be truncated. The data field has remained intact.

316 --- A MACRO may not be expanded due to critical errors in its definition.

401 --- A duplicate equal sign has appeared in an arithmetic expression. The expression cannot be evaluated.

402 --- Invalid sequence of operational symbols.

403 --- Missing right parenthesis at end of statement.

404 --- No arithmetic expression exists on the right side of an equal sign.

405 --- Number of right parenthesis exceeds number of left parenthesis.

406 --- Expected operator is missing.

407 --- In an arithmetic expression a comma may only be used to delimit the arguments of a function.

410 --- The indicated SPEC card parameter may not be specified to be negative or zero. A default will be supplied.

414 --- The STOP time specified is less than the START time previously encountered. A default value will be substituted.

415 --- The START time specified is greater than the STOP time previously encountered. A default value will be substituted.

416 --- PRTPER was specified but this program has no PRINT statements. The PRTPER will be ignored.

417 --- PLTPER was specified but this program has no PLOT statements. The PLTPER will be ignored.

501 --- The first character of a variable name must be either alphabetic (a-z) or the $.

502 --- The indicated character is illegal in a variable name. Characters other than the first must be alphabetic (a-z), numeric (0-9), the $ or underscore characters.

503 --- The indicated variable exceeds the maximum length of 6 characters.

504 --- The indicated time subscript is invalid. It must be either J, K, L, JK, or KL.

505 --- A variable may not be dependent on its own present value.

506 --- A function does not have the correct number of arguments.

508 --- The time subscript for this variable is missing.

512 --- This table name was used previously.

516 --- No equation exists for this variable; it is therefore undefined.

518 --- Reserved words, functions, and tables may not be specified in a PRINT statement.

519 --- A variable is expected to occur in the indicated position.

521 --- The first argument of any TABLE function must be a TABLE.

522 --- A TABLE may only be used as the first argument of a TABLE function.

523 --- NDTRAN does not support this function.

524 --- This level variable must be given an initial value.

525 --- An N equation does not initialize a LEVEL. Its type has been changed to a constant.

526 --- 'intgrl' may only be used in a level equation.

527 --- A function may not be used in a level equation; only integrators are permissable.

530 --- Variable should be unsubscripted.

532 --- Variable should be subscripted 'K'.

533 --- Variable should be subscripted 'JK'.

534 --- Variable should be subscripted 'KL'.

536 --- The indicated variable is not defined in the model and cannot be defined in a RERUN.

537 --- Changing the value of the indicated variable will not affect the model during RERUN since the variable is not used in any equation.

538 --- DT is the only parameter that may be changed during a RERUN. The indicated parameter may not be specified.

539 --- The number of elements in a table array change for a RERUN is not consistent with the number in the model.

540 --- The indicated variable has been previously defined as a table.

541 --- The indicated variable has been previously defined as a constant.

544 --- The indicated variable has been previously defined as a level.

545 --- The indicated variable has been previously defined as an auxiliary.

546 --- The indicated variable has been previously defined as a rate.

547 --- The indicated variable has been previously defined as a supplementary.

548 --- A variable type may not change in a rerun.

550 --- TIME should be initialized through the START parameter.

551 --- The user may not write an equation for a parameter.

559 --- The variable on the left of the equal sign has already been initialized.

560 --- An equation cannot be written for TIME.

571 --- Constants may not be used in this equation.

572 --- Parameters may not be used in this equation.

574 --- Levels may not be used in this equation.

575 --- Auxiliaries may not be used in this equation.

576 --- Rates may not be used in this equation.

577 --- Supplementaries may not be used in this equation.

587 --- This variable is used only for output. It should have been defined as a supplementary.

588 --- This variable is not used for output and has no effect on any other variable.

589 --- A DEF card occurs for an undefined variable.

600 --- There is an illegal character in the numeric literal.

601 --- The expected numeric literal string is missing.

602 --- A '+' or a '-' sign may appear only at the start of the numeric literal or the start of the exponent. It is illegal as used where indicated.

603 --- Mantissa exceeds the 8 digit limit.

604 --- Exponent exceeds the 2 digit limit.

605 --- There is a duplicate decimal point.

606 --- Exponent may not contain a decimal point.

607 --- The number being exponentiated has a value of zero.

608 --- There is a duplicate exponent character.

609 --- An exponent character is encountered before the mantissa.

610 --- Expected mantissa is missing.

611 --- The exponent character is not followed by an exponent.

612 --- The exponent is too large or too small.

701 --- A delimiter (comma, slash, or parenthesis) is repeated. The redundant delimeters are ignored.

