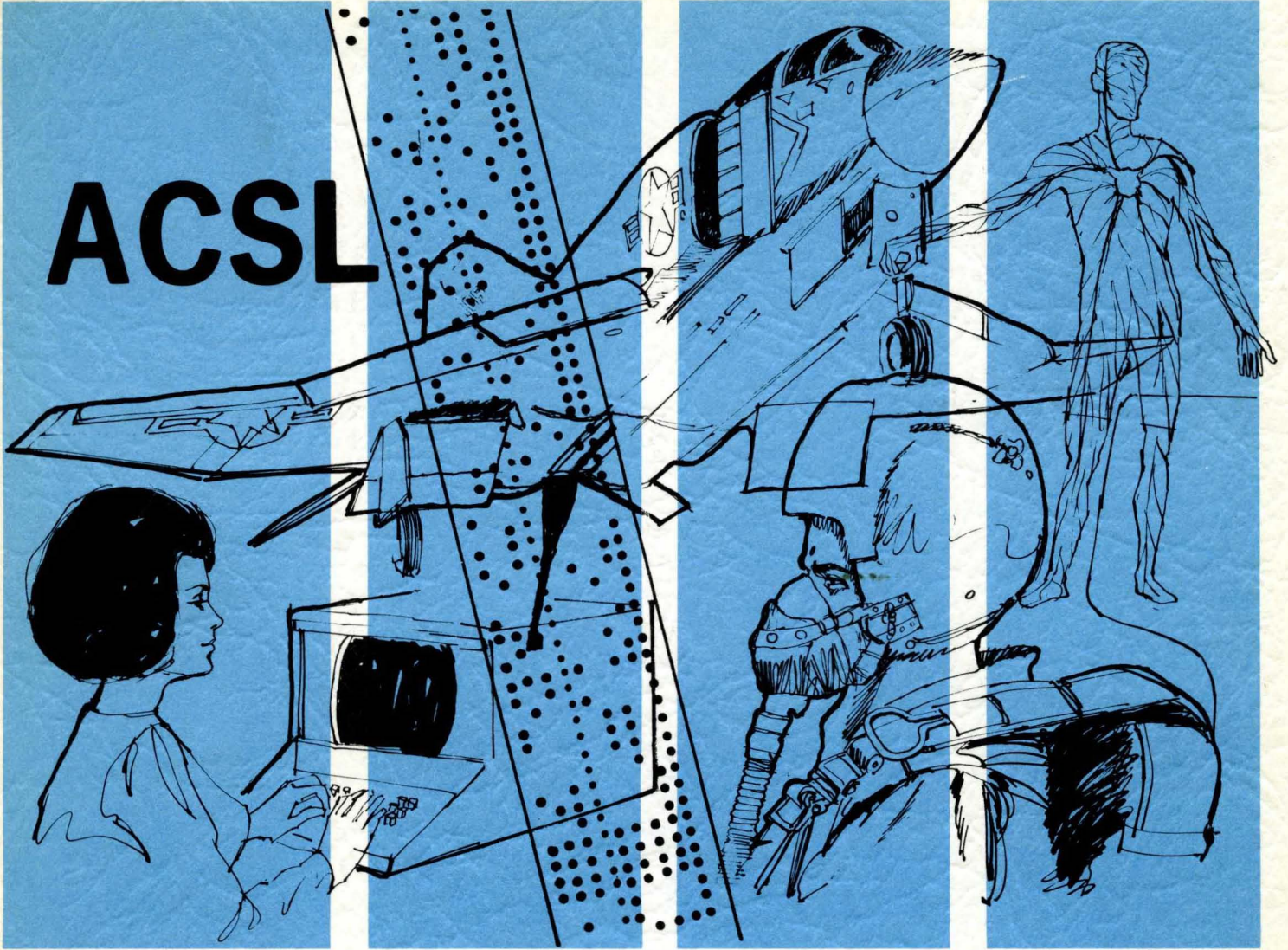


ACSL



ADVANCED CONTINUOUS SIMULATION LANGUAGE

USER GUIDE / REFERENCE MANUAL

**ADVANCED CONTINUOUS
SIMULATION LANGUAGE
(ACSL)**

USER GUIDE / REFERENCE MANUAL

Mitchell and Gauthier, Assoc., Inc.
73 Junction Square Drive
Concord, Mass. 01742

Copyr. © Mitchell and Gauthier, Associates 1981.

Printed in the United States of America

Third Edition

FOREWORD

This manual describes the operations available with the Advanced Continuous Simulation Language, ACSL (pronounced "axle"). This language was designed for modelling and evaluating the performance of continuous systems described by time dependent, non-linear differential equations.

In the development of the simulation system, emphasis has been placed on the ability to run and evaluate the model on-line. Provision has been made to overcome the usual problem of high data volume -- monitoring information can be directed to the terminal, high volume output to a local line printer.

In becoming familiar with this book, it is recommended that the reader look through Chapters 1, 2 and 3 and then turn to the example programs given in Appendix A -- making reference to the operators, Chapter 4, and run-time commands, Chapter 5, as necessary. These chapters listing the available operators are not designed for direct reading because of the alphabetic organization of the material.

Before running a simulation, make sure Chapter 7 on program debugging has been studied.

TABLE OF CONTENTS

<u>Section/Para</u>	<u>Page</u>
1. INTRODUCTION	1-1
1.1 JOB PROCESSING	1-1
1.2 LANGUAGE FEATURES	1-1
1.3 CODING PROCEDURE	1-4
1.4 RESERVED NAMES	1-4
1.5 FORTRAN SUBROUTINES	1-5
2. LANGUAGE ELEMENTS	2-1
2.1 CONSTANTS	2-1
2.2 VARIABLES	2-2
2.3 LABELS	2-3
2.4 EXPRESSIONS	2-4
3. EXPLICIT STRUCTURE	3-1
3.1 PROGRAM FLOW	3-1
3.2 PROGRAM SORTING	3-3
4. ACSL STATEMENTS	4-1
4.1 ABS	4-2
4.2 ACOS	4-3
4.3 AINT	4-3
4.4 ALGORITHM	4-3
4.5 ALOG	4-4
4.6 ALOG10	4-5
4.7 AMAXO	4-5
4.8 AMAX1	4-5
4.9 AMINO	4-5
4.10 AMIN1	4-5
4.11 AMOD	4-5
4.12 ARRAY	4-5
4.13 ASIN	4-6
4.14 ASSIGN	4-6
4.15 ASSIGNMENT STATEMENT	4-6
4.16 ATAN	4-6
4.17 ATAN2	4-7
4.18 BCKLSH	4-7
4.19 BOUND	4-8
4.20 CALL	4-8
4.21 CINTERVAL	4-8
4.22 CMPXPL	4-10
4.23 COMMENT	4-10
4.24 CONSTANT	4-11
4.25 CONTINUE	4-11
4.26 COS	4-11
4.27 DBG	4-11
4.28 DBLINT	4-12
4.29 DEAD	4-12

TABLE OF CONTENTS (Continued)

Section/Para		Page
4.30	DELAY	4-13
4.31	DERIVATIVE	4-14
4.32	DERIVT	4-16
4.33	DIM	4-17
4.34	DISCRETE	4-17
4.35	DO STATEMENT	4-19
4.35	DYNAMIC	4-19
4.36	END	4-20
4.37	EQUIVALENCE	4-20
4.38	ERRTAG	4-21
4.39	EXP	4-21
4.40	EXPF	4-21
4.41	FCNSW	4-22
4.42	FORMAT	4-22
4.43	GAUSS	4-22
4.44	GAUSI or UNIFI	4-23
4.45	GO TO STATEMENT	4-23
4.46	HARM	4-24
4.47	HYSTERESIS	4-24
4.48	IABS	4-24
4.49	IDIM	4-25
4.50	IF STATEMENTS	4-25
4.51	IMPL	4-25
4.52	INITIAL	4-27
4.53	INT	4-27
4.54	INTEG	4-27
4.55	INTEGER	4-29
4.56	INTERVAL	4-29
4.57	INTVC	4-30
4.58	I/O STATEMENTS	4-30
4.59	ISIGN	4-31
4.60	LEDLAG	4-31
4.61	LIMINT	4-32
4.62	LINES	4-32
4.63	LOG	4-33
4.64	LOGICAL	4-33
4.65	LSW, RSW	4-33
4.66	MACRO	4-34
4.67	MAX0	4-34
4.68	MAX1	4-34
4.69	MAXTERVAL	4-34
4.70	MERROR, XERROR	4-34
4.71	MIN0	4-35
4.72	MIN1	4-35

TABLE OF CONTENTS (Continued)

Section/Para	Page	
4.73	MINTERVAL, MAXTERVAL.....	4-35
4.74	MOD.....	4-36
4.75	MODINT.....	4-36
4.76	NSTEPS.....	4-37
4.77	OU.....	4-37
4.78	OUTPUT, PREPAR.....	4-39
4.79	PAGE.....	4-39
4.80	PREPAR.....	4-40
4.81	PRINT.....	4-40
4.82	PROCEDURAL.....	4-40
4.83	PROGRAM.....	4-41
4.84	PTR.....	4-41
4.85	PULSE.....	4-41
4.86	QNTZR.....	4-42
4.87	RAMP.....	4-42
4.88	READ.....	4-43
4.89	REAL.....	4-43
4.90	REALPL.....	4-43
4.91	RESET.....	4-43
4.92	RSW.....	4-44
4.93	RTP.....	4-44
4.94	SAVE.....	4-45
4.95	SCALE.....	4-45
4.96	SIGN.....	4-45
4.97	SIN.....	4-46
4.98	SQRT.....	4-46
4.99	STEP.....	4-46
4.100	TABLE.....	4-46
4.101	TAN.....	4-48
4.102	TERMINAL.....	4-48
4.103	TERMT.....	4-48
4.104	TRAN.....	4-49
4.105	TYPE.....	4-49
4.106	UNIF.....	4-51
4.107	UNIFI.....	4-51
4.108	VARIABLE.....	4-51
4.109	WRITE.....	4-51
4.110	XERROR.....	4-51
4.111	ZHOLD.....	4-51
4.112	ZOH.....	4-52
5.	ACSL RUN-TIME COMMANDS.....	5-1
5.1	ACTION.....	5-2
5.2	ANALYZ.....	5-2
5.3	COMMENT.....	5-5

TABLE OF CONTENTS (Continued)

Section/Para	Page
5.4 CONTIN	5-5
5.5 DISPLY (sic): Short Form D	5-5
5.6 END	5-5
5.7 MERROR, XERROR	5-6
5.8 OUTPUT	5-6
5.9 PLOT	5-7
5.10 PREPAR	5-9
5.11 PRINT	5-9
5.12 PROCED	5-10
5.13 RANGE	5-10
5.14 REINIT	5-11
5.15 RESTOR	5-11
5.16 SAVE	5-11
5.17 SET: Short Form S	5-11
5.18 SPARE	5-12
5.19 START	5-12
5.20 STOP	5-12
5.21 XERROR	5-12
6. MACRO LANGUAGE	6-1
6.1 MACRO DEFINITIONS	6-1
6.2 MACRO DEFINITION HEADER	6-2
6.3 MACRO DIRECTIVE STATEMENTS	6-3
6.4 MACRO EXAMPLES	6-7
6.5 MACRO CALLS	6-10
7. PROGRAM DEBUGGING	7-1
7.1 MEANING OF DEBUG PRINT OUT	7-2
8. APPLICATION NOTES	8-1
8.1 PARAMETER SWEEP	8-1
8.2 PHASE AND GAIN PLOTS	8-2
8.3 SUMMARY OUTPUT	8-3
8.4 IMPULSE AND STEP RESPONSE	8-3
8.5 EXTERNALLY DEFINED VARIABLES	8-5
APPENDIX A	A-1
1. LIMIT CYCLE	A-1
2. SPRING	A-7
3. CONTROL LOOP	A-7
4. PILOT EJECTION STUDY	A-19
5. TEMPERATURE DISTRIBUTION ALONG A RADIATING FIN	A-28
6. AIRCRAFT ARRESTING GEAR SYSTEM	A-41
7. LONGITUDINAL STUDY	A-51
8. PHYSBE	A-62
9. PHASE AND GAIN	A-76
10. MISSILE AIRFRAME MODEL	A-84
11. DISCRETE SAMPLED COMPENSATOR	A-104
12. ASPIRIN DOSAGE EVALUATION	A-112

TABLE OF CONTENTS (Continued)

Section/Para	Page
APPENDIX B	B-1
1. AGET (name, a), APUT (name, a)	B-1
2. BLDDCT (nHname, name, type, size)	B-1
3. DEBUG	B-2
4. IGET (nHname,i) RGET(nHname,i)	B-3
5. INTEG.	B-3
6. LISTD (file)	B-3
7. LOG.	B-4
8. RGET (nHname, i)	B-4
9. SET (value, name, times)	B-4
10. TIMER	B-4
11. VPUT (name, i, value)	B-4
12. WEM (nH message, nchar)	B-4
13. WRITG	B-5
14. XFERB	B-5
APPENDIX C	C-1
1. Refers to Plots in General	C-1
2. Refers to Printer Plots	C-2
3. Refers to Line or Calcomp Plots	C-2
4. Refers to Strip Plots	C-4
5. Refers to Print Data	C-4
6. Integration Control	C-5
7. General	C-5
APPENDIX D	D-1
APPENDIX E	E-1
1. Batch Operation	E-1
2. Terminal Operation	E-2
APPENDIX F	F-1
1. ACSL Translator Error Messages	F-1
2. Run-Time Error Messages	F-3
INDEX	G-1

THIS PAGE INTENTIONALLY LEFT BLANK

1. INTRODUCTION

Simulation of physical systems is a standard analysis tool used in the evaluation of hardware design prior to actual construction. The continuous system simulation language described herein, and referred to as ACSL - Advanced Continuous Simulation Language - has been developed expressly for the purpose of modelling systems described by time dependent, non-linear differential equations and/or transfer functions.* Typical application areas are control system design, chemical process representation, missile and aircraft simulation or fluid flow and heat transfer analysis. Program preparation can either be from block diagram interconnection, conventional FORTRAN statements or a mixture of both.

Highlights of the language are free form input, function generation of up to three variables, and independent error control on each integrator. Flexibility is provided for plotting the behavior of the models under a number of external forcing functions. Many simulation oriented operators such as variable time delay, dead zone, backlash and quantization are included and made readily accessible.

The ACSL program was intended to provide a simple method of representing these mathematical models on a digital computer. Working from an equation description of the problem or a block diagram, the user writes ACSL statements to describe the system under investigation. Drive cards are written to exercise the model and these statements are keypunched and submitted for solution. An alternate mode is via a remote terminal such as a teletype; true on-line control is now possible with the user changing model constants and monitoring the resulting solutions.

Statements describing the model need not be ordered since the ACSL processor will sort the equations so that no values are used before they have been calculated (applies only to the DERIVATIVE section of an explicitly structured program - see subsection 3.0). This operation of the language is to be contrasted to the usual digital programming languages like FORTRAN, where program execution depends critically on statement order. The ACSL sorting procedure is described in more detail in Subsection 3.2.

1.1 JOB PROCESSING

The cards supplied by the user are in two distinct groups - the first group contains those concerned with defining the model or structure of the system being simulated; the second group contains the sequence of commands that exercise this model - i.e., change parameters, start runs, control which plots to make.

The model definition statements are read by the ACSL program which translates these to a set of intermediate FORTRAN programs. These FORTRAN programs are compiled, loaded with the ACSL Run-time Library and executed. The run-time drive cards now are read by the executive which decodes and executes commands in sequence - refer to Section 5 for a description of the commands usable at run-time. At least a START command is necessary at this point to exercise the model.

1.2 LANGUAGE FEATURES

The language consists of a set of arithmetic operators, standard functions and a set of special statements and a MACRO capability which allows extension of the special statements, either at the system level for each installation, or for each individual user. The arithmetic operators are +, -, *, /, **, denoting addition, subtraction, multiplication, division and exponentiation. The functions are listed in Section 4 and consist of special ACSL operators such as QNTZR (quantization), UNIF (uniform random number), etc. In addition, all the functions of the FORTRAN library are available such as SQRT (square root), AMOD (modulus), etc. Integration is a special ACSL operator that is accomplished either by INTEG or INTVC, i.e.

* The basic structure follows the specification established by the Technical Committee on Continuous System Simulation Languages and under the auspices of Simulation Councils, Inc. (SCi) in SIMULATION 9 (Dec. 1967) pp 281-303.

$$R = \text{INTEG}(X, \text{RZERO})$$

implies

$$R = \text{RZERO} + \int_0^T X \, dt$$

This integration operator is the heart of the simulation system. In building any model, it is necessary to change differential operators into integration operators and this is conventionally accomplished by expressing the highest derivative of a variable in terms of lower derivatives and other variables. As an example, consider the spring dashpot system excited by a given function of time $F(t)$. In general form it can be written

$$\ddot{x} + 2\zeta \omega \dot{x} + \omega^2 x = F(t)$$

where ω is the natural frequency and ζ the damping ratio. Expressing this equation in terms of the highest derivative \ddot{x} , we have

$$\ddot{x} = F(t) - 2\zeta \omega \dot{x} - \omega^2 x$$

Since \dot{x} appears on the right hand side, we must give it a name (XD). The two integrations can be written as

$$\text{XD} = \text{INTEG}(F(T) - 2*\text{ZE}*W*\text{XD} - W**2*X, \text{XDIC})$$

$$X = \text{INTEG}(\text{XD}, \text{XIC})$$

This process transforms the original set of differential equations to a set of first order equations which can be solved directly by integrating.

It is necessary to be somewhat careful in the transformation process to avoid the introduction of redundant state variables. In the above sequence we could have avoided the reference to \dot{x} (XD) by calculating \ddot{x} (XDD) directly and embedding an INTEG operator in the expression, i.e.

$$\text{XDD} = F(T) - 2*\text{ZE}*W*\text{INTEG}(\text{XDD}, \text{XDIC}) - W**2*X$$

Now X is the second integral of XDD so

$$X = \text{INTEG}(\text{INTEG}(\text{XDD}, \text{XDIC}), \text{XIC})$$

In these two equations, there are three INTEG operators - each one a state variable - but two are integrations of XDD with the same initial condition, one of these is redundant and can be eliminated by explicitly naming the first derivative as shown previously.

Before we give a detailed description of the language, let us code a simple problem using the arithmetic operators and the SQRT and INTEG functions defined above.

The coding for the equations

$$\dot{X} = Y + X(1 - X^2 - Y^2)/\sqrt{X^2 + Y^2}; X(0) = 0.5$$

$$\dot{Y} = -X + Y(1 - X^2 - Y^2)/\sqrt{X^2 + Y^2}; Y(0) = 0.0$$

is shown in Figure 1-1.

While this series of statements completely describes the equations to be solved it does not represent the complete problem statement. Figure 1-2 lists the complete running program where each card is numbered for reference. To complete the model definition a PROGRAM card (1) is needed to start; the communication interval must be defined by the CINTERVAL operator (3); the termination condition must be specified as a logical expression forming the argument for the TERMT operator (7); and the model definition must be completed by an END card (8) which matches the PROGRAM card. This end card tells the translator nothing further is expected unless FORTRAN subroutines or functions are included. A further refinement is in assigning symbolic names for the initial condition, $X(0)$ and $Y(0)$, and for the stop time TSTOP. These are preset in the CONSTANT statement (2).

$$\begin{aligned}
 R2 &= X^{**2} + Y^{**2} \\
 X &= \text{INTEG}(Y + X*(1.0 - R2)/\text{SQRT}(R2), 0.5) \\
 Y &= \text{INTEG}(-X + Y*(1.0 - R2)/\text{SQRT}(R2), 0.0)
 \end{aligned}$$

Figure 1-1. Model Equations Corresponding to Mathematical Definition - Limit Cycle Problem

MODEL DEFINITION CARDS	
(1)	PROGRAM LIMIT CYCLE
(2)	CONSTANT XZ = 0.5, YZ = 0.0, TSTOP = 4.0
(3)	CINTERVAL CINT = 0.1
(4)	R2 = X**2 + Y**2
(5)	X = INTEG(Y + X*(1.0 - R2)/SQRT(R2), XZ)
(6)	Y = INTEG(-X + Y*(1.0 - R2)/SQRT(R2), YZ)
(7)	TERMT(T.GE.TSTOP)
(8)	END
RUN-TIME DRIVE CARDS	
(9)	OUTPUT T, X, Y
(10)	START
(11)	'CHANGE INITIAL CONDITIONS AND RUN AGAIN'
(12)	SET XZ = 0.7
(13)	START
(14)	STOP

Figure 1-2. Programs for Model Definition and Run-time Drive Cards - Limit Cycle Problem

Once a symbol is assigned to a variable it can be changed at run-time. **Embedding** literal constants (the characters 0.5 are a literal constant) in the program is not good programming practice. It is much better to use a symbol preset to the value desired. Changes then occur in one place.

In addition to the model definition statements, the run-time group of cards must be defined that tells how the model is to be exercised. These are separated from the model definition section by an end-of-record separator for a batch run (7-8-9 multipunch in column one on a CDC 6000/7000 machine) or else a separate file. Interactively the run time commands can come from a key board or a terminal. The OUTPUT statement (9) defines the variables which are to be listed at each communication interval; then the model must be told to go by the command START (10). Card number eleven is a comment, twelve changes the initial condition and thirteen says run again. Card number fourteen terminates the simulation study. Section 1 of Appendix A lists the actual program and the output listing and plots obtained in solving this set of equations.

1.3 CODING PROCEDURE

The ACSL coding line contains eighty characters of which the first seventy-two are used for program information and 73-80 for identification, if needed. The ACSL statement may be placed anywhere on the card and the user should arrive at some standards to preserve a uniform appearance of the source text. Suggested standards are as follows:

Section delineators - PROGRAM, INITIAL, DYNAMIC, END, etc., starting in column 1

PROCEDURAL brackets - matching pairs of PROCEDURAL/END statements starting in column 6

Unlabelled statements - starting in column 7

Labelled statements - label starting in column 6

Assignment statements - ensure the equal sign (=) falls in column 15 or beyond

These procedures are not necessary, but they make the resulting program much neater and easier to read.

More than one statement can be placed on one card by separating them by a dollar sign (\$). Any statement can be continued on to the next line by adding an ellipsis (three consecutive periods . . .) anywhere at the end of the card - to the right of any non-blank information but before column seventy-two. Any trailing blanks are squeezed out of the card containing the ellipsis and the following card is added directly - it is as though the characters were strung on the end after the last non-blank character of the preceding card, starting with column 1. Leading blanks of the continuation card are not suppressed. Comments may be added by enclosing a complete statement in quotes. Note that neither a quote (') nor a dollar sign (\$) can be part of a comment. Examples:

```
'THIS IS A COMMENT CONTINUED ON TO . . .
```

```
THE NEXT LINE'
```

```
'THE NEXT LINE HAS TWO STATEMENTS'
```

```
A = B + C    $    X = Y + Z
```

```
'COMMENTS CAN BE ADDED IN LINE AS . . .
```

```
A SEPARATE STATEMENT'
```

```
X = Y + Z    $    'ASSIGN VALUE TO X'
```

```
'NEXT ARE TWO LABELLED STATEMENTS'
```

```
LABEL . . C = D + 50
```

```
1000 . . FORMAT(1X, 3E12.5)
```

Blank lines can be interspersed throughout the code in order to help produce an attractive program format.

1.4 RESERVED NAMES

The only reserved names in the language are those of the form Z0nnnn and ZZaaaa, where 'n' is any digit and 'a' is any alphanumeric character. All generated variables and system subroutine names are in this form. However, it behooves you, the user, to keep in mind the system variable default names listed in Table 1-1. These default names can be changed to any name you please, but if you do not, the default names will be in existence so they can be set by program control in the model definition section. Other uses of these names, e.g., defining MAXT as a state variable, could cause conflicts.

TABLE 1-1. System Variable Default Names

Set By Statement	Default Name	Default Value	Definition
CINTERVAL	CINT	0.1	Communication Interval
MINTERVAL	MINT	1.0E-10	Minimum Calculation Interval
MAXTERVAL	MAXT	1.0E+10	Maximum Calculation Interval
NSTEPS	NSTP	10	Number of Calculation Intervals
VARIABLE	T	0.0	Independent Variable
ALGORITHM	IALG	5	Integration Algorithm
ERRTAG	none	.FALSE.	Error Indicator

NOTE: It is probably a good idea to treat these names as reserved and not use them for any purpose except that stated.

A second level of names is the run-time executive system constants listed in Appendix C. These are flags and numeric values that determine the manner in which plots are going to be made, print intervals, etc.

Such quantities as the y-axis length in inches (YINCPL) for line plots, number of points between grid lines in the x-direction for printer plots (NGXPPL) can be under user's control - at run-time - if desired. All these symbols have a default value that is chosen for reasonableness. Access is by normal SET commands, i.e.,

SET YINCPL = 5.0

Any symbol in the SET command causes a search of the user's dictionary - established by the symbols in the model definition code - and if that is unsuccessful, the dictionary of run-time system symbols is searched. Thus, if you use one of the system constants as a name in the program, that use will override the internal use within the run-time executive. It merely means that you will not have the luxury of modifying the system default value.

1.5 FORTRAN SUBROUTINES

Any user defined FORTRAN function or subroutine may be used in the simulation program by including it following the END card that matches the PROGRAM card. The translator looks for the following:

```

REAL FUNCTION      ...
INTEGER FUNCTION   ...
LOGICAL FUNCTION   ...
SUBROUTINE         ...
FUNCTION           ...
PROGRAM            ...
    
```

and if found, assumes all the rest of the cards in the model definition section are to be passed directly to the FORTRAN compiler. The format of these cards is not changed in any way so FORTRAN card format must be followed exactly, i.e. start in column seven or beyond etc.

The only change the translator makes is to look for a dollar sign (\$) in column one. This card is replaced by the simulation common block and type statements - which may run to many cards - so making available to the subroutine the names and values accessible in the main program.

WARNING: There is no selectivity in this process - either all the symbols are obtained or none. It is easy to get name conflicts and also violate the translator sorting algorithm by this action.