702 --- PRINT cards are not allowed to have slashes.

703 --- A PRINT or PLOT card should not begin or end with a comma or slash.

704 --- This PRINT or PLOT card contains no variables.

705 --- Character sequence /) or ,) was encountered. NDTRAN assumes user intended)/ or),

706 --- Two non-consecutive open parentheses were encountered; NDTRAN assumes the second to be a closed parenthesis.

707 --- There was no comma or slash after the indicated closed parenthesis.

708 --- A closed parenthesis was encountered for which there was no open parenthesis.

709 --- A second comma was encountered after an open parenthesis. A closed parenthesis is missing, and assumed to be there.

710 --- A slash is encountered after an open parenthesis; a closed parenthesis is missing.

711 --- A PRINT or PLOT card may contain no more than ten variables.

712 --- A PRINT card may not have an '='.

713 --- A PLOT character may be only one character long.

714 --- Any character may appear only once on a PLOT card.

715 --- There is no PLOT character after the indicated equal sign.
Note: NDTRAN does not accept the following as PLOT characters: '=', ',', '(', ')', and '/'.

716 --- Range for this PLOT series has been previously defined.

717 --- The narrow option allows only 5 variables on a PLOT card and 6 on a PRINT card.

718 --- A table array should not begin or end with a comma.

719 --- The high value is missing from this range; the default value will be used.

720 --- A run number is expected after a period in an output variable name.

721 --- A run number must be one character between 1 and the number of reruns.

722 --- Neither parentheses nor range specifications may occur on a PRINT card.

723 --- PLOT characters are not to be specified for the independent variable.

724 --- The comparative output variable may be the only variable on the output card.

725 --- The asterisk cannot appear as a run number except for a comparative output plot.

726 --- The NARROW option restricts the number of variables outputted; some of the runs will not produce output.

727 --- The independent variable in this plot is a constant.

728 --- This output card has requested a run number that is greater than the number of runs in the program.

801 --- This card's equation contains no equal sign and cannot be processed.

802 --- The table contains no elements.

803 --- A table should contain more than one element. It will be handled as a constant.

804 --- A table array may not contain variables or arithmetic expressions.

805 --- Only a single variable may occur on the left side of an equal sign.

806 --- Only a numeric literal is permissible on the right side of this equation.

901 --- Within an integrator, only addition and subtraction of rate variables may occur.

902 --- Arithmetic expressions may not occur outside of an integrator.

903 --- The 'INTGRL' function must occur immediately after the equal sign in any level equation.

904 --- A level equation may not contain numeric literals.

905 --- The only function permissible in a level equation is 'INTGRL'.

999 --- A computed constant or computed DT may not be redefined in a rerun.

!
READY

NDTRAN

********** ERROR MESSAGES **********

100 --- The card type indicator which begins in column one is not recognized by NDTRAN. That card will not be processed.

101 --- The indicated card is the continuation of a card whose type was unrecognizable. That card will not be processed.

102 --- The indicated card exceeds the limit of one continuation per statement and will not be processed.

103 --- A MEND statement was encountered but a MACRO was not being processed. The MEND statement will be ignored.

104 --- RUN or RERUN mode has been indicated and a MACRO was being processed. All of the statements for that MACRO will be ignored.

105 --- PRINT, PLOT, N, S, A, R, L, CPLOT, MACRO, MEND, and EXPND statements may not appear in RERUN mode. The indicated card will be ignored.

110 --- The indicated card contains no equation or data and therefore, cannot be processed.

201 --- A control card may not be continued. This card will be ignored.

202 --- The indicated card is a duplicate title control card and will be ignored.

204 --- The indicated control card specifies an option which is not recognized by NDTRAN.

205 --- The indicated control card requests an option which was previously specified. This request will be ignored.

206 --- 'No' may not be specified before an integration option.

301 --- SPEC, PRINT, PLOT, and CPLOT cards may not appear in a MACRO . The indicated card will be ignored.

302 --- A MACRO contains more than 18 arguments.

303 --- Missing parenthese delimiter for argument list.

304 --- Argument list for MACRO is missing.

305 --- The user has already defined a MACRO with this name.

306 --- A new MACRO definition occurs before for last MEND for the last MACRO.

307 --- An argument may not contain an arithmetic operation.

308 --- The same dummy argument name occurs twice.

309 --- Excess information occurs after indicated end of MACRO or EXPND argument list.

310 --- There was an EXPND statement encountered for an undefined MACRO, i.e., MACRO name encountered in EXPND statement is neither a built-in MACRO nor a user-defined MACRO.

311 --- A MACRO in an EXPND statement does not have the number of arguments required by either the MACRO definition or by the built-in MACRO requirements.

312 --- An expanded statement has generated more than one continuation.

401 --- A duplicate equal sign has appeared in an arithmetic expression. The expression cannot be evaluated.

402 --- Invalid sequence of operational symbols.

403 --- Missing right parenthesis at end of statement.

404 --- No arithmetic expression exists on the right side of an equal sign.

405 --- Number of right parenthesis exceeds number of left parenthesis.

406 --- Expected operator is missing.

407 --- In an arithmetic expression a comma may only be used to delimit the arguments of a function.

410 --- The indicated SPEC card parameter may not be specified to be negative or zero. A default will be supplied.

414 --- The STOP time specified is less than the START time previously encountered. A default value will be substituted.

415 --- The START time specified is greater than the STOP time previously encountered. A default value will be substituted.

416 --- PRTPER was specified but this program has no PRINT statements. The PRTPER will be ignored.

417 --- PLTPER was specified but this program has no PLOT statements. The PLTPER will be ignored.

501 --- The first character of a variable name must be either alphabetic (A-Z) or the $.

502 --- The indicated character is illegal in a variable name. Characters other than the first must be alphabetic (A-Z), numeric (0-9), the $ or underscore characters.

503 --- The indicated variable exceeds the maximum length of 8 characters.

504 --- The indicated time subscript is invalid. It must be either J, K, L, JK, or KL.

505 --- A variable may not be dependent on its own present value.

506 --- A function does not have the correct number of arguments.

507 --- The variable being initialized is not a LEVEL.

510 --- The indicated variable may not appear on a SPEC card.

512 --- This table name was used previously.

516 --- No equation exists for this variable; it is therefore undefined.

518 --- Reserved words, functions, and tables may not be specified in a PRINT statement.

519 --- A variable is expected to occur in the indicated position.

520 --- This SPEC card parameter has already been assigned a value. This specification will be ignored.

521 --- A table name may only be the first argument of a table function.

523 --- NDTRAN does not support this function.

524 --- This level variable must be given an initial value.

526 --- Variable or integrator may not be used as a function.

527 --- A function may not be used in a level equation; only integrators are permissable.

528 --- A function is used as a variable, i.e., it has no arguments.

530 --- Variable should be unsubscripted.

532 --- Variable should be subscripted 'K'.

534 --- Variable should be subscripted 'JK'.

535 --- Variable should be subscripted 'KL'.

536 --- The indicated variable is not defined in the model and cannot be defined in a RERUN.

537 --- Changing the value of the indicated variable will not affect the model during RERUN since the variable is not used in any equation.

538 --- DT is the only SPEC card parameter that may be changed during a RERUN. The indicated parameter may not be specified.

539 --- The number of elements in a table array change for a RERUN is not consistent with the number in the model.

540 --- The indicated variable may not be changed in an equation of this type.

551 --- The user may not write an equation for a reserved word.

552 --- The user may not write an equation for an integrator.

555 --- The symbol on the left side of the equal sign has already been defined as a constant.

557 --- The symbol on the left side of the equal sign has already been defined as a supplementary.

558 --- The symbol on the left side of the equal sign has already been defined as an auxiliary.

559 --- The symbol on the left side of the equal sign has already been defined as a rate.

560 --- The symbol on the left side of the equal sign has already been defined as a level.

567 --- Reserved words may not be used in this equation.

571 --- Constants may not be used in this equation.

573 --- Supplementaries may not be used in this equation.

574 --- Auxiliaries may not be used in this equation.

575 --- Rates may not be used in this equation.

576 --- Levels may not be used in this equation.

581 --- The symbol on the left side of the equal sign has already been defined in another constant equation.

582 --- The indicated level variable has been previously initialized; this equation will be ignored.

583 --- The symbol on the left side of the equal sign has already been defined in another supplementary equation.

584 --- The symbol on the left side of the equal sign has already been defined in another auxiliary equation.

585 --- The symbol on the left side of the equal sign has already been defined in another rate equation.

586 --- The symbol on the left side of the equal sign has already been defined in another level equation.

600 --- There is an illegal character in the numeric literal.

601 --- The expected numeric literal string is missing.

602 --- A '+' or a '-' sign may appear only at the start of the numeric literal or the start of the exponent. It is illegal as used where indicated.

603 --- Mantissa exceeds the 8 digit limit.

604 --- Exponent exceeds the 2 digit limit.

605 --- There is a duplicate decimal point.

606 --- Exponent may not contain a decimal point.

607 --- The number being exponentiated has a value of zero.