2. LANGUAGE ELEMENTS

The following list of language elements must be understood before proceeding to the next section which describes the individual ACSL statements in detail. Most of the basic elements are defined the same as in FORTRAN with the exception of the list in Table 2-1, which lists the major differences between ACSL and FORTRAN. A reader who is familiar with FORTRAN could continue to the next section at this point.

TABLE 2-1. Major Differences, ACSL to FORTRAN

LABELS	Symbolic as well as numeric labels allowed. Labels are separated from statements to which they are attached by two periods. Due to problems with MACRO expansions within labelled statements, it is recommended that labels only be used with CONTINUE statements. See LABELS section 2-3 for further information.
NAMES	No embedded blanks are allowed in names. Each name must have no more than <i>SIX</i> characters. Names should not be of form Znnnnn or ZZaaaa where n is any digit and a is any alphanumeric character.
TYPES	Variables starting with I, J, K, L, M or N are <i>not</i> automatically considered integer. All variables and functions are considered real, floating point variables unless typed explicitly otherwise.
CODING	Free format - use any columns 1 through 72.
CONTINUATIONS	An ellipsis (. . .) at the end of a line implies continuation onto the next card image. A non-blank column six has no significance in free format.
COMMENTS	A fully quoted statement is a comment and cannot contain another quote or dollar sign. A 'C' in column one has no significance in free format.

2.1 CONSTANTS

Constants may be either integer, real, logical or Hollerith. An integer constant is a whole number, written without a decimal point or embedded blanks. Positive numbers may be prefixed with a plus sign; negative numbers must be prefixed with a minus sign. Length is implementation dependent (CDC 6600, about thirteen digits). Subscripts of arrays are limited to about five digits.

Examples: 0 +526 -63 476

A real constant is written with a decimal point together with an optional exponent. Positive numbers may be prefixed with a plus sign; negative numbers must be prefixed with a minus sign.

Examples: 3.E1 3.1416 31.416E-1
 -0.000345 0.347E03 -27.6E+220

Logical constants may be one of two values:

.TRUE. .FALSE.

Hollerith constants are strings of letters and digits and are defined differently for the translator and at run-time. The translator accepts the normal character count - an unsigned integer - followed by H, followed by the character string. The general form is

$$nHh_1h_2 \dots h_n$$

Examples of this are

3HSAM	12HCONTROL DATA
1H)	5H LEAD

Note that blanks are significant in the character count. For the run-time executive, Hollerith strings are defined by enclosing in quotes - so implying that the quote character cannot be included in the Hollerith constants, i.e.,

SET TITLE = 'LIMIT CYCLE, RUN1'

While literal constants can be used in expressions directly, there is no way of changing the value at run-time. The original source cards must be changed and the program retranslated, compiled and executed. If the same constant appears more than once, it implies more changes in the source text. Better programming practice is to give all constants symbolic names to be used in expressions, presetting these via CONSTANT statements; i.e.,

don't say

AREA = 3.142*R**2

Instead:

CONSTANT PI = 3.142

AREA = PI*R**2

2.2 VARIABLES

Variable names are symbolic representations of numerical quantities that may or may not change during a simulation run. They refer to a location and have a value equal to the current number stored in that location. Both simple and subscripted variables may be used. A symbolic name must start with a letter and be followed by zero to five letters and digits (not more than *SIX* characters). All variables in an ACSL program are assumed to be of type REAL unless explicitly typed otherwise. The FORTRAN convention for integers starting with the letters I through N does not apply.

Examples of simple variables:

A A57 AB57D K20

A subscripted variable may have one, two or three subscripts enclosed in parentheses. Subscripts can be expressions in which the operands are simple integer variables and integer constants, and the operators are addition, subtraction, multiplication and division only. For more information on storage allocation see ARRAY statement - Section 4.12.

Examples of subscripted variables names are:

B(5, 2)	B53(5*I + 2, 6, 7*K + 3)
C47(3)	ARRAY(2)

In the above example I and K must be declared explicitly to be INTEGER variables.

2.3 LABELS

A label may be attached to a statement so that control may be transferred to it by GO TO or used to label a FORMAT statement. Labels may be either:

- 1) One through six alphanumeric characters with the first alphabetic
- 2) One through five numeric characters (digits)*

The labels must be set off from the statement to which they apply by two periods - i.e.,

```
L1 . . X = A + B
1000 . . Y = C + D
GO TO L1
```

Although it is legal to attach a statement label to any executable statement, it is recommended that these labels only appear on CONTINUE statements. Aside from the improvement in appearance, structure and modifiability, many problems can be avoided. The basic difficulty is that the statement body may contain MACRO calls. The effect of a MACRO call inside a statement - functional form - is that the macro is expanded first *followed* by the labelled statement. As an example, consider the random number generator, GAUSS which is implemented as a MACRO. It's use could be in a PROCEDURAL in a labelled statement

```
GO TO LABEL
```

```
...
```

```
LABEL . . X = GAUSS(MEAN, SIGMA)
```

This would expand to the FORTRAN code

```
GO TO 99996
```

```
...
```

```
Z09999 = MEAN + GRV(ZZSEED)*(SIGMA)
```

```
99996 X = Z09999
```

Now the intermediate variable Z09999 won't be assigned a value if the GO TO transition is followed. However, if we use a CONTINUE instead

```
GO TO LABEL
```

```
...
```

```
LABEL . . CONTINUE
```

```
X = GAUSS(MEAN, SIGMA)
```

This code expands to

```
GO TO 99996
```

```
...
```

```
99996 CONTINUE
```

```
Z09999 = MEAN + GRV(ZZSEED)*(SIGMA)
```

```
X = Z09999
```

which gives the correct sequence. The stand alone form of the MACRO works in this case and avoids the extra assignment statement, since no intermediate variable is generated, i.e.

* The system uses labels starting at 99999 and works downwards. Labels with too high a value should not be used, else a conflict may result.

LABEL .. GAUSS(X = MEAN, SIGMA)

becomes*

99996 X = MEAN + GRV(ZZSEED)*(SIGMA)

To avoid the label problem, we had originally thought of attaching the LABEL to the first line of the code generated from the MACRO but then the problem with the DO loop raises it's head - now the label must be the last operation of the loop - statements generated after the labelled statement are not included in the loop. Even if we differentiate between the DO-loop labels, the language allows a loop with direct GO TO's to the terminating statement label:

DO LABEL I = 1, 20

...

IF (condition) GO TO LABEL

...

LABEL .. X = X + GAUSS(MEAN, SIGMA)

To formulate rules to translate this sequence appears to be impossible. If the operation is programmed with labelled CONTINUE statements, two labels will be necessary and the translation is straightforward, i.e.

DO LABEL1 I = 1, 20

...

IF (condition) GO TO LABEL2

...

LABEL2 .. CONTINUE

X = X + GAUSS(MEAN, SIGMA)

LABEL1 .. CONTINUE

2.4 EXPRESSIONS

An expression is a combination of operators and operands which, when evaluated, produces a *SINGLE* numerical value.

The arithmetic operators are

+ addition	* multiplication
- subtraction	/ division
	** exponentiation

The relational operators are

.EQ. equal to	.NE. not equal to
.GT. greater	.GE. greater than or equal to
.LT. less than	.LE. less than or equal to

(The results of a relational expression can only be .TRUE. or .FALSE.)

The logical operators are

* To use this form, you must be familiar with the implementation of each of the operators defined in the language - i.e. whether MACRO or function subroutine. It only works for single line MACROS if at the end of a DO-loop.

.NOT. reverses truth of logical expression that follows it

.AND. combines two logical expressions to give value .TRUE. when BOTH are .TRUE. otherwise .FALSE.

.OR. combines two logical expressions to give value .TRUE. when EITHER is .TRUE. otherwise .FALSE.

The operands may be either constants, variables (simple or subscripted) or functions. Examples of arithmetic expressions:

A	5.76
3 + 16.5	B + A(5)/2.75
-(C + DEL*AERO)	(B-SQRT(A**2 + X**2))/2.0

Two arithmetic operators may not appear next to each other in an arithmetic expression. If minus is to be used to indicate a negative operand, the sign and the element must be enclosed in parentheses is preceded by an operator -

i.e.,

$B*A/(-C)$ *not* $B*A/-C$
 $A*(-C)$

but,

$-A*B-C$

Parentheses may be used to indicate groupings as in ordinary mathematical notation but they may not be used to indicate multiplication.

Relational expressions are a combination of two arithmetic expressions with a relational operator. The relational expression will have the value .TRUE. or .FALSE. depending on whether the stated relation is valid or not. The general form of the relational expression is:

$a_1 \text{ op } a_2$

where the a's are arithmetic expressions and op is a relational operator.

NOTE: A relational operator must have two operands combined with one operator. Thus the form $a_1 \text{ op } a_2 \text{ op } a_3$ (A. EQ. B. EQ. C) is INVALID.

Examples are:

A. EQ. B	A + D .LT. 5.3
A57. GT. 0.0	(5.0*B - 3.0).LE.(4.0 - C)

Logical expressions are combinations of logical operands and/or logical operators which, when evaluated, will have a value of .TRUE. or .FALSE.

$l_1 \text{ op } l_2 \text{ op } l_3 \text{ op } l_4 \dots$

where the l's are logical operands or relational expressions. i.e.,

LOGICAL AA, CC, LFUNC

AA. AND. (B. GE. C). OR. CC. AND. LFUNC(X,Y). AND. (X. LE. Y). OR.. NOT. AA

Note that the symbolic names AA, CC and LFUNC have been declared to be of type logical. LFUNC is a function with a .TRUE. or .FALSE. result calculated from the value of the variables X and Y.

Note that arguments of functions may, in general, be expressions. Since expressions can contain functions, an arbitrary depth of complexity can be generated. Just using the SIN and COS function as an example, a valid expression would be

$$A + \sin(X \cdot \cos(5 \cdot X + Y)) - \cos(A + Z / \sin(5.3 \cdot C) + C \cdot Y / \sin(\sin(X + Y) \cdot \pi))$$

Remember the sole requirement for an expression is that it has ONE value when evaluated numerically.

3. EXPLICIT STRUCTURE

The basic program structure where the model is surrounded by PROGRAM...END cards has limitations when it comes to calculating initial conditions. They must be real variables, expressions or constants and cannot incorporate multiple statement logic. In order to have a more flexible structure, three extra regions of the program may be introduced - INITIAL, DYNAMIC and TERMINAL - and the model definition is placed in a DERIVATIVE section embedded in the DYNAMIC. Each section must be terminated with an END card as shown in Figure 3-1. If this method is chosen, any or all of these four explicit blocks may be included: Each block must have its own terminating END card and the ordering INITIAL, DYNAMIC, embedded DERIVATIVE and TERMINAL must be followed.

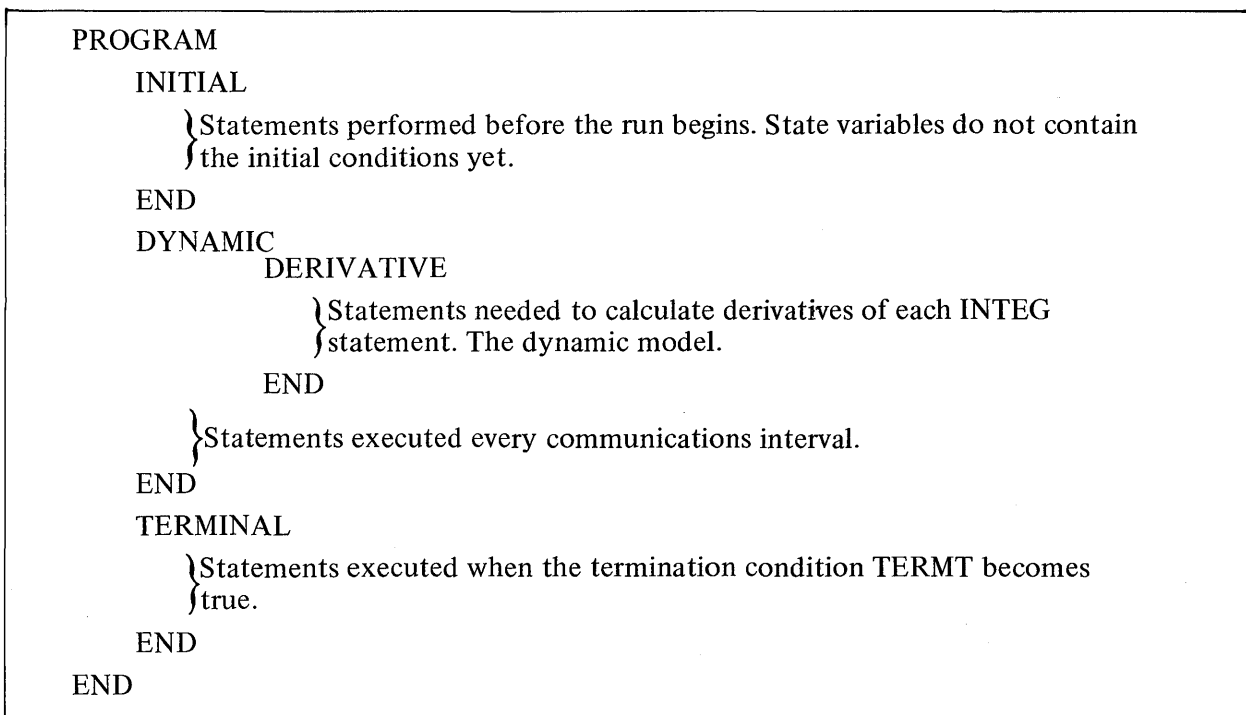


Figure 3-1. Outline of Explicitly Structured Program

3.1 PROGRAM FLOW

The program flow when leaving the executive (due to the START command) is to sequentially proceed through the INITIAL section (Figure 3-2 shows a flow diagram). At this point, the initial conditions have not been transferred into the state variables (outputs of integrators) so these variables will be undefined if any attempt is made to use them. It should be possible to arrange the calculation of any necessary initial conditions in terms of any constants or other initial conditions already defined. Another alternative is to use the RESET (q.v.) operator that will transfer the available initial conditions to the state names and may (RESET('EVAL')) or may not (RESET('NOEVAL')) be accompanied by a call to the derivative evaluation routine to calculate all intermediate variables.

Leaving the INITIAL region, the integration routine is initialized which involves transferring all initial conditions into the corresponding states and evaluating the code in the DERIVATIVE section once. This

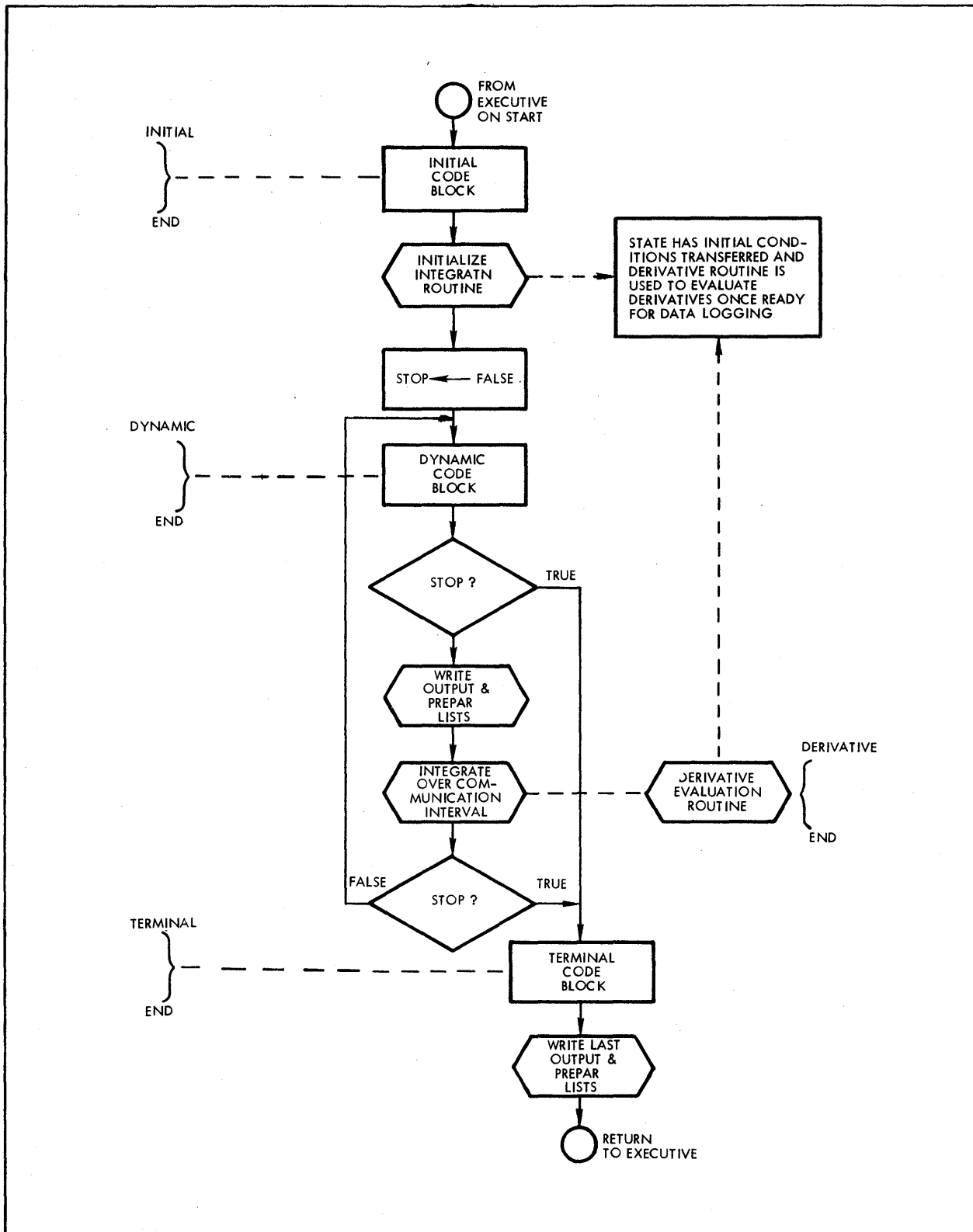


Figure 3-2. Main Program Loop of ACSL Model

action is taken to ensure that all calculated variables are known before the data logging at the time equals zero condition. Note that it may appear from the program listing that the DYNAMIC section is executed before the DERIVATIVE section; in actual fact, this is not so. You should not rely on calculations in the DYNAMIC section to initialize variables for this DERIVATIVE section. On the other hand, all values calculated in the DERIVATIVE section are available in the DYNAMIC section.

Still following the flow chart in Figure 3-2, the STOP flag is reset and the program starts to execute the code within the DYNAMIC block.

There is no restriction on the variables that may be referred to in this DYNAMIC section. All the states will have values and intermediate calculations in the DERIVATIVE section will have been executed.

After this block the STOP flag is tested to see if it is time to transfer to the TERMINAL region. Note, the STOP flag is set by the TERMT statement and if any of these statements are included in the DYNAMIC block, exit will occur at this check when one of the arguments becomes true.

If the STOP flag is not set, the program writes out the values of all the variables specified in the OUTPUT and PREPAR lists - the former to the output file or printer, the latter to a scratch file for later plotting.

The integration routine is now asked to integrate over a communication interval using the code embedded in the DERIVATIVE block to evaluate the state variable derivatives.

The integration routine returns with the states advanced through the communication interval and again the STOP flag is tested. At this point, it may have been set by a TERMT statement placed in the DERIVATIVE routine. If not set, the program loops and reexecutes the DYNAMIC section.

Control can be transferred between sections using GO TO's and statement labels, if needed. It is illegal to transfer into the DYNAMIC region, since the integration initialization won't be performed correctly. Transfer *from* the dynamic region to either INITIAL or TERMINAL is quite acceptable, so also, is transfer between INITIAL and TERMINAL blocks. Note at least one GO TO in the DYNAMIC region or one TERMT statement must be included or the program will never stop!

Control cannot be transferred either into or out of the DERIVATIVE section since this is changed into a separate subroutine and as such, is inaccessible to the main program loop.

When the STOP flag is set true, program control transfers to the TERMINAL section which is executed in sequence. On passing out of the section END, control returns to the executive which will read and process another control card (run-time command - PLOT, DISPLY, etc.). Note, if a jump (GO TO) is included in the TERMINAL section back to the INITIAL, the last output will not be written out, unless the LOG operator is used (see Chapter 4).

3.2 PROGRAM SORTING

The model definition code that is placed in the DERIVATIVE subroutine is sorted so that outputs are calculated before they are used. The sort algorithm is relatively simple and consists of two passes. Pass number one examines each statement; output variables are marked as calculated and an input list is established for the statement - all the variables on the right of the equals sign. A variable name may appear simultaneously on both left and right hand sides of an equal sign (=) in either an assignment statement or PROCEDURAL header. In this case sorting takes place only on the left hand or output variable so that the block is positioned before any use of the variable. Pass number two takes the list of statements and examines them in turn. If none of the variables on the input list have their calculated flags set, the statement is added to the compile file and the calculated flag for the output variable (or variables) is turned off.

If any of the variables on the input list are marked calculated, the statement is saved and the next statement examined. If any statement has been added to the compile file, all the saved statements are

reexamined to see if they can now be disposed of and their calculated flags reset. This algorithm works because any output variables that have already been processed will have their calculated flags reset. Only output variables coming up in the statement stream will be flagged and as such can hold up statements that depend on these outputs. State variables, of course, are not flagged as calculated - the calculation must be performed by the integration routine and all derivative evaluations finally derive from the current value of the state variables. Example:

CONSTANT PI = 3.142, RZ = 1.0

AREA = PI*R**2

R = RZ + LR

LR = INTEG(AREA, 0.0)

The first statement is translated into the FORTRAN DATA statement and an I/O list established. There are no inputs and the variables PI and RZ have their calculated flags set.

DATA PI/3.142/, RZ/1.0/

The second statement has inputs PI and R, and variable AREA has its calculated flag set. Third statement has an inputs, RZ and LR; the calculated flag for R is set. The last statement establishes LR as a state and AREA as the derivative. The calculated flag for LR is not set - it is assumed known at the start.

At the end of the first pass the symbol table looks like Table 3-1, and the calculated flags are set as in the (a) column.

TABLE 3-1. Calculated Flags in Symbol Table During Sort

Symbol	(a) Begin Pass 2	(b) After Constant	(c) After R = Statement	(d) After AREA = Statement
AREA	ON	ON	ON	
LR				
PI	ON			
R	ON	ON		
RZ	ON			

The second pass starts by reading the statements in sequence again, along with the associated I/O lists. The first statement has no input list, so it can be output to the compile file directly - in so doing, the calculated flags for PI and RZ are reset - Column (b), Table 3-1.

No other statements are pending so the next statement is read. Inputs are PI and R. R is still flagged as calculated so the statement is saved.

The next statement is read. Inputs are RZ and LR - neither of these are flagged so that statement is transferred to the compile file and the output variable flag for R is reset. The saved statement is reexamined and now, no input variables are flagged so the statement can be disposed of, erasing the flag on AREA.

Any well posed problem (without algebraic loops) can be completely disposed of by this method and the compile file will have all the statements correctly ordered.

CONSTANT statements are treated like regular assignment statements as far as sorting so if you place all the CONSTANT statements at the end of the program almost all the other statements will have to be saved

or held up. It's usually considered good practice to place these at the front of the modules in which they are used.

Algebraic loops are identified inside the ACSL translator by the fact that statements are left over at the end of the DERIVATIVE section sort. This is normally considered an error since true algebraic loops should be broken with the IMPL statement. In this case the remaining statements are listed by chaining through the loop, listing each statement in turn. It is usually easiest to follow the loop backwards from the bottom of the list where it will be found that each variable on the left side of an equals (=) sign will appear on the right hand side of the statement above it. Procedural blocks make following the path more difficult since the entire block is listed and considered as one statement but the position is determined by the variables on the left and right hand side of the PROCEDURAL header. When the code is reconstructed, the PROCEDURAL header has been lost so the dependencies should be marked in to replace the entire code block.

Most algebraic loops are programming errors caused by incorrect PROCEDURAL headers or missing state equations. For example, the PROCEDURAL block:

```
PROCEDURAL (A, B = C, D)
  A = C
  B = D
END $ 'END OF PROCEDURAL'
```

may cause an algebraic loop by implying that A is a function of D and B is a function of C. Use PROCEDURALS sparingly and with care. Using many small PROCEDURALS is better than using a few large ones.

Algebraic loops can be broken by algebra, the IMPL operator or by use of a PROCEDURAL . . . END block to hide the actual computational order from the sorter so that the previous value of the implicit variable can be used (needs initializing).

THIS PAGE INTENTIONALLY LEFT BLANK

4. ACSL STATEMENTS

Included in this section are all the basic statements which can be recognized by the ACSL translator. A number of these are the same as equivalent FORTRAN statements with the exception that there is no restriction on which card column to start in.

As a general philosophy, ACSL uses the equals sign in an unfamiliar way. In translating a user's program, ACSL needs to know what are the inputs to each statement and what are the outputs in order that the statement sequence can be sorted into the correct order. For an assignment statement, all variables to the right of the equal sign are input, the variable to the left is given the single numerical value of the right hand expression and so is the output. The idea is extended to cases where more than one element is an output so that all elements to the left of the equal sign are considered outputs, those to the right are considered input.

```
PROCEDURAL (A, B, C = D, E)
. . . block of statements
END
```

This tells the translator to treat the statements bracketed by the PROCEDURAL . . . END statements as a block (i.e., not to rearrange the order) and that it is to consider D and E as inputs; A, B and C as outputs. Only variables calculated elsewhere in the same DERIVATIVE block must be listed on the input list. Expert users can use judicious manipulation of the input and output lists in order to override the ACSL sort operation, if necessary.

Elements listed in lower case are syntactical elements (i.e., variables) and may be replaced by any character string that satisfies the definition. Elements listed in uppercase must have these characters exactly as spelled out in the statement. Ambiguities can normally be resolved by assuming that any applicable FORTRAN standards hold.

A number of special functions are described that aid in the definition of simulation models. In general, the output of each function is a single number (usually floating point) and the arguments are arithmetic expressions which may be of arbitrary complexity; i.e., these may contain functions which contain arguments which contain functions to any depth desired. Included in this list for completeness are all the standard FORTRAN library functions. In general, logical or relational expressions are used to determine switching criteria; for constants, .TRUE. and .FALSE. should be used since the bit pattern of logical variables depends on the installation and FORTRAN compiler in use.

The basic program structure is explained in detail in Section 3, which includes program flow and statement sorting. The program is set up to calculate the derivatives of the state variables (outputs and integrators) and the statements that perform this operation are bracketed by PROGRAM . . . END cards. Each statement will have certain input variables and it will calculate the value of one (usually) or more output variables. The program sorting sequence will rearrange the statement sequence if necessary so that values are not used before they have had fresh values calculated. Blocks of code that the user requires for a fixed sequence may be bracketed by PROCEDURAL . . . END cards which tell the sort routine to treat the included cards as a single block. In that case all inputs needed by the block and all values output must be included in the argument list of the PROCEDURAL (q.v.) block header. A variable may not be both input and output for a single PROCEDURAL BLOCK simultaneously. Variables may be both input and output to a statement if they cross a memory or integration operator., i.e.

```
X = INTEG(-K*X + F, XIC)
Z = DELAY(K1*Z + K2*F, . . .)
```

is a normal model description but

$$X = Y + 3.0*\text{SIN}(X)$$

is illegal since the sort cannot be performed. See the IMPL operator.

More flexibility is obtained by including INITIAL, DYNAMIC and TERMINAL sections to evaluate code only at the beginning (INITIAL), at every communication interval (DYNAMIC) or at the end (TERMINAL). Statements included in these sections will not have the order of calculation changed; they will be executed directly in the sequence given.

The use of labels on statements normally implies that the order of the statements is important. ASSIGN, GO-TO and IF statements are normally associated with changing the order of program execution. In order to avoid the rearrangement that goes on in the DERIVATIVE section of an explicit program or within the PROGRAM block of an implicit program they should normally be enclosed in PROCEDURAL . . . END brackets so that the group of statements is treated as a whole. It is not necessary to bracket blocks in the INITIAL, DYNAMIC or TERMINAL region since these are not sorted into a different order.

Some of the operators involve state variables and can only be invoked from within a DERIVATIVE section. While they can be included in a first level PROCEDURAL block these must always be executed and cannot be successfully by-passed by jumping around operator statements. The following is a list of such operators:

CMPXPL	DBLINT	INTEG	LEDLAG
OU	REALPL	TRAN	

If an attempt is made to skip around any of these statements, the derivative for the state variable will usually be left a non-zero value (constant while the operator is skipped) so that the internal state variable will continue to change. The correct method for stopping a state variable changing is to ensure that the derivative goes to zero.

Other operators are defined using a MACRO skeleton in which case two alternative forms of invocation are possible when the operator only has one output. As an example consider REALPL, the first order lag. A conventional use of this operator is:

$$Y = K1*\text{REALPL}(T1, X)$$

where the state variable (output of the real pole function) is given an assigned dummy name and so not considered visible. It is usually advantageous to multiply the input by constants rather than the output - synonymous if the operator is linear - as so:

$$Y = \text{REALPL}(T1, K1*X)$$

where the input to the operator is now an expression. If the form of the statement is then a single assignment, the stand alone macro invocation can be used as so:

$$\text{REALPL}(Y = T1, K1*X)$$

and in this case (only) the variable Y will be assigned to the state table. This stand alone form is usually preferred in order to minimize the number of internally generated variables but numerically all forms are equivalent. Operators that can be expressed in stand alone form are so indicated in the following list.

4.1 ABS

Absolute value of the argument expression

$$y = \text{ABS}(x)$$

x is a real floating point expression

4.2 ACOS

Arc-cosine is a function

$$y = \text{ACOS}(x)$$

where x is real floating point variable or expression lying between - 1.0 and + 1.0. Result is real number in radians (zero to π).

4.3 AINT

Integerize by using the functional form

$$y = \text{AINT}(x)$$

where x is a real floating point variable or expression of arbitrary complexity. Result - Y sign of X times largest integer $\leq |x|$ and is a real floating point number.

4.4 ALGORITHM

The algorithm used by the integration routine may be chosen and/or the system variable name changed. The name IALG is the default and may be set numerically at run-time to control the execution algorithm. Standard form is

$$\text{ALGORITHM name} = \text{integer constant}$$

where "name" will be the new name for the integer defining the run-time algorithm.

At present there are seven integration algorithms available as listed in Table 4-1. The Adams-Moulton (IALG=1) and Gear's Stiff (IALG=2) are both variable step, variable order integration routines that are self-initializing. In general they will attempt to keep the per-step error in each state variable below the desired value. This desired value is obtained by taking the maximum of the corresponding absolute allowed error (XERROR) and the relative allowed error (MERROR) multiplied by the maximum absolute value of the state so far

$$E_i = \max(X_i, M_i |Y_i|_{\max})$$

TABLE 4-1. Integration Algorithm Numbers

IALG	ALGORITHM
0	Sample Data Systems
1	Adam's Moulton; variable step, variable order
2	Gears Stiff; variable step, variable order
3	Runge-Kutta first order or Euler
4	Runge-Kutta second order
5	Runge-Kutta fourth order
7	User supplied subroutine (INTEG)

The order of integration starts at one and then changes dynamically as the program progresses, along with the calculation interval as the integration routine attempts to take the largest possible step consistent with the allowed error bounds.*

* For more information see subroutine DIFSUB in "Numerical Initial Value Problems in Ordinary Differential Equations" C.W. Gear, Prentice-Hall, N.J. 1971 pp 150 et seq.

Gear's Stiff integration method can take calculation steps that are orders of magnitude larger than the smallest time constant. There is an overhead involved, however, since a linearized state transition matrix must be formed and inverted. Tests have shown that for problems where the range of time constants only differs by one or two decades, there is little benefit in using this method - the Adams-Moulton technique is invariably faster. If the range of time constants covers more than three or four decades, then this technique should be significantly faster than any other.

A problem was noted in using the Stiff integration algorithm with one of the states only defined when positive. This difficulty occurred in integrating for mean square receiver noise in a missile adjoint formulation. The simulation model took the square root of this quantity. In starting off the algorithm the linearized state derivative matrix - or Jacobian - is evaluated numerically by perturbing each state first minus, then plus* and computing the approximate slope from the change in the derivative value. The perturbation magnitude used to compute this slope is the current allowable error obtained from MERROR and XERROR. If the positive state starts with an initial condition of zero, then the first perturbation will make it negative - an unacceptable situation for the derivative evaluation. The way round this problem is to start the integrator off with enough bias in the initial condition so that the calculation of the Jacobian doesn't violate any constraints. Since the number used for the perturbation is the allowable per-step error, then a bias of this amount should not affect overall model validity.

At the end of a simulation run that uses a variable step-size algorithm, statistics are written out giving the weight each state had in controlling step size. You can adjust the error criterion using this information. A separate listing indicates the number of times the predictor-corrector algorithm failed to converge and cause a general step size reduction - usually considered a more serious failure than bumping into the allowable error tolerance. This error summary may be suppressed by setting (SET) the system variable WESITG (write error summary, integration control) to .FALSE. - useful when running from a terminal.

Current step size (CSSITG) and current integration order (CIOITG) are available as system variables which may be OUTPUT or PREPARED. CIOITG is an integer and cannot be plotted.

The Runge-Kutta routines (IALG = 3, 4 and 5) work by evaluating the derivatives at various points across a calculation interval. A weighted combination of these derivatives is then used to step across the interval. Runge-Kutta second order (IALG = 4) advances the state with two derivative evaluations per step. This usually needs a smaller step than Runge-Kutta fourth order (IALG = 5), four derivative evaluations per step): However, for the same step size, it should run about twice as fast: Experiment is worthwhile for production jobs.

Integration algorithm zero is treated as a special case and is included to model discrete controllers with samplers controlling transfers to and from the continuous world - modelled by another DERIVATIVE block with a smaller effective step size. DERIVATIVE blocks with an integration algorithm of zero are called DISCRETE blocks and are further described under that heading.

4.5 ALOG

Natural logarithm of real argument x by

$$y = \text{ALOG}(x)$$

x should be > 0.0.

NOTE: This function is NOT the antilogarithm.

* Normally if the system variable TSMITG (two sided matrix evaluation, integration control) has its default value .FALSE., the states are only perturbed in the negative direction.

4.6 ALOG10

Logarithm to base ten of real argument x is given by the function

$$y = \text{ALOG10}(x)$$

x should be > 0.0 .

4.7 AMAX0

Determine maximum argument; standard functional form

$$y = \text{AMAX0}(j_1, j_2, \dots, j_n)$$

where the j_i are integer variables or expressions. Y will be the real floating point value of the maximum argument.

4.8 AMAX1

Determine maximum argument; standard functional form

$$y = \text{AMAX1}(x_1, x_2, \dots, x_n)$$

where the x_i are real, floating point variables or expressions of arbitrary complexity. Y will be given the value of the maximum x_i . Negative values are considered less than positive values.

4.9 AMIN0

Determine minimum integer argument and convert to floating point; standard form

$$y = \text{AMIN0}(j_1, j_2, \dots, j_n)$$

Similar to `AMAX0` except returns the value of the minimum argument.

4.10 AMIN1

Determine minimum real argument; standard functional form

$$y = \text{AMIN1}(x_1, x_2, \dots, x_n)$$

Similar to `AMAX1` except returns the value of the minimum argument.

4.11 AMOD

Remainder of modulus can be obtained by

$$y = \text{AMOD}(x_1, x_2)$$

which returns the floating point remainder when dividing x_1 by x_2 . The definition is a bit loose if x_2 is negative so it is actually defined as $x_1 - [x_1/x_2] x_2$ where $[]$ determines the integer with magnitude not greater than the argument and with the same sign. x_1 and x_2 may be real floating point variables or expressions of arbitrary complexity.

4.12 ARRAY

Equivalent to the FORTRAN DIMENSION statement, this operator allocates space for up to three dimensions to be associated with a variable name. Standard form of the ARRAY statement is

$$\text{ARRAY } v_1, v_2, v_3, v_4, \dots, v_n$$

The variable names v_i may have 1, 2 or 3 integer CONSTANT subscripts separated by commas; i.e., `SPACE(5,5,5)`. (Note the subscripts must not be symbolic). While this statement can, in general, appear anywhere in the program, it is recommended that it be placed in the beginning of each module. It must appear

before any invocation of a MACRO that uses dimension information. Section 10 of Appendix A gives an example of the use of this statement.

4.13 ASIN

Arc-sine of the real argument x is

$$y = \text{ASIN}(x)$$

x should be between - 1.0 and + 1.0 and the result will be in radians ($-\pi/2 \leq Y \leq + \pi/2$)

4.14 ASSIGN

A label can be assigned to a simple integer variable for use later in an assigned go to. Standard form of the statement is

ASSIGN k TO m

where k is a statement label and m is a simple integer variable. It is important for the ACSL translator to find a space between the label k and the following TO. Subsequent evaluation of

GO TO m

will transfer control to the statement labelled, k. Example:

ASSIGN RTURN2 TO M

The variable M should be explicitly typed INTEGER.

4.15 ASSIGNMENT STATEMENT

variable = expression

'variable' may be simple or subscripted. If subscripted in a sort section it must be enclosed in a PROCEDURAL block. On execution, the single value of the expression is stored into the location defined by 'variable'. A special form is the integration statement

A = INTEG(expression, AIC)

which marks A as a state variable. It actually stores a value (of expression) in the derivative of A.

The integration statement can be embedded at any depth in an expression since it only has one value, so the following is valid.

X = A*INTEG(DERIV, IC) + 3.0*C

In this case a made-up variable is assigned to be a state and the value of this used in the expression. This form is not recommended since states should be well defined and accessible to the user. Define your own intermediate variable, i.e.,

ST = INTEG(DERIV, IC)

X = A*ST + 3.0*C

4.16 ATAN

Arc-tangent of real argument x is

$$y = \text{ATAN}(x)$$

x is unlimited - except infinite - and result Y will be in radians such that ($-\pi/2 < Y < + \pi/2$).

NOTE: For full coverage of the circle it is better to use ATAN2 (q.v.).

4.17 ATAN2

Arc-tangent of angle formed by the point with real coordinates x , y and the positive X axis. Function is

$$z = \text{ATAN2}(y, x)$$

x and y can be both positive and negative, defining the full circle of revolution. Result will be in radians such that $-\pi < z \leq +\pi$.

4.18 BCKLSH

Backlash may be modelled using the following standard forms:

$$y = \text{BCKLSH}(ic, dl, x)$$

$$\text{BCKLSH}(y = ic, dl, x)$$

where

ic = initial conditions on y . Output will always lie between the limits $x - dl$ and $x + dl$

dl = half width of backlash - see Figure 4-1

x = input, a real variable or expression

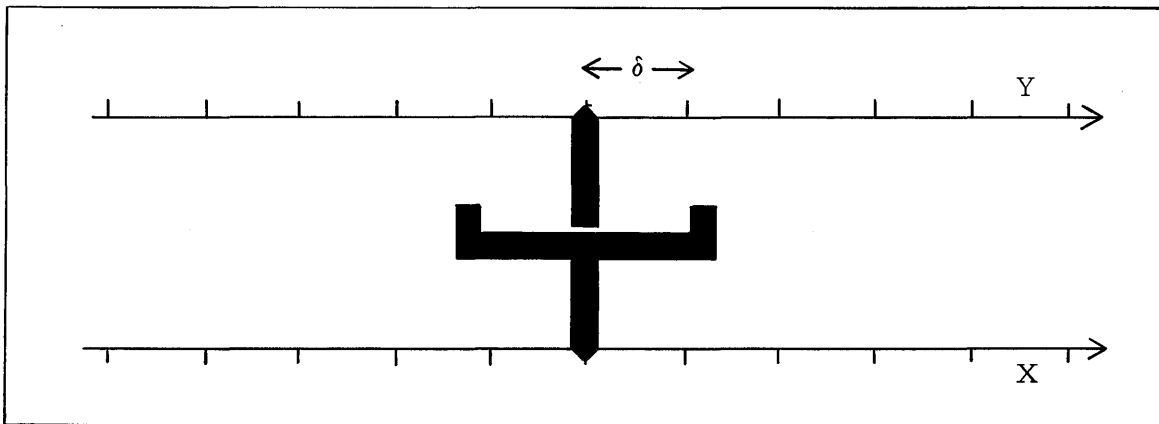


Figure 4-1. Mechanism Illustrating BCKLSH Operator

For unsymmetric applications change the expression for the input x , i.e. If Y is to move when x is greater than UL or less than LL then the invocation would be

$$Y = \text{BCKLSH}(YIC, 0.5*(UL - LL), X - 0.5*(UL - LL))$$

4.19 BOUND

Variables may be bounded or limited with this function. It should not be used to limit the output of an integrator since the integrator itself will continue to wind up and must actually integrate out of the limit. The function LIMINT should be used in this case, which sets the derivative to zero if it is still driving into the limit. On sign reversal the integrator will immediately come out of the limit. Standard form of the BOUND function is

$$y = \text{BOUND}(ll, ul, x)$$

Result

$$\begin{array}{ll} y = ll, & x < ll \\ y = x, & ll \leq x \leq ul \\ y = ul, & x > ul \end{array}$$

4.20 CALL

Subroutines may be invoked by an explicit call which has the standard forms

CALL name

CALL name (p_1, p_2, \dots, p_n)

name is the name of the subroutine being called and p_i are actual arguments which may be expressions of arbitrary complexity for input values. Variables, arrays and subscripted variables may be used for arguments that have values stored into them. Note that in this form the translator cannot tell which arguments are inputs and which are outputs so the call must be embedded in a PROCEDURAL block if it is in the sortable section of the program (DERIVATIVE), i.e., if O_i are output variables and P_i are input variables or expressions, then the following cards could be used

PROCEDURAL ($O_1, O_2, O_3 = P_1, P_2, P_3, P_4$)

CALL SUBR($O_1, P_1, P_2, P_3, P_4, O_3, O_2$)

END

Now the translator will handle this section as a block and it will have been told which variables are inputs and which are outputs.

A third form of the call is provided just in case the subroutine happens to be written with the input expressions on the left and the output variables on the right. In this case use

CALL SUBR($O_1, O_2, \dots, O_m = P_1, P_2, \dots, P_n$)

where the O_i are output variables - names or arrays that have their values calculated by the subroutine and the P_i are input expressions.

The translator will rearrange the order and change the equal sign (=) to a comma so that the resulting call will be

CALL SUBR($P_1, P_2, \dots, P_n = O_1, O_2, \dots, O_m$)

4.21 CINTERVAL

The communication interval is the interval where control returns to the dynamic section and output variables have their values recorded. In general, no finer grain detail can be seen, so it is extremely important that the user choose this communication interval with care. The standard form of the statement is

CINTERVAL name = real constant

where name is a simple, unsubscripted variable.

The system default name is CINT and the default value is 0.1. The value of the name defined in the CINTERVAL statement may be changed by the program so that different phases can be viewed at different data rates.

Assume we have a missile simulator that has four phases of flight:

- 1) Initial turn
- 2) Midcourse
- 3) Acquisition
- 4) Terminal

Now we would like to measure these phases at different rates - initial turn at a fairly fine level of a tenth of a second; the long midcourse only every second and acquisition and terminal at the fine rate of every 50 msec. In the initial section (Figure 4-2) PHASE is declared to be an integer and initialized to one. An array STEP is defined and filled with the desired communication intervals to match each phase of flight; PHASE is one for initial turn, two for midcourse, three for acquisition and four for terminal. In the DYNAMIC section the communication interval (using default name CINT) is set using the current value of the flight PHASE (will range from one to four). In the derivative section the value of PHASE is computed from the logic used to establish the different flight regions. This is shown as a block since the algorithm will depend on the implementation of the model. However, when it is implemented, the value of PHASE should be maintained in the range one to four.

INITIAL SECTION	<pre> INTEGER PHASE ARRAY STEP (4) CONSTANT STEP = 0.1, 1.0, 0.05, 0.05 PHASE = 1 \$ 'INITIALIZE PHASE TO START'</pre>
DYNAMIC SECTION	<pre> ... CINT = STEP (PHASE) ...</pre>
DERIVATIVE SECTION	<pre> 'COMPUTE PHASE OF FLIGHT' PROCEDURAL(PHASE = , , ,) ... END</pre>
	<pre> ... END</pre>

Figure 4-2. Example of Program to Vary Communication Interval

The following example changes the name of the variable defining the communication interval (from the default CINT)

```
CINTERVAL CI = 0.01
```

Now all references should be made to the symbol CI when the value is to be recorded or changed.

CINT acts as an upper bound on the integration step size, except for integration algorithm zero, and also for the last step of a communication interval since the model will be advanced to the data recording time. This question is discussed in more depth under DYNAMIC (qv). The actual integration step is obtained from the following statements.

```
H = MAX(MINT, MIN(MAXT, CINT/NSTP))
```

```
H = MIN(H, time left to next event)
```

which applies bounds of MAXT and MINT to first guess of CINT/NSTP and then limits this to be no more than the time left in the current communication interval or to the next event (DISCRETE section)

4.22 CMPXPL

A second order transfer function may be conveniently implemented by using the standard forms:

```
y = CMPXPL(p, q, x, ic1, ic2)
```

```
CMPXPL (y = p, q, x, ic1, ic2)
```

Results: y will be related to input x through the transfer function

$$\frac{y}{x} = \frac{1}{ps^2 + qs + 1}$$

```
y'(0) = ic1
```

```
y(0) = ic2
```

The same restriction on IC's are present as for the INTEG operator; both may be omitted if zero. p and q may be expressions of arbitrary complexity (Figure 4-3).

```
MACRO CMPXPL(Y,P,Q,X,IC1,IC2)
MACRO STANDVAL IC1=0.0,IC2=0.0
MACRO REDEFINE YDOT
YDOT=INTEG(((X)-(Y)-(Q)*YDOT)/(P),IC1)
Y=INTEG(YDOT,IC2)
MACRO END
```

Figure 4-3. Listing of CMPXPL Operator Macro

4.23 COMMENT

Any complete statement may be inclosed in quotation marks and it will then be ignored by the translator. It is illegal to ADD the quoted string to another statement, i.e.,

```
A = B + C 'THIS IS AN ASSIGNMENT'
```

is all one statement and hence illegal. The card can be broken up into two statements by

```
A = B + C $ 'THIS IS AN ASSIGNMENT'
```

The normal rule regarding continuations using the trailing ellipsis applies, i.e.,

```
A = B + C $ 'THIS IS A LONGER COMMENT . . .
```

```
REGARDING THE ASSIGNMENT OF A VALUE TO A'
```

Since the quotes must be balanced and the statement separator (\$) overrides all other operators, comments cannot contain quotation marks or dollar signs.

4.24 CONSTANT

Similar to a FORTRAN DATA statement, this operator is used to preset symbolic locations with numeric data. Standard form of the CONSTANT statement is

```
CONSTANT d1 = a1, d2 = k*a2, d3 = a31, a32, k*a33
```

where

d_i = Identifiers representing simple variables or array names. Implied do-loop notation may not be used. If d_i is an array name, integer subscripts may be used to fill individual elements within the array, else the entire array must be filled.

a_i = Literals and signed or unsigned constants.

k = Integer constant repetition factors that cause the literal following the asterisk to be repeated k times.

No check is made that reals are stored into reals, integers into integers and logicals into logicals. A common error is to omit the decimal point from a number which will then be stored as an integer. When the symbolic name is used as a real number, the integer stored there may be considered zero. An example of the correct use of the CONSTANT statement is:

```
LOGICAL    L1
INTEGER    II
ARRAY      A(2)
CONSTANT   L1 = .TRUE., II=2, A=2*1.0, B = 5.76
```

4.25 CONTINUE

Is a do-nothing statement which is normally used for a label to transfer control to or terminate a DO-loop. For preference, this statement should be used for all labels due to problems with MACRO expansions - see sub-section 2.3.

4.26 COS

Takes the cosine of a real argument x which must be in radians

```
y = COS(x)
```

result will be $-1.0 \leq Y \leq +1.0$

4.27 DBG

Debug features are built into the translator and varying sections of the translator process can be printed on command via this statement. It should normally only be used under instruction of someone familiar with the programming of the translator.

4.28 DBLINT

A special operator is provided for integrator limiting when the limited output is the second integral of an acceleration. This type of limiting can best be described in terms of the mass spring-damper system described by

$$m\ddot{x} + b\dot{x} + cx = f(t); \dot{x}(0) = XDZ, x(0) = XZ$$

where physical stops constrain the mass to move only between x_{ll} (lower limit) and x_{ul} (upper limit) i.e.,

$$x_{ll} \leq x \leq x_{ul}$$

When the displacement (x) of the mass reaches its limit, the mass must stop, implying that the velocity (\dot{x}) is zero. The mass must remain stopped until the force acting on it ($f(t) - cx$) changes sign. The DBLINT operator (double limited integration) can be used to represent this type of limiting. The standard form of the invocation is

DBLINT(x, xd = xic, xdd, xdic, ll, ul)

where

- x = a displacement
- xd = a velocity (\dot{x})
- xic = x(0)
- xdd = the input, an acceleration (\ddot{x})
- xdic = $\dot{x}(0)$
- ll = the lower limit
- ul = the upper limit

Figure 4-4 gives a listing of the macro operator.

```
MACRO DBLINT(X, V, XIC, A, VIC, LBX, UBX)
MACRO REDEFINE VL, IC
CONSTANT IC=0.0
CALL ZZDLIM(V, VL=IC, X, INTEG(ZZLIMF(X, A, LBX, UBX), VIC), 0.0, LBX, UBX)
X=INTVC(V, XIC)
MACRO END
```

Figure 4-4. Listing of DBLINT Operator Macro

An alternate way of performing this operation is to wrap a stiff spring around the loop when the wall is approached - this corresponds to what happens physically since the wall will always have a finite spring constant (Figure 4-5). The problem with this representation is in the behavior of the digital integration routine in the vicinity of the wall when the wall stiffness is made extremely high.

4.29 DEAD

Dead space has standard form

$$y = \text{DEAD}(ll, ul, x)$$

Result is:

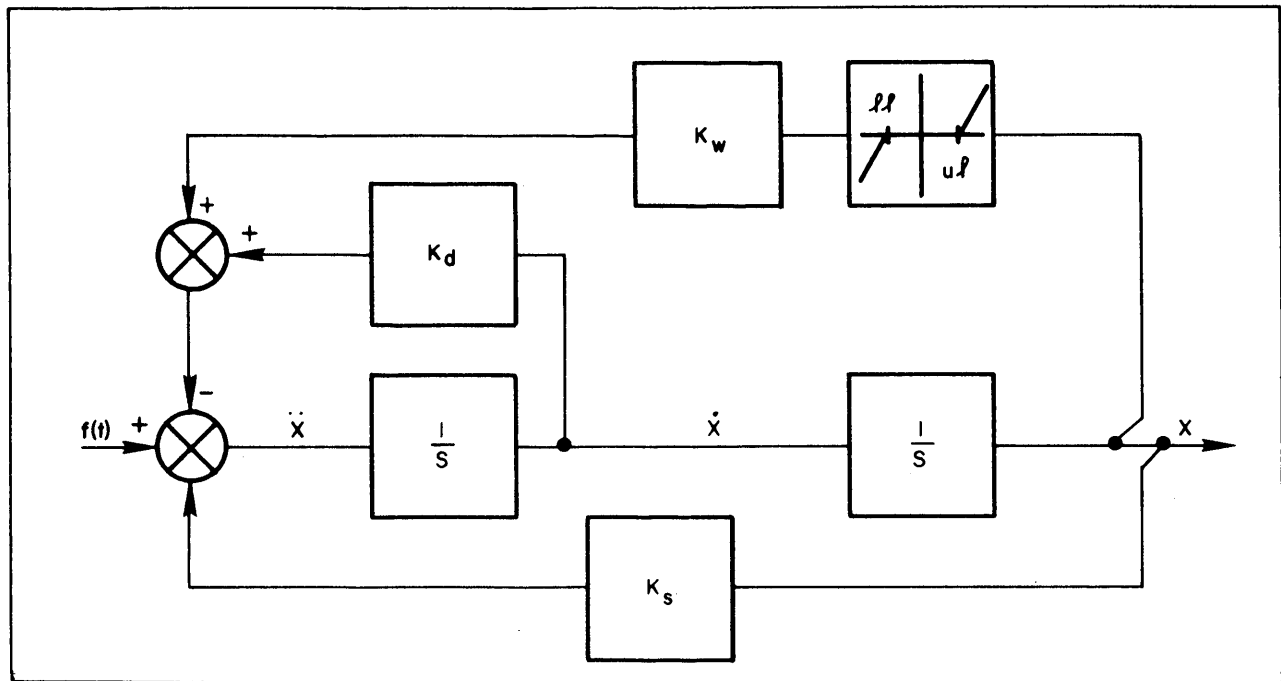


Figure 4-5. Limited Displacement - Spring Stiffness, K_s , Wall Stiffness K_w , Damping Constant K_d

$$\begin{aligned}
 y &= x - ll, & x < ll \\
 y &= 0.0, & ll \leq x \leq ul \\
 y &= x - ul, & x > ul
 \end{aligned}$$

4.30 DELAY

A variable must be delayed in time to model passage through a pipe or other transport effects. This operator should not be used lightly since it tends to use a lot of storage and the extra calculation time can be significant.