608 --- There is a duplicate exponent character.

609 --- An exponent character is encountered before the mantissa.

610 --- Expected mantissa is missing.

611 --- The exponent character is not followed by an exponent.

612 --- The exponent is too large or too small.

701 --- A delimiter (comma, slash, or parenthesis) is repeated. The redundant delimeters are ignored.

702 --- On this card, the first delimiter should be used throughout to separate the items in this list.

703 --- A PRINT or PLOT card should not begin or end with a comma or slash.

704 --- This PRINT or PLOT card contains no variables.

705 --- Character sequence /) or ,) was encountered. NDTRAN assumes user intended)/ or),

706 --- Two non-consecutive open parentheses were encountered; NDTRAN assumes the second to be a closed parenthesis.

707 --- There was no comma or slash after the indicated closed parenthesis.

708 --- A closed parenthesis was encountered for which there was no open parenthesis.

709 --- A second comma was encountered after an open parenthesis. A closed parenthesis is missing, and assumed to be there.

710 --- A slash is encountered after an open parenthesis; a closed parenthesis is missing.

711 --- A PRINT or PLOT card may contain no more than ten variables.

712 --- A PRINT card may not have an '='.

713 --- A PLOT character may be only one character long.

714 --- Any character may appear only once on a PLOT card.

715 --- There is no PLOT character after the indicated equal sign.
Note: NDTRAN does not accept the following as PLOT characters: '=', ',', '(', ')', and '/'.

716 --- Range for this PLOT series has been previously defined.

717 --- The narrow option allows only 5 variables on a PLOT card and 6 on a PRINT card.

801 --- There is an equal sign missing.

802 --- The table contains no elements.

803 --- A table must contain more than one element.

804 --- CP and TP statements may occur only in RERUN. This card will be processed as a C or T card.

901 --- Within an integrator, only addition and subtraction of rate variables may occur.

902 --- Arithmetic expressions may not occur outside of an integrator.

903 --- All level equations must contain an integrator.

904 --- A level equation may not contain numeric literals.

1101--- The indicated RERUN statement is not followed by a valid variable change. This RERUN will not be executed.

1102--- RUN and RERUN cards may not be continued.

APPENDIX C: Execution Time Error Messages

These messages will appear during execution of a program, and will normally halt the program execution. In each case an error message is printed out that tells three things:

1. The statement number where the error occurred;
2. The time period when the error occurred;
3. The cause of the error.

There are 6 types of execution-time errors that are detected by NDTRAN. In each instance an error message is printed out with the following form:

ERROR TYPE " in statement XXXXX at time = t. "

If your program attempted to divide , in an equation where the divisor were equal to 0 (zero) , the following message would be printed out:

DIVISION BY ZERO HAS OCCURRED IN STATEMENT XXXXX AT TIME = T.

The types of error messages that can occur (will be detected are):

1. OVERFLOW HAS OCCURRED ...
2. UNDERFLOW HAS OCCURRED ...
3. DIVISION BY ZERO HAS OCCURRED ...
4. A NEGATIVE NUMBER IS BEING RAISED TO A FRACTIONAL POWER ...
5. THE NATURAL LOG OF A NON-POSITIVE NUMBER HAS BEEN REQUESTED ...
6. THE SQUARE ROOT OF A NEGATIVE NUMBER HAS BEEN REQUESTED ...

For these six error messages a standard fixup is taken and execution continues. If more than 10 execution-time errors occur, then the run will terminate.

For NDTRAN version 1 the previous six error messages occur plus two additional messages:

7. THE HIGH BOUND OF THE INDEPENDENT VARIABLE IS LESS THA THE LOW BOUND IN A TABLE FUNCTION ...
8. THE VALUE OF THE INDEPENDENT VARIABLE IS OUTSIDE OF THE SPECIFIED RANGE IN A TABLE FUNCTION ...

These two error messages will cause the run to be terminated immediately.

Modifications of the Table function TABLE in NDTRAN2 make these two messages not required.

APPENDIX D

University of Notre Dame, Department of Economics

NDTRAN2: System Errors

1. Program is too large to compile.

2. One equation has too many operators to scan for syntax.

3. Symbol table has overflowed.

4. Numeric literal table has overflowed.

5. One equation has too many operators to compile.

6. Parenthesis nesting is too deep in an equation.

7. Insufficient disk storage is available.

8. Insufficient memory available for PRINT/PLOT output.

   Normal correction -- reduce time of Run or increase DT.

FORM OF ERROR MESSAGE.

The error prints out as follows - normally after the compile step:

SYSTEM ERROR: 8

INDEX