The operator is invoked by the standard function forms

$$y = \text{DELAY}(x, ic, tdl, nmx)$$

or

$$\text{DELAY} (y = x, ic, tdl, nmx)$$

where

- x = the input - an arithmetic expression of arbitrary complexity
- ic = the initial value of the output until the independent variable has advanced by the delay, tdl
- tdl = the delay between input and output
- nmx = an integer CONSTANT (i.e 10) giving the maximum number of calculation intervals in the delay. The calculation interval may vary but the sum of nmx calculation intervals must always be greater than the current time delay.

This operator is implemented by allocating a dummy array, 2*nmx words long, and prefilling with the value of ic, extending over all past history. Each entry in the table is associated with a time and at each new calculation interval a new value is inserted into the array - treating it as a circular list. To compute the output value, tdl, the current time delay is subtracted from the independent variable value and the table is searched for time values that bracket this required previous time. A linear interpolation is performed between the corresponding input values. If not enough data points are present, an error is reported. This operator approximates the pipeline with a varying velocity. For best results the time delay should vary slowly.

NOTE: This operator requires the sort algorithm to separate sections of code and so should not be included in a PROCEDURAL block.

4.31 DERIVATIVE

Identifies the blocks of code performed at the request of the integration routine to evaluate the state variable derivatives. The integration routine is called from the dynamic section (Subsection 3.1) and asked to advance the state over the next communication interval using the code embedded in the DERIVATIVE blocks to evaluate the state variable derivatives. The actual number of evaluations depends on the integration algorithm employed. All the statements in the DERIVATIVE blocks are translated into a separate subroutine so it is illegal to transfer control by GO TO's from these blocks to other sections (INITIAL, DYNAMIC, or TERMINAL) or vice versa.

More than one DERIVATIVE section may be used, each with its own independent integration algorithm and integration step size. Although this technique can save execution time when correctly used, any implementation must be approached with caution since, in general, incorrect answers will be obtained unless the model is split with a full understanding of the effects of computation delays for variables that cross block boundaries. This tool has been provided for research into problem split rules and should be regarded more as a state-of-the-art technique rather than for every day practical implementation. For anyone who considers himself a novice, no more than one DERIVATIVE section should ever be used, unless it is to model a discrete controller where natural delays occur.

The basic structure is to include more than one derivative block - delimited by DERIVATIVE...END statements within the DYNAMIC block, i.e.

```
DYNAMIC
DERIVATIVE SECTION 1
...
END $' OF SECTION 1 '
DERIVATIVE SECTION 2
...
END $' OF SECTION 2 '
etc
END $' OF DYNAMIC '
```

Each derivative section can have its own integration algorithm, maximum step size, minimum step size and NSTEPS value. Default values for these quantities are established in normal fashion. If a value is specified outside a DERIVATIVE section, this becomes the default for all DERIVATIVE sections. Values relating to a particular DERIVATIVE section are defined by including the appropriate statement between the DERIVATIVE . . . END delimiters, i.e.

```
ALGORITHM IALG = 4
```

```

NSTEPS NSTP = 1
DERIVATIVE SEC1
  ALGORITHM ALG1 = 3
  MAXTERVAL MAX1 = 0.001
  ...
END $' OF SEC1 '
DERIVATIVE SEC2
  MAXTERVAL MAX2 = 0.010
  ...
END $' OF SEC2 '

```

The first two statements establish algorithm number four (4) to be the default algorithm and each section will have a NSTP of 1. Within derivative section one, the algorithm is specified to be three (3) with a maximum step size of 1 msec (0.001). Within derivative section two, the default algorithm is taken and the maximum step size set to 10 msec (0.010). Since NSTP is one, these will be the actual step sizes.

Implementation for these block descriptor names is by forming an array of length the number of derivative sections for each one of the describing quantities. The names used are the default or those specified outside the DERIVATIVE sections. In the preceding example, this would be IALG(2), NSTP(2), MAXT(2) and MINT(2). Names specified within the derivative sections are equivalenced into appropriate slot of the main array, i.e.

```

IALG(1) ≡ ALG1
MAXT(1) ≡ MAX1
MAXT(2) ≡ MAX2

```

and preset to the value indicated. These values can be changed by SET commands at run-time. To change the integration algorithm for section two, it must be referred to as IALG(2) since a name was not explicitly given, i.e.

```

SET ALG1 = 5, IALG(2) = 5

```

Because of the equivalencing convention, it is important that unique names be used for descriptors defined within DERIVATIVE blocks.

At the start of the simulation run, the DERIVATIVE blocks are placed on an event list and executed in the order in which they are specified in the model definition section. Note that the sort algorithm cannot rearrange statements over a block boundary, so that if a value calculated in the second block is used in the first block, the first time it will be undefined. Such quantities should be initialized in the INITIAL section.

Each block is placed back on the event list assigned with the time to which that block has advanced. In the case given of a 1 msec step associated with Section 1 and a 10 msec step associated with Section 2, the event list will look like:

A	B	C	D	E	F	G
0.0 1	0.0 2	0.001 1	0.002 1		0.010 1	0.010 2	0.011 1
0.0 2	0.001 1	0.010 2	0.010 2		0.010 2	0.011 1	0.020 2

After Section 1 has been advanced one step to 1 msec, the event list has Section 2 at the top (B) with a time of 0.0. Advancing this section one step leaves it at 10 msec and now Section 1 is back at the head of the list again (C). Section 1 keeps on being advanced until they are both at 10 msec (E). Now we assume that 1 is a little ahead of 2, and goes to 11 msec (F). Next step advanced 2 by 10 msec and the event list picture changes to (G). This cycle then repeats.

The only break in the regular progression is at a communication interval or at an equivalent barrier represented by a DISCRETE block or a block with an integration algorithm of zero. The times for all DISCRETE blocks and also the communication action are entered onto a separate event list where all actions are ordered in time; the next time on the event list is called the barrier time.

For all other DERIVATIVE blocks, the current step size is checked against the current time (T) and the barrier time from the next event list. If the integration with the current step size will exceed the barrier time, the actual step is reduced so that the last step is made exactly up to the event. All states will then line up in time for the event to take place. If the event is a communication interval data recording, the derivatives are evaluated once more so that all algebraic variables depending on the states will be consistent. This action is not taken for barriers established by DISCRETE blocks since differences are likely to be small and the overhead of an extra forced derivative evaluation comparatively high. If any problems are encountered remember that the state variables will be at the correct time and any algebraic variables can be rederived in the sampling code if necessary.

It seems most people choose step sizes and communication intervals that are integer multiples of each other. The ACSL system does not require this however, and it is acceptable to choose a fixed step length for the continuous section of 4 msec, a sampling INTERVAL (qv) in the DISCRETE block of 11 msec and a communication interval of 20 msec. The first few calculation intervals would then be

4, 4, 3, 4, 4, 1, 2, 4, 4, 3, 4, 3, 4, . . .

The first short step of 3 msec brings time up to 11 msec the first barrier time. Then two more normal steps are followed by a short step of 1 msec to bring time up to the communication time of 20 msec. The next barrier is the DISCRETE block at 22 msec causing a step size of 2 msec and so on. The variable step algorithms are similar in that the step length will vary but the last step will always be reduced to move the block exactly up to the barrier time. Note however for monitoring purposes, the variable CSSITG or current step size will contain the step length the integration algorithm would like to take, not the shorter one actually taken up to the communication interval time.

4.32 DERIVT

The derivative function can be implemented if ABSOLUTELY necessary. Note that it is never necessary to invoke a derivative - it can be expressed instead in terms of all the other states in the system. Since the derivative operator is a first order approximation, it can lead to instability if it is used to represent any major loop. The only time this may be justified is for a minor term where a large amount of extra calculation may be needed to reform the problem in terms of the states. Standard form is

y = DERIVT(ic, x)
DERIVT(y = ic, x)

where:

ic = Y(0)
x = the input, an arithmetic expression

NOTE: This operator should not be included in a PROCEDURAL block.

4.33 DIM

Positive difference is obtained by the function

y = DIM(x₁, x₂)

where x₁ and x₂ are real floating point variables or expressions. Result is

y = x₁ - x₂ if x₁ > x₂
y = 0.0 otherwise

4.34 DISCRETE

An equivalent to the DERIVATIVE block is introduced by the keyword DISCRETE. The intent is to make it easy to model digital sampled data controllers where the communication to and from the continuous world occurs at fixed, known in advance, times. The format of the statement is

```
DISCRETE name  
...  
END
```

inserted within the DYNAMIC code block, if any, at the same level as any DERIVATIVE blocks. The action of the statement is to demarcate a code sequence that is executed at a discrete event or time point with the execution being controlled by the keyword INTERVAL. Like the DERIVATIVE blocks, each DISCRETE block has a time associated with it which is entered into an event list, which time will become a barrier for all the other DERIVATIVE blocks ensuring that they all take a final step (may be short) up to the barrier time before the code in the DISCRETE section is executed. If the INTERVAL statement is included, the DISCRETE block is re-entered on the event list with a time equal to the current time plus the current contents of the INTERVAL variable.

An example using a DISCRETE block to model a control computer is given in Section 11 of Appendix A and is shown in outline here by the following structure

```
PROGRAM discrete controller  
...  
DERIVATIVE CONTIN  
...  
(X DEPENDS ON U)  
...  
END $' OF CONTINUOUS SECTION'  
DISCRETE SAMPLE  
INTERVAL DTSAMP = 0.100
```

(U DEPENDS ON X)

...

END \$' OF DISCRETE'

END \$' OF PROGRAM'

In the continuous section we define a plant that can be as complex as necessary but structurally has an output X that depends on a control U. The control U should be visualized as being generated via a digital to analog converter so that the value of the control remains constant between output intervals. The next section is a DISCRETE block that is executed every DTSAMP seconds, defined by the INTERVAL statement to be 0.1 seconds. This block uses the output value of the plant X, which will be at the sample time, to compute a control U which will be used over the next sample interval. The successful operation of this simulation requires the continuous section to be exactly at the sample time when the plant value X is used to determine the next control U.

Jumps in the control U will modify all the high order derivatives estimated for the continuous plant so variable step algorithms will usually have to reduce the step size or even restart, since the fixed step size algorithms retain no memory of previous history, the new step will start out with a new evaluation of the discontinuous derivatives. The key to the action is the fact that the effective time of the continuous section is exactly at the sample time when the DISCRETE section code is executed.

The order of execution of the blocks is as given in the model definition code at time equals zero and both DERIVATIVE and DISCRETE blocks can be mixed. In general, in a closed loop situation, some variables will have to be initialized in the INITIAL section since they are used before being calculated in a later block. Reference as an example the control variable U in the preceding code sequence which is used in the continuous section before it is calculated in the DISCRETE block. If used like this without initialization, the first value of U used in the continuous section will be the last value left behind by the previous run! A test of the presence of these initialization problems is to make two identical runs i.e. START \$ START. If any answers are different it usually indicates an error.

The sequence of events at the time-equals-zero condition differs somewhat from the corresponding DERIVATIVE blocks. For conventional DERIVATIVE blocks, the initial conditions are moved to the state variable array and the derivative code executed once. Then the integration algorithm takes over and in integrating over the first step re-evaluates the derivatives again (the variable step algorithm IALG = 1 or 2 will predict the state variables first before the correct iteration) so it appears that two derivative evaluations are used at time-equal-zero. With the DISCRETE block, the first evaluation takes place and then the block is placed back on the event list at some time in the future (depends on the INTERVAL statement) with any state variables still with their initial condition values. Now time will advance for other sections until the DISCRETE section barrier time is reached when the integration algorithm first advances the state variables, if any, to the current time by euler integration and then re-evaluates the DISCRETE section code. In general integration statements (INTEG or INTVC) are not used inside DISCRETE blocks since the facility is designed to represent sampling actions independent from the continuous physical world where the true integrations take place. There is sometimes a need for a simplified integration algorithm with a fixed step size and a single derivative evaluation per step. This has arisen in the past in translating DYNAMO models to run under ACSL. In this case the entire model can be placed in a DISCRETE block and no extra derivative evaluations will be inserted at the communication interval times. Since the states are always advanced just before a re-evaluation of the derivative code, algebraic variables are always synchronized to the state variables for data recording.

The mechanism for implementation of a DISCRETE section is to set an effective algorithm of zero in the corresponding IALG slot which can be seen from the debug output. The INTERVAL variable is equivalenced into the MINT (MINTERVAL) array and preset to the appropriate numeric value. Slots in the corresponding MAXT (MAXTERVAL) and NSTP (NSTEPS) arrays are ignored.

4.35 DO STATEMENT

The standard FORTRAN DO statement may be included. Note that the loop should normally be embedded in a PROCEDURAL block so that the sort routine will not rearrange the order of execution. This is not necessary within the INITIAL, DYNAMIC or TERMINAL regions of an explicit program since these are not sorted - each statement being executed in sequence. The loop cannot extend from INITIAL to TERMINAL to execute a sequence of runs. Each loop must be closed within its own block. The reason is the CONTIN statement that branches directly into the DYNAMIC loop. Most FORTRAN compilers will reject a branch into a loop. If a loop is established to make successive runs, then CONTIN cannot have any meaning since the initialization won't be performed properly.

A DO statement makes it possible to repeat a group of statements a designated number of times using an integer variable whose value is progressively altered with each repetition. The initial value, final value and rate of increase of this integer variable is defined by the set of indexing parameters included in the DO statement. The range of the repetition extends from the DO statement to the terminal statement, which must follow the DO statement, and is called the DO loop. The standard form of the DO statement is

```
DO n i = m1, m2  
DO n i = m1, m2, m3
```

where

- n = a label of the terminal statement of the loop, may be a symbol or numeric.
- i = a simple integer variable called the index variable. With each repetition its value is altered by the increment parameter, m₃. This variable may not be changed within the loop.
- m₁ = initial parameter, the value of i during the first loop.
- m₂ = terminal parameter, when the value of i surpasses the value of m₂, DO execution is terminated and control goes to the statement immediately following the terminal statement.
- m₃ = increment parameter, the amount i is increased with each repetition. If omitted - first form above - the value 1 is assumed.

The m₁, m₂, and m₃ must be simple integer variables or unsigned integer constants.

In general, the use of DO statements should be minimized. More flexibility is usually obtained by programming each loop explicitly, i.e.,

```
I = m1  
L1 .. IF(I.GE.m2) GO TO L2  
... } loop  
I = I + m3  
GO TO L1  
L2 .. CONTINUE
```

Now general expressions, arrays and real variables can be used for the indexing parameters.

For more information on the structure of the DO statement see the local FORTRAN reference manual. The ACSL translator checks the syntax of the DO statement but does not validate the correct nesting of loops or terminal statements. Errors of structure will be indicated by the FORTRAN compiler.

4.35 DYNAMIC

Identifies the block of code that is performed every communication interval throughout the run. It must be accompanied by its matching END card. Code executed in the DYNAMIC block can leave variable values around for use by the DERIVATIVE and DISCRETE blocks over the next communication interval and in that sense it can act as a sampler or D/A converter. It is preferable to include code calculations in a separate DISCRETE block and so make model behavior independent of the data recording action. The intent within the DYNAMIC block is to provide a place to put output related calculations so that they can be performed at the usually slower data recording rate rather than at each derivative evaluation. Examples are unit conversions such as radians to degrees or meters/seconds squared to gees. The time interval for the next communication interval can be itself changed, based on some simulation phase or configuration so obtaining variable data recording rates - see CINTERVAL statement for example.

No sorting is performed on any of the code within the DYNAMIC block.

4.36 END

Denotes the end of a block or section. Subsection 3.1 shows the use of END's in structuring an explicit program. One of the most common errors is not getting the right number of END's to balance off the program blocks. Error messages 'NOT ENOUGH ENDS' and 'TOO MANY ENDS' are issued when the count is incorrect. It acts like a right parenthesis in an arithmetic expression, except that it terminates blocks of statements. An incorrect count corresponds to unbalanced parentheses.

4.37 EQUIVALENCE

This operator is similar to the FORTRAN version in that it allows renaming of areas of storage. Due to the nature of storage in ACSL (all variables appear in a single common block) there are specific rules for the use of EQUIVALENCE which do not appear in normal FORTRAN usage. The standard form of the statement is:

```
EQUIVALENCE (Main variable, equivalenced variables(s)) . . .  
              ,(Main variable, equivalenced variables(s)) . . .  
              ,(etc.)
```

where "main variable" will appear in the users common block, and thereby reserve storage, while the equivalenced variables will be assigned storage relative to the main variable, but will not reserve space. All variables will appear in the dictionary with their own types and dimensionality.

Rules which restrict the use of EQUIVALENCE are as follows:

- 1) System variables (CINT etc.), States, Derivatives and Initial Conditions may only appear as main variables.
- 2) A main variable may never appear as an equivalenced variable.
- 3) An equivalenced variable must not appear more than once.
- 4) Variables may have no subscript or only one when being specified in an EQUIVALENCE statement. Multi subscripted arrays must appear using their equivalent single subscript. For example, given A(5,5), equivalencing the (3,3) element to SAM would require a reference to A(5*3 + 3) or A(18) i.e.

```
EQUIVALENCE (A(18), SAM)
```

In using equivalenced variables, any reference to an equivalenced variable name is taken to be a reference to its associated main variable, even through the equivalenced variable may be only one element of a large main variable array. Thus the sorter will produce the error message "multiply defined symbol" if two equivalenced variables are assigned values outside a PROCEDURAL block.

```
ARRAY RM(3)
EQUIVALENCE (RM(1), RM1), (RM(2), RM2), (RM(3), RM3)
```

...

```
RM1 = 0.0
```

```
RM2 = 10000.0
```

*****MULTIPLY DEFINED SYMBOL*****

Generally the main variable is equal to or larger than it's equivalenced variable(s) since the main variable reserves storage, i.e.,

```
ARRAY A(10)
EQUIVALENCE (A, B)
```

Here A is the main variable, B is the equivalenced variable. A reference to B uses the first element of A (= A(1)). B(2) is not allowed unless B is a separately defined array.

```
ARRAY A(10)
EQUIVALENCE (B, A)
```

Now B is the main variable - which reserves one word of memory - and A is the equivalenced variable. A reference to A(2) will access the variable that follows B in the common block, generally not known to the user.

4.38 ERRTAG

The name given to the variable that is used to indicate an attempt to reduce the step size below the minimum, MINT, can be changed. This name will be set .TRUE. to indicate an error by a variable step size integration algorithm which is reducing the step size to control the error bound. Standard form of the statement is

```
ERRTAG name
```

where 'name' is a simple unsubscripted variable. The type of 'name' will be automatically set to LOGICAL and it will be preset to .FALSE. to start with. The variable step integration routine will call the derivative subroutine once with the name given under ERRTAG set to .TRUE. if it needs to reduce the step size below the minimum calculation interval, MINT. If it is still .TRUE. on return, the termination flag for that run will be set. Control should then revert to the terminal section, or executive and the next command will be read. If provision is made to handle this case and reset the flag, then care must be taken that an endless loop is not formed.

4.39 EXP

Exponential of real argument x

```
y = EXP(x)
```

x is limited in size such that the maximum machine word size should not be exceeded by the exponential. Result is

```
y = ex
```

4.40 EXPF

The exponential function can be switched on or off. The output is a function rising on a time constant to 1.0 (ON = .TRUE.) or decaying to zero (ON = .FALSE.). Standard form is

```
y = EXPF (ic, ta, on)
```

EXPF (y = ic, ta, on)

where:

ic = Y(0) - should be between zero and 1.0

ta = the time constant (τ)

on = a logical variable or expression, denotes rise or decay

Results:

$y = 1.0 - \text{EXP}(-(T - T_0)/\tau)$; ON = .TRUE.

$y = \text{EXP}(-(T - T_1)/\tau)$; ON = .FALSE.

T_0 and T_1 are the times at which ON changed .FALSE. to .TRUE. and .TRUE. to .FALSE. respectively.

4.41 FCNSW

Function switch operator has standard form

$y = \text{FCNSW}(p, x1, x2, x3)$

Results:

$y = x1, p < 0.0$

$y = x2, p = 0.0$

$y = x3, p > 0.0$

4.42 FORMAT

The FORTRAN FORMAT statement may be used and must have a label. In an explicit program, the FORMAT statement and corresponding I/O statement (READ, WRITE, etc.) must both be in the DERIVATIVE section or neither. Standard form of the FORMAT statement is

L1 . . FORMAT(character string)

The ACSL translator does not check the detailed syntax of the FORMAT statement. For further specification, reference should be made to a standard FORTRAN manual.

4.43 GAUSS

A normally distributed random variable can be generated by

$y = \text{GAUSS}(m, s)$

GAUSS(y = m, s)

y will be normally distributed with mean m and standard deviation s.

NOTE: Warning listed under UNIF.

The seed for a random sequence can be reset by UNIFI or GAUSI. If not set, a different random sequence will be in effect for each run. Figure 4-6 gives a listing of the operator macro.

```
MACRO GAUSS(Y, AVE, SIG)
Y=AVE+GRV(ZZSEED)*(SIG)
MACRO END
```

Figure 4-6. Listing of GAUSS Operator Macro

4.44 GAUSI or UNIFI

The seed for the random number generator can be initialized by using this operator. Standard form of the call is

GAUSI(k)

UNIFI(k)

where k is an integer constant or expression and should be an odd number for a maximal length sequence. If it's small (1,3,5 . . .) the first five or ten values of random numbers will be highly correlated. Figure 4-7 gives a listing of the operator macro expansion. Only one initialization routine should be used since they both set the same seed variable.

NOTE: This operator should only be invoked in the INITIAL section of an explicit program or provision must be made to skip over it except at the beginning of each run in an implicit program. If it is repeatedly executed, the random numbers won't change.

```
MACRO GAUSI(SEED)
PROCEDURAL(=SEED)
ZZSEED=SEED
END
MACRO END
```

Figure 4-7. Listing of GAUSI Operator Macro

4.45 GO TO STATEMENT

GO TO statements transfer control to a labelled statement whose reference is fixed or which is assigned during execution of the program. The statement labels used in the GO TO statements must be associated with executable statements in the same program unit as the GO TO statement. In explicit programs, the INITIAL, DYNAMIC and TERMINAL sections exist in a single program unit. However, control cannot be transferred into the DYNAMIC region. The reason for this is that the integration routines must be initialized at the start of the run and this initialization operation is done on leaving the INITIAL section and entering the DYNAMIC section. Control can be transferred *from* the DYNAMIC region and between INITIAL and TERMINAL

regions. The DERIVATIVE section is a separate program block.

In the following a statement label may be either numeric or symbolic. Standard forms of the various labels are:

GO TO k

Execution resumes at the statement labelled k. Example:

GO TO LOOP

GO TO m,(n₁, n₂ . . . , n_m)

GO TO m

The statement acts as a many branched GO TO. m is a simple integer variable ASSIGNED a label value. The n_i are statement labels which should correspond to a possible label assigned to m.

GO TO (n₁, n₂, . . . , n_m), i

where:

n_i = statement labels

i = a simple integer variable that has been given an integer value between 1 and m.

If i has the value k, then control will be transferred to the kth statement label, n_k, in the above list.

WARNING: The inclusion of a large number of GO TO's is considered harmful to the successful completion of any simulation project.

4.46 HARM

A sinusoidal or harmonic drive function can be defined by:

y = HARM(tz, w, p)

Result:

y = 0.0 T < tz

y = SIN(w*(T - tz) + p) T ≥ tz

where:

tz = delay in sec

w = frequency in rad/sec

p = phase shift in rad

Note that if p is nonzero a jump discontinuity will be involved.

4.47 HYSTERESIS

Use the backlash operator BCKLSH

4.48 IABS

Absolute value of an integer argument can be obtained by the function

n = IABS(j)

j is an integer variable or expression of arbitrary complexity.

NOTE: Integer variables must be declared to be type INTEGER. The default type is REAL. The function name IABS should be typed INTEGER as well.

4.49 IDIM

Integer positive difference can be obtained by the function

$$n = \text{IDIM}(j_1, j_2)$$

where j_1 and j_2 are integer variables or expressions.

NOTE: IDIM should be declared of type INTEGER prior to its use.

Result:

$$\begin{array}{ll} n = j_1 - j_2 ; & j_1 \geq j_2 \\ n = 0; & j_1 < j_2 \end{array}$$

4.50 IF STATEMENTS

IF statements are used to transfer control conditionally. At the time of execution, an expression in the IF statement is evaluated and the result determines the statement to which the jump will be made. The most useful form is the one-branch logical IF. It has the standard form

IF(*lexpr*) *statement*

where *lexpr* is a logical or relational expression (one that has a single value - either .TRUE. or .FALSE.) and *statement* is any executable FORTRAN statement except another logical IF, a DO statement or END. Non-FORTRAN statements or those particular to ACSL may expand to more than one actual FORTRAN statement. Only a single statement can be included in the logical IF statement. Examples are:

IF(A. LT. 5.0) A = A + 0.1

IF(T. GT. TSTOP) GO TO FINISH

The real switch (RSW) operator (q.v.) can often be used in place of an IF which then avoids having to bracket the IF code sequence by PROCEDURAL . . . END statements in order that statement order can be maintained.

The three-branch arithmetic IF is also included for completeness. In general, these should be avoided entirely due to the difficulty encountered in following any complex branching structure. Use only logical IF's given before with explicit GO TO's which effectively act like a two-branch IF and avoid the statement label on fall through. Standard form of arithmetic IF is

IF(*c*) n_1, n_2, n_3

where:

c = any arithmetic expression

n_i = statement labels

Control is transferred as

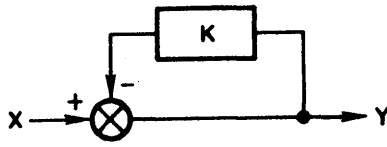
$c < 0$, jump to statement n_1 ,

$c = 0$, jump to statement n_2 ,

$c > 0$, jump to statement n_3

4.51 IMPL

The solution of simultaneous or algebraic equations cannot be written directly because of the sort algorithm. Every value is assumed calculable on the outputs of the integrators or state variables. Equations of the form $x = f(x)$ cannot be expressed in this form and it is necessary to find another way to calculate the output. Often an algebraic operation can be performed that can simplify the result. Feedback round a summer,



leads to

$$Y = X - KY$$

which cannot be modeled directly.

A little algebra can easily give

$$Y = \frac{X}{1 + K}$$

Some functional relations may be impossible to solve explicitly and so the implicit operator can be invoked. This will separate out the algebraic loop and use a Newton-Raphson iteration method to try to find a solution.

WARNING: This operation can multiply the computer time need for a simulation study many times. It is worth spending some time trying to avoid implicit loops.

The standard form for the function is:

$$y = \text{IMPL}(yz, e, m, efl, \text{expr}, ydl)$$

$$\text{IMPL}(y = yz, e, m, efl, \text{expr}, ydl)$$

where:

yz = initial guess

e = the error bound - real constant or arithmetic expression

m = the maximum number of iterations to try - must be an integer constant or variable - not an expression

efl = the error flag - a variable only - which is set nonzero if the iteration does not converge

expr = the expression that contains or eventually leads back to Y

ydl = the value used to increment the initial value to estimate the derivative. If not specified the assumed value is 0.0001

NOTE: The IMPL operator should not be used in a PROCEDURAL block that will prevent the correct sorting of any intervening statements.

The iterative method used is Newton-Raphson to solve

$$Y = f(Y)$$

$$C_n = \frac{f_n - f_{n-1}}{Y_n - Y_{n-1}} \quad \text{is the derivative estimate}$$

$$Y_{n+1} = \frac{f_n - C_n Y_n}{1 - C_n}$$

Upon initial entry

$$Y_{n-1} = YZ$$

$$Y_n = (1.0 + Y_n) * Y_{n-1}$$

The error criterion for successful completion of the iteration is

$$|Y_n - f_n| \leq |Y_n| * E$$

4.52 INITIAL

Identifies the block of code that is performed just once before the start of each run. It must be accompanied by its own matching END card enclosing the block and the block must be placed before the DYNAMIC (q.v.) section. Refer to Subsection 3.1 for rules of explicitly structured programs. No sorting is done on any code within the INITIAL block.

4.53 INT

Integerization of a real floating point argument

$$n = \text{INT}(x)$$

n will be the sign of x times the largest integer $\leq |x|$

NOTE: INT should be declared of type INTEGER prior to its use.

4.54 INTEG

All integration in an ACSL program is handled by a centralized integration routine. In performing integrations, the integration algorithms utilize two intervals - the calculation interval and the communication interval. Since digital integration is basically a discrete process, the calculation interval is the fundamental interval over which the state variables are updated. No finer detail is accessible except by some interpolation scheme. All integration schemes for the set of first order differential equations

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

finish up looking like

$$x_{n+1} = x_n + hf(x_{n+\alpha})$$

where $0 \leq \alpha \leq 1$. The problem is to find the effective derivative to be used in updating the state vector. Note that with suitable conditions on continuity and differentiability, the mean value theorem guarantees that an α exists that will produce an exact answer for the state trajectory - finding it is another matter, however. Different integration schemes approximate the derivative in different ways - usually by expanding the derivative function in a Taylor series, about the current state.

As an example, consider the Runge-Kutta integrations. The derivative function $f(x)$ is evaluated four times and saved as the v_i . The first is the slope at the beginning of the interval:

$$v_1 = f(x_n)$$

This slope is extrapolated half way across the interval and a new value calculated:

$$v_2 = f(x_n + 0.5 hv_1)$$

This new slope is brought back to the beginning of the interval and again used to extrapolate half way across the interval and a new slope evaluated

$$v_3 = f(x_n + 0.5 hv_2)$$

This third slope is brought back to the beginning of the interval and used to extrapolate all the way across and a new slope evaluated there

$$v_4 = f(x_n + hv_3)$$

We now have four slopes - one an exact one at the beginning of the interval and approximations - two in the middle and one at the end. Runge-Kutta says that the average slope to be used in updating the state is the

weighted average of these - assigning twice the weight to the slopes in the center of the interval as the ends, i.e.,

$$f(x_{n+a}) = (v_1 + 2v_2 + 2v_3 + v_4)/6$$

Errors introduced at each step will normally be proportional to h^5 for this method.

The form of the integration statement is either

$$\text{state} = \text{INTEG}(\text{deriv}, \text{ic})$$

or embedded in any legal expression as a function that has a single output, the value of the state. When the integration statement is alone, then the state name can be identified - embedding in an expression means a generated variable must be used for the state.

'state' = a simple variable or subscripted array name with a single integer CONSTANT subscript.*

'deriv' = an arithmetic expression of arbitrary complexity, i.e., may contain further INTEG statements, functions or MACRO's

'ic' = a simple non-subscripted variable, a real constant or a general expression enclosed in parentheses. If it is a simple variable (preferred), then this variable name must not be used as another initial condition, state, derivative or system variable name.

If the expression form is chosen the statement is sorted in such a way that all components in the expression are evaluated before the state value is assigned. Equations using the state will then follow. Problems may occur when using this form with the REINIT command (q.v.) at model execution time. The expression will always be evaluated and substituted for an initial condition established by the reinitialization operation.

Examples of use are as follows:

```
ARRAY X(5)
Y = INTEG(5.0*X + C, YIC)
X(1) = INTEG(A + B, X1IC)
X(2) = INTEG(INTEG(4.0*DD + X(1), 0.0), X2IC)
Z = P*INTEG(ZDOT, 0.0) + BT
W = INTEG(WDOT, (2.0*AL*SIN(TH)))
```

The following comments are to be noted.

- 1) That Z, in the above, is not a state since the INTEG function appears embedded in an expression.
- 2) Once the array X has been defined as a state by the single INTEG statement, another assignment statement cannot be used to fill the other slots of an array. If one element of an array is a state, ALL elements of the array must be states; i.e.,

```
X(3) = W + 5.0*Y
X(4) = AL*INTEG(ALP, 0.0) + W
```

are both illegal when accompanying the above.

Initial conditions can only be names not used as states, other initial conditions or system variables; i.e.,

```
YY = INTEG(YYD, YIC)
ZZ = INTEG(Z*4.0, X)
ZK = INTEG(KK, CINT)
```

* See the vector integration operator INTVC for another way to solve matrix differential equations.

are all illegal when used with the above cards. CINT could be used if the system default name has been changed by a CINTERVAL statement (q.v.).

The reason for the rules may be better understood if the actual translation operation is explained. The translator is building up three lists of names that will be placed in sequence in a FORTRAN common block. The state list will contain all the state variables and arrays, the derivative list will contain names that have values for the derivatives stored into them and the initial condition list will have names with values corresponding to the equivalent slot in the state list. The problem comes about that each name in these lists must be unique. The initial condition name for each state must be different. If it is a constant or an expression, a generated name is used (Znnnnn) but if it is a simple variable the translator uses that name directly. The expression form is changed into a test (IF statement) on the first evaluation of a run, in which case the expression is evaluated and the result stored into the state. The derivatives at present are all generated variables to avoid conflict since it was thought that integrating a state was too common a feature to eliminate. The statements

```
YD = INTEG(5.0*X, 1.0)
Y = INTEG(YD, YIC)
```

will generate the following assignment statements

```
Z09999 = 5.0*X
Z09998 = YD
```

and a DATA statement for the generated initial condition

```
DATA Z09997/1.0/
```

Now the state list will be YD,Y; the derivative list will be Z09999, Z09998; and the initial condition list will be Z09997,YIC. Each name on the list is unique.

From this we can show that it saves time to include the derivative expressions in the INTEG statement rather than calculating the derivative by name explicitly. Of course, if the derivative is needed in other calculation sequences then it must be given a name, but this would be an unusual case, i.e.,

```
YD = 5.0*X + YY
Y = INTEG(YD, YIC)
```

would be better done by

```
Y = INTEG(5.0*X + YY, YIC)
```

unless the value of the expression $5.0*X + YY$ is needed elsewhere in the program. The extra assignment statement is avoided. However, see the use of INTVC below for single elements - pseudo arrays of size one.

4.55 INTEGER

See type statements.

4.56 INTERVAL

In order to schedule repeated execution of a DISCRETE (q.v.) block, the INTERVAL statement is used both to define the name of the variable controlling the repetition period and its initial value. Standard form for the statement is

```
INTERVAL name = real constant
```

where 'name' is a simple unsubscripted variable name. There is no default and a DISCRETE section without an INTERVAL statement will never be executed.

The mechanization of the INTERVAL feature is to use the slot in the global MINTERVAL array corresponding to the DISCRETE block and equivalence the INTERVAL variable into this array. If no INTERVAL statement is placed within the DISCRETE block, a value of zero is used as the default which flags the DISCRETE block as not to be executed during the initialization phase.

4.57 INTVC

The restrictions on the INTEG operator with regard to initial condition and derivative arrays can be avoided by using this vector integrator operator. Standard form is

$$x = \text{INTVC}(xd, xic)$$

where x, xd and xic are arrays of the same size and correspond to state, derivative and initial condition respectively. See the program in Section 10 of Appendix A for examples of the use of the INTVC operator.

The restrictions on the use of the INTVC operator are as follows:

- 1) INTVC cannot be used in an expression.
- 2) The derivative array must not appear anywhere else as a state. (If you must use a state as a derivative - velocity, for instance - use the block transfer (XFERB) subroutine to move it into another array before using INTVC.
- 3) The initial condition cannot be a constant or expression.
- 4) The array size may be one - or equivalently a single undimensioned variable can be used instead. In this case, the derivative name is used explicitly and no extra assignment statement is generated.
- 5) The derivative array cannot be preset with a CONSTANT statement since it is cleared automatically to zero before the derivative evaluation routine is called the first time - after the INITIAL section - after every START.

As an example consider

```
ARRAY X(10), XD(10), XIC(10), M(5, 5), MD(5, 5), MIC(5, 5)
```

```
...
```

```
X = INTVC(XD, XIC) $ 'VECTOR INTEGRATION'
```

```
M = INTVC(MD, MIC) $ 'MATRIX INTEGRATION'
```

If R, V and A are range, velocity and acceleration vectors so

```
ARRAY R(3), V(3), A(3)
```

the following is ILLEGAL -

```
V = INTVC(A, VIC)
```

```
R = INTVC(V, RIC)
```

since these statements ask for V to be considered as both a state and a derivative at the same time. Using the XFERB (transfer block) subroutine and defining an RD (R dot) array, the sequence becomes

```
V = INTVC(A, VIC)
```

```
CALL XFERB (RD = V, 3)
```

```
R = INTVC(RD, RIC)
```

4.58 I/O STATEMENTS

FORTRAN read, write and file handling statements can be included and are recognized by the ACSL translator but a minimum of syntax checking is performed. Most errors will be indicated at the FORTRAN compilation that follows translation.

The following definitions apply to all the I/O statements covered:

i = logical I/O number that determines the file. The logical unit numbers available depend on machine type and operating system. See addendum.

i = 5 being the INPUT file or card reader

= 6 being the OUTPUT or print file

= 9 being the PRINT file - must be disposed of explicitly

n = FORMAT declaration identifier which must be a statement label (number or symbol)

L = input/output list. This list portion of an input/output statement indicates the data items and the order, from left to right, of transmission. The input/output list can contain any number of elements. List items may be array names, simple or subscripted variables, or an implied DO-loop. Items are separated by commas, and their order must correspond to any FORMAT specifications associated with the list. External records are always read or written until the list is satisfied. The ACSL translator does not check the syntax of the list - any character string will be accepted and errors will be indicated by the subsequent FORTRAN compilation. For more details on list format, refer to a FORTRAN reference manual.

Standard form of the I/O statements are as follows:

PRINT *n*, *L* - Information in the list *L* is transferred to the OUTPUT unit-printer or remote terminal - in accordance with the FORMAT declaration, *n*. *N.B.* Use LINES (q.v.) before the PRINT or WRITE to tell the executive how many lines are being written. Top-of-form is then handled automatically.

WRITE(*i*, *n*)*L* - Same as PRINT above except the file is determined by the value of the unit *i*. Valid logical unit numbers depend on machine type and operating system. See addendum.

READ *n*, *L* - One or more card images are read from the standard INPUT unit - card reader or remote terminal. Information is converted from left to right in accordance with the FORMAT specification, *n*, and is stored in the locations named in the list, *L*.

READ(*i*, *n*)*L* - Same as READ above, except the file is determined by the value of the unit *i*. Valid logical unit numbers depend on machine type and operating system. See addendum.

4.59 ISIGN

Append a sign by

$$n = \text{ISIGN}(j_1, j_2)$$

where j_1 and j_2 are integer variables or expressions. Result is sign of j_2 times absolute value of j_1 and the result will be an integer.

NOTE: ISIGN should be declared to be of type INTEGER prior to its use.

4.60 LEDLAG

A lead-lag compensator may be conveniently implemented by using the standard form

$$y = \text{LEDLAG}(p, q, x, ic)$$

$$\text{LEDLAG}(y = p, q, x, ic)$$

Result: *y* will be related to input *x* through the transfer function

$$\frac{y}{x} = \frac{ps + 1}{qs + 1}$$

$$y(0) = \frac{p}{q} x(0) + ic$$

The same restriction on *ic* is present as for the INTEG operator. *p* and *q* may be expressions of arbitrary complexity. Figure 4-8 shows the mechanization of this operator as a system macro.

```

MACRO LEDLAG(OUT,P,Q,IN,IC)
MACRO STANDVAL IC=0.0
MACRO REDEFINE X,Y
X=IN
OUT=Y+(P)*X/(Q)
Y=INTEG((X-OUT)/(Q),IC)
MACRO END

```

Figure 4-8. Listing of LEDLAG Operator Macro

4.61 LIMINT

Integrators should not be limited using the BOUND function since the integrator will continue to integrate and will thus require a time to integrate out of the limit when the derivative changes sign. The LIMINT operator holds the integrator at the limit as long as the derivative is of such a sign to drive it further into limit.* When the derivative reverses sign, the integrator will immediately come off limit. The operator has the standard form

$$y = \text{LIMINT}(yd, ic, ll, ul)$$

$$\text{LIMINT}(y = yd, ic, ll, ul)$$

where:

yd = an expression for the derivative

ic = *y*(0) - same restriction on *ic* as on the INTEG statement; may be omitted if zero

ll = lower limit on *y*

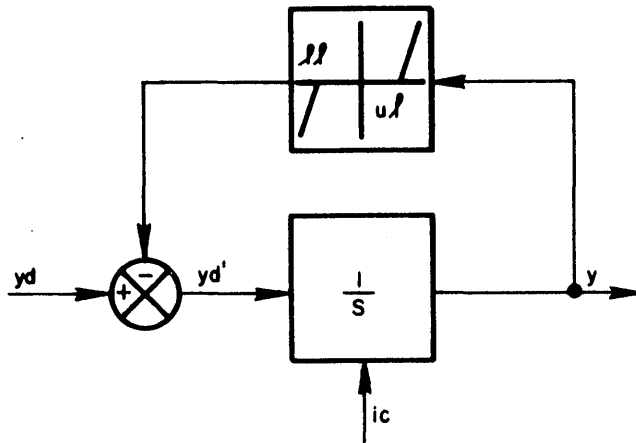
ul = upper limit on *y*

Figure 4-9 shows the effect of the LIMINT operator.

4.62 LINES

Used to tell the executive processor how many lines are going to be written with PRINT or WRITE statements. Pass by this if you do not use FORTRAN FORMAT statements. The executive processor at run-time keeps track of the number of lines written on the output file and every 55 lines issues a top-of-form and prints out the header and TITLE. If FORTRAN write statements are included in the model definition they are expected to inform the processor how many lines they are going to write so that the pagination is performed correctly. Standard forms of the call are

* See DBLINT operator for double integration, acceleration to displacement with limited displacement.



yd' IS MADE ZERO WHEN y EXCEEDS BOUNDS

Figure 4-9. Effect of LIMINT Operator

CALL LINES (n)

where n is the number of lines about to be printed. If n lines are not available on the current page, the page will be ejected. You should not issue your own top-of-form by making Column 1 a digit 1.

CALL LINES(2)

PRINT99, A, B, C

99 . . FORMAT(1X, 2E12.5/1X, F12.2)

In order to force a top-of-form prior to the write, call subroutine PAGE(q,v) before calling LINES.

4.63 LOG

The data logging operation for the OUTPUT and PREPAR lists can be forced by a call to this subroutine. Data logging only takes place after each pass through the DYNAMIC section and on exit from the TERMINAL section. Finer detail can be recorded by judicious use of this operation. An example of use is as follows:

```
LOGICAL HIRATE
IF(HIRATE) CALL LOG
```

which will force the data logging action whenever logical variable HIRATE is true.

4.64 LOGICAL

See type statements.

4.65 LSW, RSW

Logical (or integer) and real switch functions. The standard form of these two operators is

$$i = \text{LSW}(p, j_1, j_2)$$

$$y = \text{RSW}(p, x_1, x_2)$$

The functions take on the value of the second argument, j_1 or x_1 , when the logical expression p has the value .TRUE., otherwise the value of the third argument, j_2 or x_2 . The j_1 and j_2 are any integer or logical expressions; x_1 and x_2 are real expressions.

NOTE: LSW should be declared to be of type INTEGER prior to its use.

4.66 MACRO

Denote the beginning of a MACRO definition if not already within a definition. Within a definition denotes special subcommands to be interpreted by the MACRO processor. For a full description of the MACRO capability see Section 6.

4.67 MAX0

Maximum of a string of integer arguments can be obtained from

$$n = \text{MAX0}(j_1, j_2, \dots, j_n)$$

where the j_i are integer variables or expressions; any number of arguments may be included.

NOTE: MAX0 should be declared of type INTEGER prior to its use.

4.68 MAX1

Maximum of a string of real arguments can be found from the function

$$n = \text{MAX1}(x_1, x_2, \dots, x_n)$$

where the x_i are real, floating point variables or expressions. Any number may be included. Result will be the integerized value of the largest x_i , where the integerized value is the sign of the x_i times the largest integer $\leq |x_i|$

NOTE: MAX1 should be declared to be of type INTEGER prior to its use.

4.69 MAXTERVAL

See MINTERVAL.

4.70 MERROR, XERROR

Relative and absolute error bounds - per step - for individual state variables. They are written in the standard form -

$$\text{MERROR } v_1 = \text{real constant}, v_2 = \text{real constant}, \text{ etc.}$$

$$\text{XERROR } v_1 = \text{real constant}, v_2 = \text{real constant}, \text{ etc.}$$

where the v_i are unsubscripted variable names that are state variables, i.e., appear opposite an INTVC or INTEG statement that is not embedded in an expression. The state itself can be an array, but then the errors specified will apply to all elements in the array. Individual elements cannot be given separate error tolerances.

MERROR is used for relative (fractional) errors and

XERROR for absolute errors.

If any relative or absolute errors are specified, the FIRST specification encountered will be applied to all integrators that are unspecified. If no values are specified, the relative and absolute errors will both be set to 1.0E-4.

Example:

MERROR X = 1.0E-4, Y = 1.0E-6

where:

X = INTEG(XD, XIC)

Y = INTEG(YD, YIC)

Z = 5.0*INTEG(ZZZ, 0.0) + COS(X)

Note Z in the above example cannot have any error bounds specified since it is not a state. The actual state will be a generated variable which will be placed in the assignment statement instead of the INTEG function.

The MERROR and XERROR statements have meaning only for the variable step-size integration algorithms and are used to bound the error introduced at each step. Based on the maximum absolute value of a state variable $|V_i|_{\max}$ since the start of the run*, the allowable error bound is defined to be

$$E_i = \text{MAX}(X_i, M_i|v_i|_{\max})$$

If any of the predicted errors in a step of the integration program are greater than the corresponding E_i , the calculation interval or step size is reduced appropriately. If the error is still too large after the step size has been reduced to the minimum, the ERRTAG variable, if any, is set .TRUE. and the run aborted.

4.71 MINO

Minimum of a string of integer arguments can be found from the function

$$n = \text{MINO}(j_1, j_2, \dots, j_n)$$

where the j_i are integer variables or expressions. Any number of arguments may be included.

NOTE: MINO should be declared to be of type INTEGER prior to its use.

4.72 MIN1

Integerized minimum of a string of real arguments can be found from the function

$$n = \text{MIN1}(x_1, x_2, \dots, x_n)$$

Arguments will be the same as for MAX1.

NOTE: MIN1 should be declared of type INTEGER prior to its use.

4.73 MINTERVAL, MAXTERVAL

The minimum and maximum calculation intervals (integration step size) can be controlled and renamed using these statements. If a variable step size algorithm attempts to go below the value of the minimum step size, the error tag will be set. If it is already set, the termination flag will be set. Standard form for the statements are

MINTERVAL name = real constant

MAXTERVAL name = real constant

where 'name' is a simple unsubscripted variable name. Default names for these variables are MINT and MAXT so why not use them? They can be set by assignment statements in the program or by SET commands at run-time. Example:

MINTERVAL MINT = 1.0E-6

* Use of CONTIN at run-time defines the start of a new run in this sense since the integration routine has to be reinitialized.

then at run-time

SET MINT = 2.0E-6

MAXTERVAL may be used to control the step size independently of the communication interval changes if NSTEPS (q.v.) is made 1. The calculation interval (step size) is started off at the CINT/NSTP and this is then forced to lie in the region MINT to MAXT.

4.74 MOD

Modulus, or remainder, of an integer divided by an integer can be obtained by

$$n = \text{MOD}(j_1, j_2)$$

where j_1 and j_2 are integer constants, variables or expressions.

Result is remainder of j_1 divided by j_2 . When j_2 is negative, this isn't quite correct. What actually happens is that the result is $j_1 - [j_1/j_2] j_2$ where $[]$ is an integer with magnitude of not more than the argument and with the same sign.

NOTE: MOD should be declared to be type INTEGER since the default for all variables is type REAL.

4.75 MODINT

Moded integrator allows operation in reset and hold mode as well as normal operate. The mode of the integrator is determined by the two flags in the call. Standard form is

$$Y = \text{MODINT}(y_d, ic, l1, l2)$$

$$\text{MODINT}(y = y_d, ic, l1, l2)$$

where:

y_d is the derivative variable or expression.

ic is the initial condition - see INTEG for restrictions on initial conditions.

$l1$ and $l2$ are logical variables or expressions of arbitrary complexity denoting the mode.

The truth table is

$l1$	$l2$	Mode
T	F	reset
F	F	operate
T	T	operate
F	T	hold

where

T = .TRUE. and F = .FALSE.

Figure 4-10 shows the mechanization of this operator as a system macro.

```

MACRO MODINT(Y, D, IC, LCV1, LCV2)
MACRO RELABEL B
MACRO REDEFINE ICTEM, STATE, STORE, MODED
LOGICAL MODED
CONSTANT STORE=0.
PROCEDURAL(Y, MODED=STATE, IC, LCV1, LCV2)
CALL ZZICS(ICTEM=STORE)
IF(ZZFST(STATE).LT.0.5)GOTO B
MODED=.FALSE.
IF((LCV1).AND.(LCV2).OR..NOT.((LCV1).OR.(LCV2)))GOTO B
MODED=.TRUE.
IF(LCV1)ICTEM=IC-STATE
B. .Y=STATE+ICTEM
END
STATE=INTEG(ZZRSW(MODED, 0. , D), IC)
MACRO END

```

Figure 4-10. Listing of MODINT (Moded Integrator) Operator Macro

4.76 NSTEPS

Defines the calculation interval - integration step size - in terms of the communication interval. It renames the integer variable that defines the number of integration steps in a communication interval. Standard form of the statement is

NSTEPS name = integer constant

where name is a simple unsubscripted variable. The default name is NSTP and is normally given a value of 10. This means that the integration step will be one-tenth the communication interval.

The variable named in the NSTEPS statement is automatically typed INTEGER. The name defined in the NSTEPS statement (or NSTP by default) can be set by assignment statements within the program itself.

The maximum and minimum step size values MAXT and MINT take precedence over the step size arrived at by dividing the communication interval by NSTP, i.e., the step size H is given by

$$H = \text{MAX}(\text{MINT}, \text{MIN}(\text{MAXT}, \text{CINT}/\text{NSTP}))$$

It is recommended that the value of NSTP be made unity so that the integration step size can be controlled by the value of MAXT, specified by a knowledge of the plant dynamics i.e. include the statement NSTEPS NSTP = 1. If CINT/NSTP is controlling, then a change in observation interval CINT will require a corresponding change to NSTP to keep the same step size. Having to change two variables for one result is normally considered bad programming practice.

4.77 OU

Band limited white noise can be implemented by this function. A normal random number generator (GAUSS) will deliver a fixed total power - rms value - but the frequency spread will depend on the current

calculation interval. Usually, the low frequency power density is important so casual use of GAUSS into a low pass filter will lead to ill-defined variation of this quantity as the step size is changed. The Ornstein-Uhlenbeck process maintains a constant source of power over a specified frequency band, so eliminating any problem of having to include the current calculation interval in the standard deviation of the random variable. Standard forms are

$$y = OU(\tau_a, m, s)$$

$$OU(y = \tau_a, m, s)$$

where

- τ_a is the low-pass filter time constant. Break frequency is $1/2 \pi \tau_a$ Hz
- m is the mean value of Y
- s is the standard deviation of Y (rms value)

The operator is implemented by generating a correlated noise sequence from the general formula

$$n_{i+1} = n_i e^{\frac{-\Delta t}{\tau}} + \omega_i$$

where

- τ is the correlation time constant
- Δt is the sample interval
- ω_i is a gaussian random variable

We would like to find the ω_i that will produce the correct noise power or such that

$$\overline{n^2} = \sigma^2$$

Square the above equation for n_{i+1} and take expected values

$$\overline{n_{i+1}^2} = \overline{n_i^2} e^{\frac{-2\Delta t}{\tau}} + \overline{\omega_i^2}$$

since we can assume the random drive uncorrelated with the noise sequence. i.e.

$$\overline{n_i \omega_i} = 0$$

But $\overline{n_i^2} = \overline{n_{i+1}^2} = \sigma^2$

so
$$\overline{\omega_i^2} = \sigma^2 (1 - e^{\frac{-2\Delta t}{\tau}})$$

Now the sequence can be expressed by

$$n_{i+1} = n_i e^{\frac{-\Delta t}{\tau}} + \sigma \sqrt{(1 - e^{\frac{-2\Delta t}{\tau}})} g_i$$

where g_i is a gaussian random variable of zero mean and unit variance.

4.78 OUTPUT, PREPAR

Variables to be recorded at each communication interval can be specified by these two statements. In general, these statements should be viewed as part of the exercising of the model and included in the run-time commands. It is realized, however, that standard lists may be defined and, in that case, may be included within the model definition part of the program. The statements are accomplished by system MACRO's so the arguments must be enclosed in parentheses for the model definition. The corresponding run-time commands do not have parentheses. Standard forms are

OUTPUT ($v_1, v_2, v_3, \text{etc.}$)

PREPAR ($v_1, v_2, v_3, \text{etc.}$)

where v_i are either simple variables or array names. If the v_i are array names, the entire array will be reported. In this form it is not possible to select array elements.

OUTPUT designates the variables that will have their values recorded on the print file as the run progresses at each communication interval. PREPAR records the values on a save file that is used by any subsequent PLOT or PRINT commands. The OUTPUT list cannot be in columns, since each communication interval the entire block of output variables must be printed. Once the data has been saved, it can be put into columns by repeated passes down the file.

N.B. The OUTPUT and PREPAR statements in the model *definition* wastes core space since the names must be preset into a data array and then transferred once at run-time to the lists used by the executive. Only one OUTPUT and PREPAR statement is significant. If more than one is used the first one will be effective.

Any run-time OUTPUT or PREPAR statements will override these statements embedded in the model definition.

For an explicit program (Subsection 3.1) they should be placed in the INITIAL section. They must *not* be placed in the TERMINAL section.

These statements are translated to a subroutine call that passes the list of names given. The appropriate list (OUTPUT or PREPAR) is checked and if the length is currently zero, the list of names is added. If the length non-zero, no action is performed. By this means, build up is avoided in loops and the run-time specification will override the model definition specification.

4.79 PAGE

In order to control pagination, subroutine PAGE can be used to force a top-of-form and turn on or suppress auto page eject. As a by product, the number of lines left on the current page is returned as the second argument. The format of the call is

CALL PAGE (K, *nll*) where K is a code to modify the page eject as follows:

= -2, suppress auto page eject, eject next write

= -1, suppress auto page eject

= 0, ignore but return 'NLL'

= 1, turn on auto page eject

= 2, turn on auto page eject, eject next write

nll is returned as the number of lines left on the current page. In order to force a new page before writing ten lines via a formatted write statement use

CALL PAGE (2, NLL)

CALL LINES (10)

WRITE (6, 99) . . .

The ACSL system normally operates in an internal auto page eject mode. The exception is the printer PLOT command when plots can span page boundaries. Auto page eject is restored after each PLOT.

4.80 PREPAR

See OUTPUT.

4.81 PRINT

See I/O statements.

4.82 PROCEDURAL

Denotes the beginning of a block of PROCEDURAL code that will finish with a matching END. The code within the block will be executed in the sequence given and no attempt will be made to reorder it. Form of statement

PROCEDURAL (output list = input list)

where:

output list = variable, variable, variable . . .

input list = expression, expression, . . .

The output list should contain only nonsubscripted variable names. They may belong to arrays however, in which case the entire array must be filled within the block. The sort algorithm requires that values only appear as outputs once - otherwise, the error message 'multiply defined output symbol' is issued.

So

A(1)= X + Y

A(2)= Y + Z

flags the variable A as having a value placed in it twice and so illegal. An acceptable form would embed this sequence in a PROCEDURAL block.

PROCEDURAL(A = X, Y, Z)

A(1)= X + Y

A(2)= Y + Z

END

which tells the sort routine to place the block before a reference to any element of the array A.

PROCEDURAL . . . END brackets need only be used in a DERIVATIVE section where the code is sorted. Remember PROGRAM . . . END alone is an implicit DERIVATIVE section. Code placement in the INITIAL, DYNAMIC, or TERMINAL sections is unsorted so although PROCEDURAL . . . END may be used, it will have no effect.

Within a DERIVATIVE section, the entire block is moved so that it is placed after the calculation of all variables on the input list and before the use of any variables on the output list. Although other variables may be present on the list, sorting is only with respect to variables calculated within the same DERIVATIVE section. The sort operation never moves code across section boundaries. The entire DERIVATIVE section can be made a PROCEDURAL block and since it then can't be moved, no input or output argument list is necessary. PROCEDURAL . . . END brackets should always be used around code constructs in which the order must not be changed. Typical operators are DO, GOTO and IF (three way or single branch).

Expert users may lie in specifying what is on the input/output lists. The ACSL translator never looks inside the PROCEDURAL block to ensure compliance with the list supplied, so it is possible to break algebraic loops for approximate solution by omitting one of the loop variables from the PROCEDURAL input list. Since this variable will be used before it's calculated, initialization is required in the INITIAL section. In breaking implicit loops this way, the last value of the variable will be used which is satisfactory in a large number of cases. Be warned however that the results will change with step size and a variable phase shift will be present in traversing the implicit block.

NOTE: Statements involving memory operators which must be sorted into correct execution order should not be included in a PROCEDURAL block. These are ZHOLD (zero order hold), DELAY, DBLINT, DERIVT, (the derivative operator) and for other reasons IMPLICIT.

4.83 PROGRAM

The first card in the model definition section must be this card; it has the form

PROGRAM any character string except a dollar sign

The character string is not used in any way and serves merely to identify the deck and listing. This card must be accompanied by a matching END statement to terminate the model definition section.

4.84 PTR

A resolver - polar to rectangular - can be implemented by the standard form

PTR(X, y = r, th)

Result is

$x = R \cdot \cos(\text{th})$

$y = R \cdot \sin(\text{th})$

where the angle, th, must be expressed in radians. Figure 4-11 lists the mechanization of this operator as a system macro.

```
MACRO PTR(X1, X2, R, TH)
X1=(R)*COS(TH)
X2=(R)*SIN(TH)
MACRO END
```

Figure 4-11. Listing of PTR (Polar to Rectangular) Operator Macro

This form is not a functional representation since there are two outputs. Thus, this statement cannot be embedded in an expression. It can only stand alone as shown.

4.85 PULSE

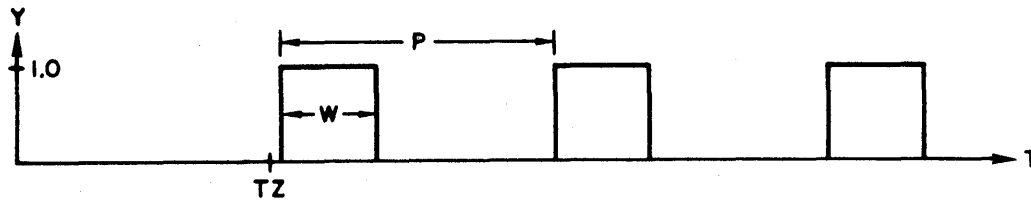
A train of pulses can be generated using the PULSE function. The independent variable, default T, is used to drive it. Note that the integration step size may affect the answers in that too large a step could cause

the pulse to stay on indefinitely. The output will always be turned on (=1.0) at the beginning of the first calculation interval that follows the exact turn on time. Standard form is

$$y = \text{PULSE}(tz, p, w)$$

Result

Y is a pulse train (0.0 or 1.0) starting at the first calculation interval that equals or exceeds tz. Period is p and width is w.

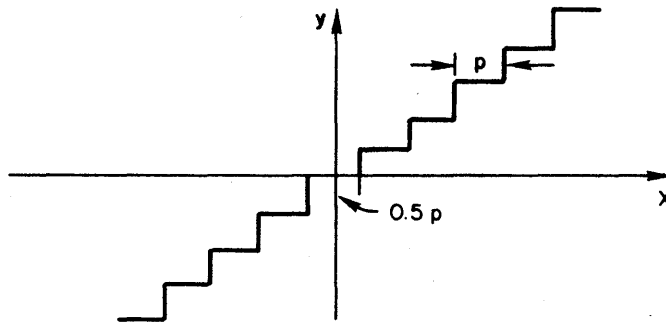


4.86 QNTZR

A variable may be quantized so that only discrete values are used. It is a zero centered system as shown in the diagram. Standard form

$$y = \text{QNTZR}(p, x)$$

Result:



where x and p are real variables or expressions.

4.87 RAMP

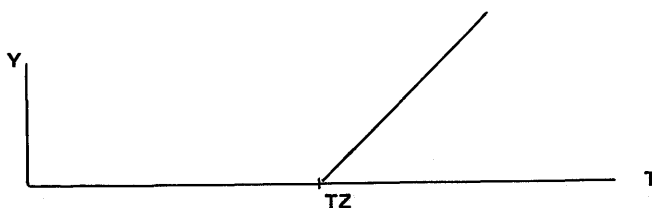
The RAMP function generates a linear ramp of unit slope, starting at a specified time. It is another way of applying a dead zone (DEAD q.v.) to the independent variable. Standard form is:

$$y = \text{RAMP}(tz)$$

Result is:

$$y = 0.0 \quad T < tz$$

$$y = T - tz, \quad T \geq tz$$



RAMP starts at first calculation interval that equals or exceeds tz.

4.88 READ

See I/O statements.

4.89 REAL

See type statements.

4.90 REALPL

A first order lag may be implemented by the standard form

$$y = \text{REALPL}(p, x, ic)$$

$$\text{REALPL}(y = p, x, ic)$$

Result:

y will be related to input x through the transfer function

$$\frac{y}{x} = \frac{1}{ps + 1}$$

$$Y(0) = ic$$

The same restrictions on ic are present as for the INTEG operator; if zero, it may be omitted. P may be an expression of arbitrary complexity. Figure 4-12 lists the macro to implement this operator.

```
MACRO REALPL(Y,P,X,IC)
MACRO STANDVAL IC=0.0
Y=INTEG((X-(Y))/P),IC)
MACRO END
```

Figure 4-12. Listing of REALPL Operator System Macro

4.91 RESET

A special operator is provided for use in the INITIAL section that can initialize the state variables and optionally perform intermediate calculations. Normally when entering the INITIAL region, the state variable names are undefined - it is only on exit from the INITIAL region to the DYNAMIC region that the initialization operation is performed. Remember the prime objective of the INITIAL region is to calculate the unknown initial conditions. Two forms of the call are possible so

RESET ('EVAL')

RESET ('NOEVAL')

where the argument explicitly says whether or not a complete derivative evaluation is to be attempted. It is the user's responsibility to ensure that unknown initial conditions have "reasonable" values to prevent arithmetic errors (divisions by zero say). An example would be calculating the initial conditions to be placed on the

accelerometer filters in a missile simulation. In order to obtain the nominal acceleration, the missile velocity vector must be rotated into the missile axes, the angle of attack determined and the aerodata looked up to obtain force coefficients. This code must be expressed in terms of the initial condition variables rather than the state variables unless the RESET operator is used.

This extra code can be avoided by

RESET ('EVAL')

at the beginning of the INITIAL region since the derivative subroutine would calculate body acceleration using the state variable names. The problem is that the process is not selective - calculations of all state variable derivatives is attempted. In the above example the output of the accelerometer filter must lead somewhere and if initialized indefinite will lead to an arithmetic error. If the undefined initial conditions are preset in a CONSTANT statement (0.5 is a useful default number) then the calculation can proceed and the meaningless numbers can be disregarded. The important thing is that all the calculations can proceed without arithmetic errors.

The operator is defined in terms of a state vector S, and the initial condition vector IC. The state derivative vector is given by

$$\dot{S} = f(S, T)$$

Now the RESET(a) action is

S ← IC, T ← XICITG

if (a=EVAL) evaluate f(S, T)

4.92 RSW

See LSW (logical switch).

4.93 RTP

A resolver - rectangular to polar - can be implemented by the standard form

RTP(r, th = x, y)

Result:

$$r = \sqrt{x^2 + y^2}$$

th = ATAN2(y, x)

The angle, th, will be in radians and will cover the range - π to + π depending on the magnitude and signs of x and y. Figure 4-13 shows the mechanization of this operator as a system macro. Note the second argument of the arc-tangent (ATAN2) is modified by the addition of a very small amount. Inputs of 0.0, 0.0 will return an angle of zero instead of indefinite.

```

MACRO RTP(R, TH, X1, X2)
R=SQRT((X1)**2+(X2)**2)
TH=ATAN2(X2, X1+1.0E-30)
MACRO END

```

Figure 4-13. Listing of RTP (Rectangular to Polar) Operator Macro

This form is not a functional representation since there are two outputs. Thus, this statement cannot be embedded in an expression. It can only be used as stand alone as shown.

4.94 SAVE

The current MACRO tables containing the MACRO names and packed definitions are written - in binary - on the ACSL System Macro File. This operation allows each user to maintain his own file of MACRO's separate from the system file.

The normal operation of the system is to read into the MACRO definition tables, the contents of this Macro File before the translation begins. If it does not exist, no MACRO definitions are present.

Action of SAVE is to write the current contents of the MACRO tables back on this file, thus destroying the original contents. To use SAVE you must have WRITE permission on the current MACRO file.

4.95 SCALE

Rounds the given maximum and minimum values so that they are suitable for plotting. Standard form of the call is

SCALE(smn, smx = ymn, ymx)

where:

ymn and ymx are the minimum and maximum values

smn and smx are the scaled minimum and maximum to be used on a plot (i.e., rounded to multiples of 1, 2, 4, or 10)

The first two arguments need not be distinct from the second, i.e.,

SCALE(ymn, ymx = ymn, ymx)

replaces actual minimum and maximum values with the rounded ones.

Normally, this operator should be used in the TERMINAL section of an explicit program to establish scale factors for subsequent PLOT (q.v.) commands. It is only used when *two* or more plots need the same, originally unknown, scale factors.

4.96 SIGN

Append a sign by

$y = \text{SIGN}(x_1, x_2)$

where:

x_1 and x_2 are real, floating point constants, variables or expressions.

Result is the sign of x_2 time the absolute value of x_1 ;

To multiply by the SGN of a variable where

$\text{SGN}(x) = +1.0 \quad x \geq 0.0$

$\text{SGN}(x) = -1.0 \quad x < 0.0$

the SIGN function can be used so

$y = X * \text{SIGN}(1.0, Z)$

which will give

$y = X * \text{SGN}(Z)$

NOTE: The result of SIGN(X,Z) is not the same thing as X*SIGN(1.0,Z).

4.97 SIN

Takes the sine of a real argument which must be in radians

$$y = \text{SIN}(x)$$

Result will be such that $-1.0 \leq y \leq 1.0$

4.98 SQRT

Take the square root of a positive real argument x.

$$y = \text{SQRT}(x)$$

4.99 STEP

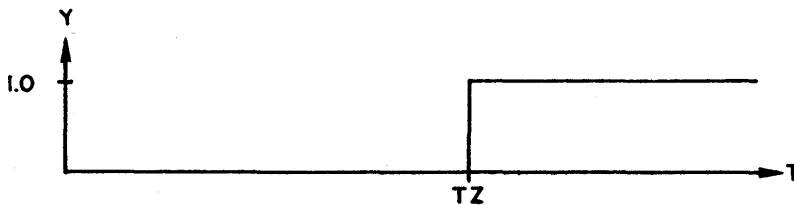
The STEP function produces a change from zero to one in the output at a specified value of the independent variable. Standard form is

$$y = \text{STEP}(tz)$$

Result is

$$y = 0.0, T < tz$$

$$y = 1.0, T \geq tz$$



Pulse starts at first calculation interval that equals or exceeds tz.

4.100 TABLE

Used to describe an arbitrary function of one, two or three variables. A separate TABLE statement must be used to define each function. The standard form is:

TABLE name, n, dimension(s)/data list/

where 'name' = name of the function. The value will be accessed by name (arg1), name (arg1, arg2) or name (arg1, arg2, arg3) for functions of one, two or three variables respectively. The arguments in the above are arithmetic expressions, hence can contain an arbitrary level of complexity.

'n' = an unsigned integer constant giving the number of independent variables; must be 1, 2 or 3.

'dimension(s)' = unsigned integer constants; the number of constants must correspond to the value for n. The value of the constants give the number of discrete data points for each successive independent variable. A dimension of one is illegal.

'data list' = real constants: First comes the list of independent variable values. The number of these points should equal the *sum* of the dimensions. Then, the data points of the function with the first argument varying fastest. The number of function data points must equal the *product* of the dimensions.

All data points for the independent variables (break points) must be of monotonically increasing order - values may be identical but a breakpoint must never be less than a preceding value. Each TABLE statement may contain as many data points as desired. Once the function has been defined it may be referenced just like any other ACSL function. Repeat counts may be used in the data specification.

Examples of use in tables of one, two and three variables. Breakpoints are 2 for arg a, 3 for arg b and 4 for arg c.

```
TABLE F1ARG, 1, 2/0.0, 1.0, f1, f2/
```

```
TABLE F2ARG, 2, 2, 3/0.0, 1.0, - 1.0, 0.0, + 1.0 . . .
```

```
    f11, f21, f12, f22, f13, f23/
```

```
TABLE F3ARG, 3, 2, 3, 4/0.0, 1.0, - 1.0, 0.0, + 1.0, 0.0, 0.1, 0.2, 0.3 . . .
```

```
    ,f111, f211, f121, f221, f131, f231 . . .
```

```
    ,f112, f212, f122, f222, f132, f232 . . .
```

```
    ,f113, f213, f123, f223, f133, f233 . . .
```

```
    ,f114, f214, f124, f224, f134, f234/
```

For all function generation routines, if the calculated values of the independent variables lie outside the range specified by the TABLE statement, the values for the function will be obtained by extrapolating from the last values given.

The table operator makes up a MACRO of the same name as the function so that all references after the table definition are caught. An array is also defined with the same name and enough storage to contain both function data values and the corresponding argument breakpoint values. This array name is entered into the dictionary and it may be accessed in normal fashion by SET, DISPLY etc. commands. One point to note is the order of data entry into the array - the function data is listed first, then the breakpoint values follow. This order is the opposite from that listed in the TABLE statement. It was felt that the more normal operation at run time was changing function values rather than breakpoints.

Consider the pitching moment table as a function of Mach number

```
TABLE CM, 1, 5 . . .
```

```
    /0.0, 0.8, 1.2, 1.5, 2.5 . . .
```

```
    ,0.50, 0.51, 0.92, 0.83, 0.15/
```

This will make up an array CM(10) - the first five words will contain the function values 0.50, 0.51, 0.92, 0.83, 0.15; the words six through ten will contain the breakpoint values 0.0, 0.8, 1.2, 1.5, 2.5. To change the function value for Mach 1.5 we can

```
SET CM(4) = 0.65
```

To change the breakpoint from Mach 1.5 to Mach 1.6, we must calculate the position in the table (= 5 + 4) so

```
SET CM(9) = 1.6
```

will change the breakpoint. For multidimension tables, the function data all comes first, then the breakpoint data in order - first, second, third argument. Remember the run-time command SET cannot access a multidimension array.

When used, the function referenced is translated into an assignment to a dummy variable from the function look up subroutine i.e.

```
Q = 0.5*RO(H)*V**2
```

would become

$$Z09999 = ZZF1(10, Z09998, RO, H)$$

$$Q = 0.5*Z09999*V**2$$

where the first argument of ZZF1 (the 10) is the number of breakpoints, the second argument is the current breakpoint interval, the third argument is the array name which will contain the function value and the fourth argument is the expression that is the original argument expression. Note the problem with labels - if the original statement is labelled so that control can be transferred by other GOTO's. i.e.

$$L1 \dots Q = 0.5*RO(H)*V**2$$

The label will still be attached to the Q = statement in the translated text and the Z09999 = assignment will be bypassed unless control flows directly. Labels in general should be avoided and when used should only be attached to CONTINUE cards to prevent this problem (see Section 2.3 for more details)

4.101 TAN

Take the tangent of the real argument x which must be expressed in radians

$$y = \text{TAN}(x)$$

4.102 TERMINAL

Identifies the block of code performed at the end of each run. It must be accompanied by a matching END card. In order to save calculating variables over and over again during the simulation run, the calculation can be placed in the TERMINAL block and executed only at the end of the run. Radial miss distance is a case in point where range components XMT, YMT and ZMT may be available throughout the flight. Radial miss distance at the end would be computed from

$$\text{MISS} = \text{SQRT}(\text{XMT}**2 + \text{YMT}**2 + \text{ZMT}**2)$$

Placing this in the TERMINAL section would save the extra central processor time to evaluate this expression every integration step. Code in the TERMINAL section is not sorted.

4.103 TERMT

The terminate conditions must be specified that will stop the simulation run. In an explicit program, control will be transferred to the TERMINAL region and from thence back to the executive which will interpret the next sequential command. An implicit program will transfer control back to the executive directly. Standard form of the operator is:

TERMT (logical expression)

The run will terminate when the logical expression is .TRUE.. More than one TERMT statement can be used though it is usually better to extend the logical expression in the argument to cover all possibilities. It should normally be placed in the DYNAMIC region of an explicit program.

Example,

TERMT((H. LE. 0.0). OR. (V. LE. VMIN). OR. . . .

(T. GE. TMAX))

A TERMT statement placed in the DYNAMIC section will stop the simulation at a communication interval. A TERMT statement placed in a DERIVATIVE or DISCRETE section will stop the simulation at the integration step (calculation interval) following when it becomes .TRUE..

4.104 TRAN

Transfer functions in the form of a ratio of polynomials in the Laplace operator, s , may be directly implemented in ACSL by the transfer function simulation operator. The simple first order transfer function - REALPL and LEDLAG and second order CMPXPL are preferred since the code generated is more efficient. Higher order operators should use TRAN. Standard forms are:

$$y = \text{TRAN}(nn, nd, qn, qd, x)$$

$$\text{TRAN}(y = nn, nd, qn, qd, x)$$

where:

nn is an integer CONSTANT giving ORDER of the numerator polynomial

nd is an integer CONSTANT giving the ORDER of the denominator polynomial

qn is the coefficient array for the numerator (may be a real constant if nn is zero)

qd is the coefficient array for the denominator

x is the input, an arithmetic expression of arbitrary complexity

The polynomials are in the form of highest power of s coefficient first and any coefficient that is missing must be input as zero. Note $nn + 1$ and $nd + 1$ numbers are required in the numerator and denominator arrays since it is the ORDER that is defined, not the number of coefficients.

All initial conditions are taken as zero and nn and nd must be integer constants, not symbols. That is, the order cannot be changed during a run, nor can it be changed artificially by setting the highest power of the denominator polynomial to zero, i.e., $QD(1)$ must be nonzero.

Example 1:

$$G(s) = \frac{3s + 2}{s^3 + 2s^2 + 5}$$

ARRAY P(2), Q(4)

CONSTANT P = 3.0, 2.0, Q = 1.0, 2.0, 0.0, 5.0

OUT = TRAN(1, 3, P, Q, IN)

Note the s^1 term has to be filled in as a zero in the Q array.

Example 2:

$$G(s) = \frac{K}{s^3 + 1}$$

ARRAY D(4)

CONSTANT D = 1.0, 0.0, 0.0, 1.0

Z = TRAN(0, 3, K, D, 5*X + COS(TH))

Note when the numerator is a single value it does not need to be declared in an array. Some way of calculating the value - constant or assignment statement or expression must be provided, however. Figure 4-14 lists the mechanization of this operator as a system macro. This listing is to be viewed as an example of the complexities that can be implemented using macros.

4.105 TYPE

Variables may be typed and optionally dimensioned at the same time by the three statements REAL, INTEGER and LOGICAL. The standard form of the statements is:

```

MACRO TRAN(OUT, NN, ND, P, Q, IN)
MACRO ASSIGN N
MACRO REDEFINE I, Z, ZD, ZIC
MACRO RELABEL L1, L2
MACRO MULTIPLY 0
MACRO INCREMENT NN
MACRO 10. .IF(N=ND)20
MACRO IF(N=1000)999
MACRO INCREMENT 1
MACRO GOTO 10
MACRO 20. .CONTINUE
ARRAY Z(ND), ZD(ND), ZIC(ND)
CONSTANT ZIC=ND*0.0
PROCEDURAL(ZD=P, Q, IN)
ZD(1)=IN-Z(1)*Q(2)
MACRO IF(ND=1)25
DO L1 I=2, ND
ZD(1)=ZD(1)-Z(I)*Q(I+1)
L1. .ZD(I)=Z(I-1)
MACRO 25. .CONTINUE
ZD(1)=ZD(1)/Q(1)
END
MACRO DECREMENT NN
MACRO IF(NN=ND)26
PROCEDURAL(OUT=P, Z)
MACRO IF(NN=0)30
OUT=P(1)*Z(N)
MACRO GOTO 27
MACRO 26. .CONTINUE
PROCEDURAL(OUT=P, Z, ZD)
OUT=P(1)*ZD(1)
MACRO 27. .CONTINUE
DO L2 I=1, NN
L2. .OUT=OUT+P(I+1)*Z(I+N)
MACRO GOTO 40
MACRO 30. .CONTINUE
OUT=(P)*Z(ND)
MACRO 40. .CONTINUE
END
Z=INTVC(ZD, ZIC)
MACRO EXIT
MACRO 999. .PRINT NUMERATOR GREATER THAN DENOMINATOR
MACRO END

```

Figure 4-14. Listing of TRAN (Transfer Function) Operator Macro

REAL	$v_1, v_2, v_3, \text{etc.}$
INTEGER	$v_1, v_2, v_3, \text{etc.}$
LOGICAL	$v_1, v_2, v_3, \text{etc.}$

where the V_i are either simple variable names or else subscripted arrays with 1, 2 or 3 integer CONSTANT subscripts separated by commas.

Examples are:

```

INTEGER K, JJ(10), FRED(2, 2)
LOGICAL FLAG

```

Note no variable name need be typed as REAL as all variables are assumed to be this form unless explicitly typed otherwise. The FORTRAN convention that symbols starting with I, J, K, L, M or N are integer does *not* hold.

4.106 UNIF

A uniform random number sequence can be generated by

$$y = \text{UNIF}(\ell, u)$$

Result is that y is a random variable uniformly distributed between a lower value ℓ and an upper value u .

WARNING: The power density or what is usually more important, the low frequency power, will depend on the calculation interval. Variable step integration methods can produce peculiar results. See OU operator.

4.107 UNIFI

See Gaussian initialization, GAUSI.

4.108 VARIABLE

A nonsubscripted variable is designated as the independent variable for integration with its initial value given by a real constant. It is written in the form

$$\text{VARIABLE name} = \text{real constant, initial condition name} = \text{real constant}$$

If this statement is omitted, the independent variable will be called T, with an initial value of 0.0. The initial condition value will not have an accessible default name.

4.109 WRITE

See I/O statements.

4.110 XERROR

See MERROR.

4.111 ZHOLD

A zero order hold can be implemented with the standard forms:

$$y = \text{ZHOLD}(ic, p, x)$$

$$\text{ZHOLD}(y = ic, p, x)$$

where:

ic is $y(0)$ if $p(0)$ is false

p is the sampling switch, i.e.,

y = x, while p is true.

y = previous value while p is false.

x is the input to be sampled - an arithmetic expression of arbitrary complexity.

Figure 4-15 gives a listing of the mechanization of this operator as a system macro.

```
MACRO ZHOLD(Y, IC, SW, X)
MACRO REDEFINE YL, YN
CALL ZZICS(YN=IC)
CALL ZZICS(YL=IC)
Y=YN
CALL ZZHOLD(X, IC, SW, YL, YN)
MACRO END
```

Figure 4-15. Listing of ZHOLD Operator Macro

A monostable can be implemented using this feature by defining a triangular waveform DL that is zero when START becomes true. Before START is true, the large initial condition ensures that DL has a large positive value

LOGICAL START, MONO

DL = T - ZHOLD (-1.0E100, START, T)

MONO = DL. LT. TMONO

The logical variable MONO will be true for a time TMONO after START becomes true. Arrangements should be made for turning off START (making false) once MONO becomes true so that it is only activated by a pulse.

IF (MONO)START = .FALSE.

The ZHOLD step will only take place on the first pass through the DERIVATIVE subroutine of each calculation interval. Subsequent passes and iterations for predictor- corrector algorithms are protected from discontinuities in this way. Thus it is necessary to wait for the hold to be established before turning off the activating variable (START).

This statement requires the sort algorithm to be operative, so it should not be included within a PROCEDURAL block.

NOTE: The ZHOLD operator should not be used in a PROCEDURAL block that will prevent the correct sorting of any intervening statements.

4.112 ZOH

A zero order hold that is similar to the one used in MIMIC may be invoked by this function. The output is sampled repetitively every dt units. The standard forms are:

y = ZOH (x, ic, tz, dt, i)

ZOH (y = x, ic, tz, dt, i)

where:

x is the input expression

ic is the initial value for the output - up to tz

tz is the time the sampling action starts

dt is the sampling interval

i is an optional integer variable starting at one and incremented before each sampling action:

Fill an array by

ZOH(ARR(I)=x, ic, tz, dt, I)

Figure 4-16 presents a listing of this operator as a system macro

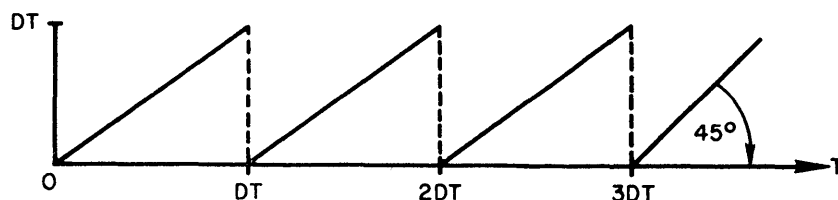
```
MACRO ZOH(OUTPUT, INPUT, IC, TO, DT, I)
MACRO REDEFINE TL, TN
MACRO ASSIGN N
PROCEDURAL(TL=TO)
TL=TN
CALL ZZICS(TL=TO)
END
MACRO IF(N=6)10
MACRO GOTO 20
MACRO 10. CONTINUE
MACRO REDEFINE IZ
INTEGER I, IZ
CONSTANT IZ=1
CALL ZZSMPL(I=I+1, IZ, TL)
MACRO 20. CONTINUE
CALL ZZSMPL(OUTPUT=INPUT, IC, TL)
CALL ZZSMPL(TN=TL+DT, TO, TL)
MACRO END
```

Figure 4-16. Listing of ZOH Operator Macro

In order to generate a triangular wave (TWV) use

$TWV = T - ZOH(T, 0.0, 0.0, DT)$

The output of the zero order hold sampling T itself will be a staircase. When this is subtracted from T a triangular wave will result for:



THIS PAGE INTENTIONALLY LEFT BLANK

5. ACSL RUN-TIME COMMANDS

Once the model has been translated, and has gone through the FORTRAN compile and load and is in execution, control is by a sequential set of commands that exercise the model. The order of execution is the order given and no branching or looping commands are available. Data values may be changed and once changed stay that way until changed by command or unless the code in the model definition recalculates them. Control is retained in the executive section until a START command is received when control is released to the model program and the integration starts. Some method of terminating the run must be present in the model definition code otherwise the executive will never regain control. (See TERMT operator, Chapter 4.)

For typical command sequences used to exercise the models see examples in Appendix A. Throughout the description of these run-time commands the philosophy employed is that symbols or variable names imply the value stored in the location corresponding to the name. Strings in quotation marks stand for themselves and so may be general Hollerith data and act as a title, comments or subcommands or qualifiers to each of the main commands.

If any data is needed by the command it must be supplied in the correct format - either integer, real or logical. An integer is a string of digits with no decimal point, a real number is a string of digits with a decimal point and/or exponent (E); a logical constant is .TRUE. or .FALSE. Hollerith data, i.e., strings in quotation marks, are considered to be of type integer. Integers used for real variables will be automatically floated prior to use.

Whenever a data element is expected, a symbol can be substituted and the program will consider the data element to be that contained in the symbol. The type of the data is considered to be that associated with the symbol when the model was defined. As an example, consider

```
OUTPUT T, A, B, 'NCIOUT' = I
```

The subcommand 'NCIOUT' expects an integer following that is going to define the number of communication intervals between OUTPUTs. The current value in the symbol I is used to specify this number. The symbol I must have been typed explicitly by

```
INTEGER I
```

in the model definition section since all variables not so typed are considered real. This feature does not provide a variable output rate changeable by the value in I as the simulation proceeds. It is the current value in the symbol that is used at the time the command is executed.

Commands are the first variable names on the statement, separated from the arguments to the command by one or more spaces. Extra commands can be written on the same line by delimiting them by a dollar sign (\$). A command may extend over the end of a line by terminating with an ellipsis (three periods . . .). The following line is appended to the end of the previous card with trailing blanks suppressed. Leading blanks on the continuation card are not eliminated. Be careful when splitting a symbol that no blanks are accidentally inserted into the resulting string. Do not forget the delimiter if splitting between symbols (usually a comma) - it can be at the end of the card before the ellipsis or on the beginning of the following card. The latter is preferred mainly so that it stands out and cannot get lost if a large number of continuation cards are used.

Arrays may be accessed by element but only a single index is allowed, i.e., referring to TABLE (2,4,3) is illegal. You should calculate the position in a linear array by assuming the first index varies fastest. If the previous array had been defined by

```
ARRAY TABLE (10, 10, 10)
```

the (2, 4, 3) element could be accessed by TABLE(232) $(2 + (4 - 1)*10 + (3 - 1)*10*10)$. In general, it is recommended that arrays with more than one dimension be avoided.

5.1 ACTION

Actions can be scheduled at different values of the independent variable. At present, these consist in changing a variable value such as setting a switch.

The debug printout is produced if a system integer variable NDBUG is greater than zero. If it is, the debug list is written out and NDBUG is decremented by one. For example, five debug listings are needed at $T = 0.0$ and two at $T = 1.0$. We can satisfy this requirement by

```
ACTION 'VAR' = 0.0, 'VAL' = 5, 'LOC' = NDBUG ...  
      , 'VAR' = 1.0, 'VAL' = 2, 'LOC' = NDBUG
```

Read this as schedule an action when the independent variable ('VAR') is zero, take a value ('VAL') of five and place it in the location ('LOC') NDBUG. When the independent variable is 1.0, take a value of 2 and place it in location NDBUG. 'VAR' and 'VAL' must precede the 'LOC' they refer to. Output rates can be changed by scheduling similar actions on NCIOUT (number of communication intervals per output - default one). Each ACTION card is cumulative, the action scheduled being added to a linked list. It is not necessary to order the values of the independent variables. As with all other commands, a symbol name may be used where a data item is expected and the contents of this variable will be used. The value used however is the value at the time the ACTION card is analyzed, not that at the time the ACTION is performed.

To remove all scheduled actions and start afresh, use

```
ACTION 'CLEAR'
```

Each time 'LOC' is mentioned, an ACTION is set up using the then current values for 'VAR' and 'VAL' which don't have to be changed every time.

5.2 ANALYZ

The ANALYZ command invokes a linear analysis capability that can evaluate the Jacobian, trim the state variables to null the rates and also calculate eigen values and their associated eigen vectors. Subcommands available under this generic command are:

```
'TRIM', 'JACOB', 'EIGEN', 'FREEZE', 'EIGVEC', 'EIGPER',  
'DISPLY', 'LIST', 'RMSEMX', 'FRACMX', 'FRACDL'
```

The subcommands 'TRIM', 'JACOB' and 'EIGEN' are action commands with no data, the others on the list require a value or a name to follow.

5.2.1 'TRIM'

The subcommand 'TRIM', transfers the initial conditions to the state variables computes the Jacobian and then using Newton-Raphson iteration adjusts the state variables until the derivatives go to zero i.e., if

$$\dot{X} = [A] X + [B] U$$

the iteration is:

$$X_{n+1} = X_n - [A]^{-1} X_n$$

Use 'FREEZE' to remove states that cause the determinant of [A] to become zero. Use 'FRACMX' and 'FRACDL' to limit the magnitude of the state change per iteration step. It is necessary to follow the TRIM by a REINIT (q.v.) in order to run the simulation from the steady state condition.

5.2.2 'JACOB'

The subcommands 'JACOB' calculates the Jacobian about the current point in state space by numerical perturbation. The result is then printed out as a large matrix. Note the states must have been given values either by a preceding START or a 'TRIM'.

5.2.3 'EIGEN'

This subcommand 'EIGEN' calculates the Jacobian and then evaluates and lists the complex eigen values and optionally the eigen vectors and/or the performance of the eigen evaluator (requires access to IMSL library for EIGRF routine).

5.2.4 'FREEZE' = X, Y . . .

This subcommand eliminates the listed variables from the state vector. Useful to eliminate open loop integrators prior to the trim operation. Open loop integrators form a zero column in the Jacobian so the inverse doesn't exist. Variables are only frozen for a single ANALYZ command and the action must be repeated for each invocation.

5.2.5 'EIGVEC' = .T. (Default is .FALSE.)

This command should precede 'EIGEN' and determines whether eigen vectors are to be calculated. Once set, it stays that way until changed.

5.2.6 'EIGPER' = .T. (Default is .FALSE.)

This subcommand should precede 'EIGEN' and determines whether performance and figure of merit is given by the eigen vector analysis routine. If the figure is less than 1, the eigen values are considered very good, between 1 and 100 probably alright and over a hundred the results can't be considered to have any accuracy. Once set, the flag stays that way until changed.

5.2.7 'DISPLY' = .T. (Default is .FALSE.)

In general ANALYZ operations are considered high volume operations written only on the PRN unit (like PLOT and PRINT). If 'DISPLY' is set true, output is repeated on the DIS unit if different for monitoring at the terminal. Once set, the flag stays that way until changed.

5.2.8 'LIST' = .T. (Default is .FALSE.)

This subcommand turns on a flag that writes out details of the 'TRIM' iteration - successive values of the state vector, the derivative vector and residual. Once set, this flag stays that way until changed.

5.2.9 'RMSEMX' = 0.001 (Default is 0.0001)

This subcommand specifies the allowable error at which trim convergence is obtained. Each state is associated with an allowable error in usual fashion by

$$dX_j = \text{AMAX1}(XE_j, ME_j \cdot \text{ABS}(X_j))$$

where XE_j and ME_j are the absolute and relative errors specified for the particular state. This quantity is used in the Jacobian calculation as from

$$\begin{aligned} \dot{X}_1 &= F_1(X_1, X_2, \dots, X_j, \dots) \\ \frac{\text{der}(F_1)}{\text{der}(X_j)} &= \frac{F_1(\dots, X_j + dX_j, \dots) - F_1(\dots, X - dX_j, \dots)}{2 dX_j} \end{aligned}$$

During trim the weighted residual R is computed from:

$$R = \sqrt{\sum_{j=1}^N \left(\frac{\dot{X}_j}{dX_j} \right)^2}$$

and the iteration is continued until R is less than the value defined by the 'RMSEMX' subcommand. Once set, the value stays that way until changed.

5.2.10 'NITRMX' = 10 (Default is 50)

This subcommand specifies the maximum number of iterations before the 'TRIM' command gives up. For a non-linear system, judicious choice of initial condition values may be necessary before a steady state can be achieved. Once set, this iteration count stays that way until changed.

5.2.11 'FRACMX' = 0.2 (Default is 1.0E30)

This subcommand specifies the maximum fractional change in a state variable - applied at each step of the TRIM iteration. In highly non-linear problems the linear extrapolation can move the state vector into unallowable regions in state space if unconstrained. The use of 'FRACMX' is to restrict the movement and typical values are 1% to 10% or 'FRACMX' = 0.10. The state movement (state X) is never limited to a smaller value than allowable error dX_j .

$$X_{n+1} - X_n \leq \max(dX_j, \text{FRACMX} \cdot \text{abs}(X_n))$$

5.2.12 'FRACDL' = 0.8 (Default is 1.0)

This subcommand specifies the fractional step actually taken when compared with that calculated by the Newton-Raphson iteration. With non-linear problems, taking the full step, 'FRACDL' = 1.0, can cause cycling. The amount of the step can be reduced to prevent this, though at the expense of convergence speed.

5.2.13 EXAMPLES OF ANALYZ COMMAND

As examples of the use of the ANALYZ command, consider the following. In general, specification statements come before action statements. The sequence of operations is performed in a left to right manner with as many continuation lines as necessary.

```
?ANALYZ 'LIST'=T., 'FREEZE'=X, 'TRIM', ...  
?      'JACOB', 'EIGEN'
```

This command turns on the list, eliminates ('FREEZE') the state X and its corresponding derivative from the Jacobian, finds a steady state and then evaluates the Jacobian and eigen values.

```
?START  
?ANALYZ 'JACOB', 'EIGVEC'=T., 'EIGEN'
```

This command determines the Jacobian about the point in state space where the simulation terminated, specifies that eigen vectors are required and then calculates both eigen values and eigen vectors. Note if we'd used 'TRIM', the ending states would have been overwritten with the initial conditions.

```
?ANALYZ 'TRIM'  
?REINIT
```

This command uses the current initial conditions to establish a steady state and then the REINIT command writes the current state vector back over the initial condition vector. Every subsequent START will now start from a trim or steady state condition.

The use of ANALYZ presupposes that the model is defined by a sequence of non-linear equations of the form

$$\dot{X}_1 = F_1(X_1, \dots, X_j, \dots, X_N, t)$$

The use of self contained states within the model - those not defined by INTEG and INTVC - will usually prevent the correct evaluation of the Jacobian. Likewise, memory operators such as ZHOLD, BCKLSH and DELAY should not be present.

Another point is that all initial conditions must be defined before 'TRIM' is invoked. If some are calculated in the INITIAL section, set a run time of zero and execute once prior to using ANALYZ i.e.

```
SET TSTP = 0.0
START
ANALYZ 'LIST' = .T., 'TRIM'
REINIT
SET TSTP = 99.9
START
```

...

REINIT is used after the 'TRIM' to move the trimmed state variables back into the initial condition values so that the steady state point in state space will become the starting point for any subsequent run (START).

5.3 COMMENT

Comments may be included in the run-time drive cards by quoting the entire command. i.e.

```
START $ 'EXECUTE THE MODEL'
```

The comment may not contain either a quote (') or dollar sign (\$).

5.4 CONTIN

The run may be continued where it left off, if the stop conditions have been changed, by CONTIN. This operation bypasses writing the initial condition into the state vector so that the program will continue to integrate from the previous position in state space. Note this command only makes sense after a START card has established a state and also the termination condition of the previous run must have been changed, otherwise it will stop immediately, i.e.,

model:

```
CONSTANT TF = 10.0
TERMT(T. GE. TF)
```

run-time commands:

```
START           $'Run to 10.0 sec'
SET TF = 15.0   $'Extend to 15.0 sec'
CONTIN          $'Run 10.0 to 15.0 sec'
```

5.5 DISPLY (sic): Short Form D

Display the values that are currently in the named list. Usually used at the end of a run to determine terminal values.

```
DISPLY X1, X2, X3(5)
```

Arrays, if not explicitly referred to by element, are listed in total.

5.6 END

Tells the run-time executive that a PROCEDURE definition is complete.

5.7 MERROR, XERROR

The relative and absolute errors allowable during the integration step may be individually specified by these commands. Each state is associated with a corresponding entry and the relative error table (MERROR) and absolute error table (XERROR): These two commands behave like a SET (q.v.) except that they access the error table corresponding to the state name rather than the state name itself

MERROR SV1 = 1.0E-3, SV2 = 0.01

XERROR SA1 = 5*0.0001, SA2(3) = 0.02

where the SV_i and SA_i are state variables and state arrays respectively. Note that arrays may be filled by a repeat count: An individual array element may be specified - at model definition time the same allowable error must be specified for all elements in an array.

During the integration step, the estimated error for each state variable is compared to the allowable error E_i obtained from

$$E_i = \max (X_i, M_i | Y_i |_{\max})$$

where $(Y_i)_{\max}$ is the maximum of the absolute value of the state achieved so far during the run.

5.8 OUTPUT

The output from the model can be obtained by the command

OUTPUT T, A, B, C(5), D

The command designates the list elements as data whose values are to be listed as the simulation model advances in time. This action occurs following a subsequent START (q.v.) command. Subcommands are available and when used must be quoted so

'NCIOUT', 'CLEAR'

The values of the list elements will normally be written out every communication interval. If this rate is too high, a reduction can be effected by including the subcommand 'NCIOUT' - number of communication intervals between output. i.e.

OUTPUT T, A, B, 'NCIOUT' = 5

Only one value for 'NCIOUT' can be in effect at one time - the last value set stays in effect until changed in a subsequent OUTPUT command. The starting value of 'NCIOUT' is one. It isn't possible to designate different list element blocks to be recorded at varying data rates - all elements on the OUTPUT list are given together.

Array elements can be listed by referring to the specific element (A(3)say). If the name given is an array but no specific element is called for, all elements will be listed.

OUTPUTs are cumulative. To clear the list and start afresh, use the 'CLEAR' subcommand.

OUTPUT 'CLEAR', BILL, JOE, SAM

Output is considered to be low volume and will be written to both the DIS and PRN files if different. The width of the line is controlled by the system variable TCWPRN (terminal character width) which controls the line width of data being written on the DIS logical unit number. If TCWPRN is set to 72, output will be three variables across: If TCWPRN is 132, output will be five variables across.

NOTE: The independent variable is not included on the OUTPUT list automatically - it must be deliberately mentioned if you want to see it.

5.9 PLOT

Both printer and line plots can be made using values recorded for any of the variables on the PREPAR list. The actual form of the plots are controlled by system symbols (q.v.) PRNPLT (printer plots), CALPLT (line plots) and STRPLT (strip plots). Note the order of commands must be:

PREPAR - establishes list to be saved (used once)

START - runs the model, saves values on the PREPAR list

...

PLOT - makes plots using the data saved during the run

with any other commands in between the above sequence. It is illegal to change the PREPAR list and continue plotting data made with a previous START. It is the current PREPAR list that is used by the PLOT command to find the data for each variable on the intermediate scratch file.

The basic form of the plot command is:

PLOT Y1, Y2, Y3, . . . , YN

which will plot the variables Y1, Y2, Y3, . . . , YN - picking as the X-axis the first variable on the PREPAR list. The form (printer plots, line plots or strip plots) will be determined (all, either or none) by the current settings of the system symbols PRNPLT, CALPLT and STRPLT.

Sub-commands are available and when used must be in quotes so:

'XAXIS', 'XHI', 'XLO', 'XLOG', 'XTAG', 'SAME', 'OVER',

'ALL', 'HI', 'LO', 'CHAR', 'LOG', 'TAG'

5.9.1 X-AXIS QUALIFIERS (XAXIS, XHI, XLO, XLOG, XTAG)

The X-axis can be changed from the default to any variable on the PREPAR list as in:

PLOT 'XAXIS'=X2, 'XLO'=5.0, 'XHI'=10.0, Y1, Y2

In addition to specifying a new X-axis variable, X2, the scales are also given so that the X-axes will run from 5.0 (low) to 10.0 (high). There is a reason for specifying the X-axis change first on the PLOT command line since the automatic scaling for the Y-axis variables is determined by picking maxima and minima within the current X-axis window when the Y-axis variable is reached, in a left to right scan. Other options relating to the X-axis variable are logarithmic scales, 'XLOG', and a character string or tag that can be appended to the right of the variable name - can be used to identify units for instance

PLOT 'XAXIS'=W, 'XTAG'='(RAD/SEC)', 'XLOG', 'XLO'=WMN, . . .

GAIN, PHASE

The respecification of the X-axis variable resets all parameters to their default values, which is why the other subcommands follow the X-axis definition in the left to right sequence. The default values are no tag string, linear scales and axis limits chosen by rounding the maximum and minimum values on the data file. When logarithmic scales are selected the minimum value must usually be specified if the default rounding is to zero. Zero or negative scale limits are not permitted with logarithmic scales and will produce a diagnostic message and a replacement with linear scales.

X-axis related parameters, once set, remain that way for all subsequent PLOT commands unless respecified or the X-axis variable itself is changed.

5.9.2 Y-AXIS QUALIFIERS (HI, LO, CHAR, LOG, TAG)

Individual y-axes variables can have the scales set, the character selected, logarithmic scales specified and a tag string given. These subcommands all follow the y-axis variable name as qualifiers, applying to the variable name to the immediate left. As an example, consider:

PLOT Y1, 'LO'=0.0, 'HI'=10.0, Y2, Y3, 'LO'=5.0, . . .
'HI'=5.0, 'CHAR'='T'

Any variable not followed by given scale factors will have 'best' scales chosen automatically. i.e., Y2 in the above example. The automatic scaling is obtained by determining the maximum and minimum values of the variable while the x-axis variable is between its given limits 'XLO' and 'XHI'. This operation ensures that the scales are chosen appropriately for the plot since it may be necessary to examine in detail a small fraction or window within the total simulation run. Other options include logarithmic scales 'LOG', and a character string or tag that can be appended next to the variable name i.e.

PLOT Y1, 'LOG', 'TAG'='GAIN', 'LO'=0.001

where the variable Y1 will be plotted on logarithmic scales. The axis will be labelled with the name Y1 concatenated with the string GAIN with a space in the middle.

It is sometimes necessary to set a group of scales to the same, originally unknown, value. If maximum and minimum values can be collected in the model definition section (the DYNAMIC section for preference), then they can be used to set scale factors by using the symbol name instead of a constant i.e.

PLOT Y1, 'LO'=YMIN, 'HI'=YMAX, etc.

Remember, any symbol stands for the value of its contents where a data item is expected.

When any array is plotted, the array name stands for all its elements and any qualifiers such as scale factors explicitly given apply to all elements of the array. For individual qualifiers each element of the array must be individually specified on the PLOT command list.

5.9.3 SAME, OVER and ALL

The 'SAME' and 'OVER' subcommands act on a string of Y-axis variables - all the variables to the left of the keyword. The command:

PLOT Y1, Y2, Y3, 'SAME'

will apply the scales selected for the first name in the list, Y1 in this case, to the rest of the elements in the list, Y2 and Y3 as shown. Note this option doesn't pick the maximum of the maxima and the minimum of the minima.

The key word 'OVER' is used to suppress the extraneous printing of the axes and is normally used in conjunction with 'SAME'. The command line:

PLOT Y1, Y2, Y3, 'OVER'

will draw and label the vertical axes for Y1, the first element in the list but will suppress the separate axes and labels (although the scales may be different) of Y1 and Y2. The main use of this is comparison plots using the strip chart option (STRPLT = .T.) because otherwise all plots on the list are drawn on separate areas of the graph.

All the variables on the PREPAR list can be plotted by a single command so:

PLOT 'ALL'

used normally for debug purposes. Plots are drawn nominally three to a page (see system symbol NPPPLT or number of plots per page).

5.9.4 EXAMPLES OF PLOT COMMAND

Some examples of PLOT commands are as follows: X, Y, Z and W are regular variables; A is a three element array.

- 1) Normal plotting, x-axis operations specified first:
PLOT 'XTAG'='(SEC)', X, Y, Z, W
- 2) Plot X and Y to the default scale of X, and Z and W to same given scale:
PLOT X, Y, 'SAME', Z, 'LO'=0.0, 'HI'=5.0, W, 'SAME'
- 3) Plot array A normally - all element scale factors are individually chosen:
PLOT A
- 4) Plot array A using default scales for A(1):
PLOT A, 'SAME'
- 5) Plot array A on given scales - the low and high apply to all elements of the preceding array
PLOT A, 'HI'=50.0, 'LO'=-50.0
or
PLOT A, 'HI'=50.0, 'LO'=-50.0, 'SAME'
- 6) Plot A(1) on the same scale as A(2) and add descriptive tag to A(3)
PLOT A(2), A(1), 'SAME', A(3), 'TAG'='(FURLONGS)'
- 7) Use logarithmic scales and tag descriptors:
PLOT 'XAXIS'=W, 'XLOG', 'XLO'=0.01, . . .
'XTAG'='(RAD/SEC)', X, 'LOG', 'LO'=1.0E-4, . . .
'TAG'='-GAIN'
- 8) Plot X and Y to the same scale as X; Z and W to the same scale as Z:
PLOT X, Y, 'SAME', Z, W, 'SAME'

5.10 PREPAR

The variables that are to be recorded on a scratch file during the run are listed so

```
PREPAR T, A, B, C(2)
```

The same comments on the arguments apply as to the OUTPUT command, with the exception that recording is every communication interval.

The plot programs assume as a default that the X-axis variable is the first one on the PREPAR list. The PREPAR list is cumulative. Reset is by PREPAR 'CLEAR', T, JOE, . . .

5.11 PRINT

All the variables on the PREPAR list can be listed in columnar form (10 columns to a page) once the run has been completed and the PREPAR file established. PRINT is similar to PLOT in that it can have a list of variables and/or subcommands. Subcommands are

```
'ALL', 'NCIPRN'
```

For example:

```
PRINT 'NCIPRN' = 5, 'ALL'
```

will print all the variables on the PREPAR list, listing values every five communication intervals (NCIPRN = number of communication intervals for print).

```
PRINT T, X1, X2, X3, X4
```

```
PRINT 'NCIPRN' = 2, T, Y1, Y2, Y3, Y4
```

...

Successive print commands can be used to format the data. Five columns are normally used for listings to be reproduced directly on "A" size (eight-and-a-half by eleven) paper. Note that the independent variable has to be included on each list if needed - it doesn't get printed automatically. The 'NCIPRN' subcommand is optional - once used the argument becomes the default from then on.

If array names are used, all array elements will be listed. An array element may be listed separately.

```
PRINT T, ARRAYA, ARRAYB(3)
```

The data is only listed on the PRN logical unit unless HVDPRN is true, so if this is set to 9, this high volume output will go onto the PRINT file for later disposal to the printer queue.

5.12 PROCED

Command sequences can become long and cumbersome when much plotting is performed. To save repeating the directions after each run a procedure can be defined. Do not confuse this with PROCEDURAL referenced during the translation phase. This command PROCED is invoked at execution time, i.e.,

```
PROCED GO $ START
PLOT 'XAXIS' = X, Y, Z
PLOT 'XAXIS' = T, N, M
DISPLY RMISS
END
```

The sequence of commands START, PLOT - - - until END are saved but not executed. But now a command GO will execute the entire sequence, i.e.,

```
SET A = 5.0 $ GO
SET A = 6.0 $ GO
SET A = 7.0 $ GO
```

which allows the value of the parameter A to be changed before executing the sequence of commands stored in the procedure 'GO'. Note that the names and values within the procedure are fixed and cannot be changed when the procedure is invoked.

5.13 RANGE

This command determines the maximum and minimum values of variables, that have been saved during a run by the PREPAR statement. Subcommands are available, and when used must be quoted

```
'ALL', 'IHI', 'ILO', 'IVAR'
```

Standard form of the command is

```
RANGE A, B, C(5)
```

As usual, array names stand for all the elements in the array. Errors will be reported if any name on the RANGE list is not defined in the PREPAR list.

Windows may be defined so that sub-ranges can be determined, by specifying both the independent variable to be used, 'IVAR', and the high 'IHI', and low, 'ILO', values to be used for the test.

```
RANGE 'IVAR'=X, 'ILO'=50.0, 'IHI'=100.0, 'ALL'
```

which would report the maximum and minimum values of all elements on the PREPAR list when the variable X lay between 50.0 and 100.0. Once set, the 'IVAR', 'ILO' and 'IHI' values remain that way for subsequent RANGE commands, if not changed. Initial values of 'ILO' and 'IHI' are - RMX and RMX respectively where RMX is the largest floating point number available on the machine. The independent variable 'IVAR' is set to be the first variable on the PREPAR list.

5.14 REINIT

Reinitialize takes the current value of the state variables and writes them back to the initial condition table, thereby destroying the original numbers on the table. REINIT can be used when a midcourse guidance system flies out and is stopped before the terminal phase. It now establishes this current point as the starting point for subsequent runs.

Refer to SAVE and RESTOR for details on how to recover back to an original condition.

5.15 RESTOR

Restores the user's data area written on a named file. The file must have been established by a previous SAVE command

RESTOR 'fn'

where fn is a valid file name - see SAVE command.

5.16 SAVE

The entire contents of the user's data block may be saved on an external file to be subsequently RESTORed and so override any intermediate changes. ACSL system constants (TITLE, PRNPLT, XINCPL, etc. are not saved. Standard form is

SAVE 'fn'

where fn is any valid file name - starts with a letter and six characters or less. Do not use INPUT, OUTPUT, RRR or PRINT. Any number of SAVE commands may be issued on the same or different files. If it is the same file, the previous information is overwritten and destroyed.

5.17 SET: Short Form S

Data can be set into any known constant array or variable by this command. If the model definition is going to calculate a new value of a variable, after START, then it does not usually make much sense to change it. The command would normally be used for changing the values of constants. Once changed, they stay that way until changed again.

Legal forms of the SET commands are

SET NSTP = 10, RANGE = 5.6E3, MAS = 4.6, GAIN = 5

SET LOGVAR = .TRUE., ARRAY(5) = 4.3, SWITCH = .T.

SET ARRAY(2) = 2.0, 3.0, 4.0, TSTOP = 4.6 . . .

, TITLE = 'DOPPLER STUDY', ARRAY(6) = 5*0.0

The data form must agree with the type of the symbol with the exception that integers may be set into real variables and automatic floating will take place: Integer variables must not have a decimal point and logical variables can only be .TRUE. or .FALSE. (.T. and .F. are Short Forms). Arrays can be set by individual elements or a data list can follow when succeeding data items will be stored in subsequent array slots. Attempting to exceed the array length will result in an error. Data can also be obtained from another symbolic location by using the symbol name, i.e.,

SET RMI = RANGE, RMT = RANGE

when after execution the variables RMI and RMT will both contain the value of the number stored in the symbol RANGE. This concept is useful if a data item has to be stored in many places. It can be stored by value once and then picked up by name subsequently, thus value changes need only be made in one place.

Hollerith data can be set into a symbol or array that is of type integer. Normally only the system array TITLE would be so used.

5.18 SPARE

A spare command is provided that will link to a user provided subroutine SPARE. A default version of this subroutine is available that will list the central processor time, i.e.,

SPARE

ACCUMULATED CP TIME nnn.nnn SEC. ELAPSED CP TIME nnn.nnn SEC.

The accumulated time is normally the time from the beginning of the job: The elapsed time is the incremental time from the previous invocation of the command. The sequence

SPARE\$START\$SPARE

can be used for timing simulation execution.

5.19 START

Command to allow the model definition to integrate over the state trajectory. Control is released to the model definition program and provision must be made to terminate execution at some time.

5.20 STOP

Tells the run-time executive that no more commands follow. A termination record is written on the line plot file, if any plots have been made.

This command should always be the last command issued so that all the files established by the executive are cleaned up or terminated correctly. It may be necessary to precede STOP with a blank for those operating systems that interpret STOP as an abort.

5.21 XERROR

Absolute error specification - see MERROR.

6. MACRO LANGUAGE

The macro capability of ACSL allows the user to expand the language capability by defining new operators as the need arises.

A macro may be used in one of two essentially distinct ways. The first is akin to a subroutine or function which is defined once and then called from many places. The macro is defined once and then invoked. Actual statements are produced for each macro invocation, but the extra amount of storage used for such instructions is always small. The only way to define operators involving integrators and memory functions is by using this macro operator.*

The second approach is to define blocks and write all the equations in terms of standard nomenclature. These blocks can become part of a system library and with all the variables relabeled, no conflict will appear between the standard block and another user's invocation of it. For instance, a standard actuator system could be defined with input, the commanded deflection; output, the actual deflection. Invocation would then be:

ACTUAT(DLC, DL)

which would tell the processor to reproduce the code to represent the actuator but use the name DLC for command deflection; DL for the actual deflection.

Some of the more important features of the macro language are:

- 1) Variables and statement labels may be locally generated. In the event the macro is called more than once, this will prevent multiply defined variables or doubly defined statement labels.
- 2) An unlimited number of macro input arguments may be used. These arguments must be valid expressions with balanced parentheses - of arbitrary complexity - or else any character string enclosed in quotation marks.
- 3) Macro definitions may *invoke*, other macros (nesting) to an unlimited level of complexity. Note, however, that macro *definitions* may not be nested. This needs a count of nesting level within the definition to match up with the correct MACRO END. This count is not performed. The first MACRO END terminates the definition.
- 4) Macro's may be placed anywhere within an ACSL program: They must be defined, however, before they are used. Any current macro may be redefined and the most recently defined macro will be used in the expansion.
- 5) The concatenation operator (→) allows arguments to be placed together without intervening spaces so making up symbols.

Argument strings are substituted for each appearance of the macro argument name. If the argument is an expression, care must be taken that the resulting code is correct after the substitution (see Section 6.5). Other names that may be substituted during macro expansion are the ASSIGNED variable (see Section 6.3.1), local variable names identified by a REDEFINE (see Section 6.3.12) and local labels identified by a RELABEL (see Section 6.3.13).

6.1 MACRO DEFINITIONS

The macro definition is a block of code which consists of the following:

- 1) Macro definition header

* For two examples of this type of macro use, see the example program, PHYSBE; Section 8 of Appendix A.

- 2) Macro directive or ACSL statements
- 3) MACRO END

The macro definition header specifies the name of the current definition and a list of dummy reference parameters, analogous to dummy arguments in a FORTRAN subroutine. The translator scans the statements in the macro body for the appearance of these names, flagging them for substitution by the actual argument supplied on invocation. The definition terminator, MACRO END, must be present to flag the translator to return to direct translation instead of saving the macro skeleton (the macro body with the substitutable arguments flagged is known as the skeleton).

If a macro name is the same as one already defined, either in the system macro file or the current model definition, the new macro definition replaces the old one. No error message is issued since this is considered to be a feature whereby the user can always override an old macro definition.

6.2 MACRO DEFINITION HEADER

Two types of macro can be defined but for the most purposes - excluding arrays - the first one is to be preferred. This is of the form:

MACRO identifier ($x_1, x_2, x_3 \dots, x_n$)

where identifier will be the macro name (6 characters or less)

x_i are variable names - not constants or expressions.

Anywhere the symbol x_i is referred to in the macro definition it will be replaced by the i th argument - symbol or expression - when the macro is invoked. Example:

MACRO MULT(X, Y, Z) \$ 'DEFINITION'

X = Y*Z

MACRO END

C = A*MULT(5.0*B, D) \$ 'INVOCATION'

The output of the function is considered the first argument X; Y will be replaced by 5*B and Z by D everywhere throughout the definition.

A second type of definition is useful when handling arrays. This form has an extra macro so

MACRO MACRO identifier (p, q, r, s)

where identifier is as defined previously, p is the primary argument (any symbol) and q, r, s are secondary arguments, optional symbols. 'p' is the dummy reference parameter and may be thought of as being an array, each element of which identifies the respective elements in the argument list at invocation time, i.e.,

MACRO MACRO HEAD(P, Q) \$ 'DEFINITION'

...

MACRO END

ARRAY B(5)

HEAD (A, B = 5*D, E + F, LOW) \$ 'INVOCATION'

Now P(1) appearing in the definition body will be substituted by the symbol A, the first argument, at invocation time. P(4) will be replaced by the expression E + F, the fourth argument.* The secondary arguments allow access to the dimension of any argument from a previous ARRAY statement. The ARRAY statement must come before the macro invocation.

* Arguments at invocation time are delimited by commas or an equals sign.

Any reference to Q for an array will access the first dimension. In the example above, Q(2) will have the value 5 from the dimension in the previous ARRAY statement. Q(3) and Q(4) will be illegal since these arguments are expressions that cannot have a dimension.

Symbols substituted for the secondary arguments R and S act similarly to Q except they provide the second and third dimensions respectively.

NOTE: Macros written in this second form are extremely hard to read since no mnemonic symbols can be used for the arguments.

The array expressions are restricted to the following forms.

- 1) P(i) - i is an integer constant
- 2) P(n) - n is the ASSIGNED variable (q.v.)
- 3) P(n ± i) - combination of the above.

In general, the substitutable symbols must be separated from other character strings by nonalphanumeric characters, i.e., *, +, -, 'blank' in order for the scan to operate. If the above macro MULT contained the statement

```
ASSIGNZ TO K
```

The symbol Z would not be identified for substitution. Here, the space is all important so

```
ASSIGN Z TO K works well.
```

In order to allow the substitutable argument to appear next to a character string the concatenation operator is defined. This operator is a right arrow (→) on CDC systems or underline (⏟) on ASCII based machines, which serves as a separator for symbol identification, but is removed entirely from the skeleton. As an example, suppose we wish to make up unique symbols by adding an F to the third argument of the MULT macro and including it in an expression.* The new name would be written F→Z (F concatenated with Z), i.e., the statement

```
X = Y*F→Z
```

could be included in the macro definition. Invoking the MULT macro with

```
A = MULT(SAM, JOE)
```

would result in the statement

```
A = SAM*FJOE
```

where the new symbol FJOE has been defined.

The above call MULT will enter a symbol JOE on the symbol table. If only the made up symbols are important, the argument may be quoted as

```
A = MULT(SAM, 'JOE')
```

where the same expression will be generated but now the symbol JOE will not be entered in the symbol table. The argument in quotes has the quote characters removed and the literal string of characters enclosed - including blanks - is substituted for the appropriate argument symbol.

6.3 MACRO DIRECTIVE STATEMENTS

The following section lists in alphabetical order the macro directive statements that may be included within a macro definition. No code will result from these statements, but extremely flexible control is possible, of the manner in which the macro is processed at invocation time. The ACSL statements themselves produce

* See PHYSBE example program in Section 8 of Appendix A.

code, and symbols in the statements will be substituted for the appropriate argument at invocation time. All these directive statements have **MACRO** in front to indicate an instruction to the macro processor. See Subsection 6.4 for some examples of the use of a macro.

All directive statements can have labels attached to them which can be used by the **MACRO GO TO** and **MACRO IF** directives. These labels must be distinct from any labels attached to non-directive statements. The label is inserted between the leading **MACRO** of the directive, i.e.,

```
MACRO S1 . . RELABEL I
MACRO S2 . . CONTINUE
MACRO S3 . . END
```

Note, these labels control the sequence of the macro processor at macro invocation time. The labels on nondirective statements control the sequence at run-time execution.

6.3.1 MACRO ASSIGN

The **ASSIGN** macro directive statement has the standard form

```
MACRO ASSIGN n
```

where *n* is a symbol (usually **N** is used). The **ASSIGN** directive assigns the number of arguments in the macro call to the variable **N**; the value of **N** will always be an integer. If the dummy reference parameter contains a variable subscript, the variable must be the same as the variable used in the **ASSIGN** statement. Whenever **N** is used as part of the dummy reference parameter subscript, the current value of **N** will refer to the *N*th argument in the macro call list at invocation time. Example:

Second header type

```
MACRO MACRO SAM(P) $ 'DEFINITION'
MACRO ASSIGN N
. . .
MACRO END
SAM(X, Y, Z)          $ 'INVOCATION'
```

Within the macro definition

P(N) is the *N*th argument 'Z'

P(1) (N) is **X(3)**

P(N - 1) (N) is the *N*th element of the (*N* - 1) argument, **Y(3)**

6.3.2 ARITHMETIC MACRO DIRECTIVES

The arithmetic macro directives have the form

```
MACRO INCREMENT      i
MACRO DECREMENT      i
MACRO MULTIPLY        i
MACRO DIVIDE          i
```

where *i* is an unsigned integer constant, the secondary arguments: *p*, *q*, *r*, *s*; or a macro argument name that has a literal numeric integer value.

These directives provide arithmetic operations on the **ASSIGNED** variable **N**. The value of **N** may be added to, subtracted from, multiplied or divided; all arithmetic operations are performed in fixed point integer.

Example:

```
MULTIPLY 0
```

will make the ASSIGNED variable zero

To make the assigned variable equal the dimension of the second argument, the following code can be used.

```
MACRO MACRO BIL(P, Q)
MACRO ASSIGN N
MACRO MULTIPLY 0
MACRO S1 . . IF(N = Q(2)) S2
MACRO INCREMENT 1
MACRO GO TO S1
MACRO S2 . . CONTINUE
```

On exit from this section, N, the assigned variable, will have the integer value Q(2) or the dimension of the second argument. In this way N can be used as a counter or control variable irrespective of its basic purpose of transmitting the actual number of arguments used at invocation time.

6.3.3 MACRO CONTINUE

The CONTINUE macro directive has the standard form

```
MACRO CONTINUE
```

It is a do-nothing and invariably is included so that it can be labelled as so

```
MACRO L1 . . CONTINUE
```

In this form the MACRO IF or MACRO GO TO can branch to this section within the definition.

6.3.4 MACRO DECREMENT

See arithmetic macro directives.

6.3.5 MACRO DIVIDE

See arithmetic macro directives.

6.3.6 MACRO EXIT

The EXIT directive statement has the form

```
MACRO EXIT
```

It stops the generation of code at invocation time. The action is the same as a MACRO GO TO to the macro definition terminator.

6.3.7 MACRO GO TO

The GO TO macro directive statement is written in the form

```
MACRO GO TO s
```

where s is a statement label attached to ANOTHER MACRO DIRECTIVE. The GO TO provides an unconditional branch to another section within the definition.

6.3.8 MACRO IF

The IF macro directive statement is of the form

```
MACRO IF (e1 = e2) s
```

where e₁ and e₂ can be the dummy reference parameters corresponding to the call list, integer constants, character strings or the identifier used in the ASSIGN directive. They must not be expressions; s is a macro directive statement label. This directive provides for a conditional branch to the macro directive label s inside the current definition if the relation e₁ = e₂ holds. The strings e₁ and e₂ are tested character by character excluding blanks for equality.

In order to compare a character string with a string passed as a macro argument, the string must be enclosed in quotes when the macro is invoked and then the MACRO IF compares the argument with an un-quoted string i.e., if the definition is as follows:

```
MACRO TEST (ARG)
MACRO IF (ARG = TOP) LAB1
...
MACRO LAB1 .. CONTINUE
MACRO END
```

and the invocation is

```
TEST ('TOP')
```

then the macro will expand via LAB1.

6.3.9 MACRO INCREMENT

See arithmetic macro directives.

6.3.10 MACRO MULTIPLY

See arithmetic macro directives.

6.3.11 MACRO PRINT

Error messages may be handled within the macro at invocation time by this PRINT directive statement. It has the form

```
MACRO PRINT any character string except a dollar sign ($)
```

This directive lists the character string on the output device. It will override any global list control. The primary use is for the user to diagnose his own errors at invocation time and output informative messages. See the examples in Subsection 6.4.

6.3.12 MACRO REDEFINE

The REDEFINE macro directive statement has the form

```
MACRO REDEFINE v1, v2, . . . , vn
```

where v_i are variable names appearing in the body of the macro. REDEFINE identifies the variables as being locally defined and specifies that they are to be replaced by unique symbols at each invocation. The generated variables consist of the letter Z followed by five digits.

6.3.13 MACRO RELABEL

The RELABEL macro directive has the form

```
MACRO RELABEL l1, l2, . . . , ln
```

where l_i is an alphanumeric label ($i \geq 1$). RELABEL specifies that all symbols in the list are locally defined statement labels which are to be substituted for by unique generated numbers at invocation time. These labels must only be attached to ACSL statements - not macro directive statements (statements preceded with MACRO).

6.3.14 MACRO STANDVAL

The STANDVAL macro directive statement has the standard form

```
MACRO STANDVAL arg1 = c1, arg2 = c2 . . .
```

where arg_i are the dummy argument names and the c_i are literal constants that can be real (1.5), integer (5) or logical (.TRUE.). The alternate MACRO invocation can also be used with STANDVAL which has the form as follows:

```
MACRO STANDVAL P(i) = ei, P(j) = ej, . . .
```

where i is an unsigned integer constant and the e_i are constants. The statement is used to provide standard values for arguments of a macro. If the i -th argument is not given, then the constant (e_i) is used in its place. If an argument is to take its standard value, then its position in the argument list must be left empty, i.e., the delimiting commas must be present, and the absence of an argument indicated by a null string (" "). Arguments may be simply omitted at the end of the argument list without the need for commas and a null string to indicate their absence. The STANDVAL directive must immediately follow the macro header in the definition if it is used at all.

6.4 MACRO EXAMPLES

The examples will be given of the use of macro calls that demonstrate some of the uses of the directive statements and of direct parameter substitution.

6.4.1 Sampler

A sampler can be built up of a switch and two zero order holds but for convenience, the entire sequence can be embedded in a macro and invoked as a function. We would like to say at invocation time

```
Y = SAMPLE (YIC, DL, T, X)
```

where YIC will be the initial value of Y, the sample will be repeated every DL of the independent variable, T. X is the variable to be sampled. Define this macro by

```
MACRO SAMPLE (SAMP, DL, T, X, IC)
MACRO STANDVAL IC = 0.0 $ LOGICAL SNSW
MACRO REDEFINE TS, SNSW
TS = ZHOLD (0.0, SNSW, TS + DL)
SNSW = T.GE.TS
SAMP = ZHOLD (IC, SNSW, X)
MACRO END
```

Note, the output name has to be included in the argument list. The sample time TS is sampled from the function $TS + DL$, but only when T equals or exceeds TS, setting SNSW nonzero (1.0) SAMP is snapped by the zero order hold from the input argument X.

6.4.2 DOT Product

To take the vector DOT product of two arrays A and B, we would like to be able to use the functional form

$$X = \text{DOT}(A, B)$$

where X is a scalar and A and B are vectors, previously dimensioned in an ARRAY statement. Since it is a function (has one output) the operator can be embedded in an arithmetic expression of arbitrary complexity. We do not want to have to mention the dimension of the vectors in the call since that is likely to change.

To pick up the array dimension we need to use the second form of the macro header. This header designates P as the primary variable and Q as the secondary variable, that will access the dimension of the corresponding primary argument. (Figure 6-1 gives macro listings.) The REDEFINE statement ensures the variable I will not conflict with any other use. If this were omitted, a program variable I could have its value changed when the macro is executed; a potentially disastrous effect. The test needed is to see if the second and third arguments have the same dimension; if not, the DOT product is undefined and a macro error message is printed. If the dimensions are correct the DO loop summation is formed. Note, Q(2) and Q(3) will be replaced at invocation time by integers corresponding to the array size of the respective arguments: The MACRO IF must branch to another macro directive statement, hence, the label on the MACRO CONTINUE. This label could not have been attached to the following statement since this is not a macro directive statement.

At invocation time, with the call shown, this will be translated into

```
INTEGER Z09999
X = 0.0
DO 99999 Z09999 = 1, 10
99999 . . X = X + Y (Z09999)*Z(Z09999)
```

where the variable I will have been changed into a unique generated variable Z09999. If embedded functionally in an expression, this code will precede the expression evaluation and a Z variable will be used in the expression.

6.4.3 Concatenation Example: A Pressure Tank

One of the problems with using MACRO is the tendency to generate large numbers of dummy variables (Z0nnnn) which have no physical significance. All REDEFINED variables have this form. An alternate approach is to use the concatenation feature to build unique symbols that are available for plotting or printing. This technique can also reduce considerably the length of the argument list which is the other alternative when unique symbolic names are required.

As an example consider a macro to define a gas holding tank which is similar to the PHYSBE example in Appendix A. We will calculate the flow in as the difference in pressure divided by a resistance. Total pressure will be the integrated net flow divided by a volume. The macro definition now looks like

```
MACRO TANK(N)
F_N_I = (P_N_I - P_N)/R_N_I
P_N = (INTEG ((F_N_I - F_N_O)/V_N, P_N_IC)
MACRO END
```

The basic equations for different vessels can now be established by the statements

```
TANK(1)
```

```

'MACRO DEFINITION'
MACRO MACRO DOT(P,Q)
MACRO RELABEL L2 $MACRO REDEFINE I
MACRO IF(Q(2)=Q(3))L1
MACRO PRINT CONFLICTING DIMENSIONS IN DOT PRODUCT
MACRO MACRO EXIT
MACRO L1..CONTINUE
      P(1) = 0.0
      DO L2 I = 1, Q(2)
L2..P(1) = P(1) + P(2)(I)*P(3)(I)
MACRO END
      ...
      ARRAY Y(10), Z(10)
      ...
'MACRO INVOCATION'
      DOT(X = Y,Z)
      ...

```

Figure 6-1. DOT Product Macro Forms $X = \sum_{i=1}^n Y_i Z_i$

TANK(3)

and the rest of the model must specify constants as interconnections. The first invocation TANK(1) will generate

$$F1I = (P1I - P1)/R1I$$

$$P1 = INTEG ((F1I - F10)/V1, P1IC)$$

Constants must be defined elsewhere for the resistance R1I, the volume V1 and initial pressure P1IC. Variables that must be defined elsewhere are the input pressure node P1I and the output flow F10. This macro will then make available to other sections of the simulation the input flow F1I and tank pressure P1.

The alternative form of the macro invocation without the concatenation feature would have to be

TANK (F1I, P1 = P1I, R1I, F10, V1, P1IC)

with similar statements for all the other vessels. There is a trade off in deciding how to define the macro which can be considered as follows: Without the concatenation feature, argument lists become long and complicated but there is flexibility in naming and arguments can be expressions. Using the concatenation feature, the argument list is simple - one argument, usually a constant, but can be mixed. The TANK macro could have the output flow and downstream pressure specified in the argument list since these are likely to be expressions i.e.

```
MACRO TANK (N, PI, FO)
F_N_I = (PI - P_N)/R_N_I
P_N = INTEG ((F_N_I - (FO))/V_N, P_N_IC)
MACRO END
```

and in invocation of

```
TANK (1, PSOURCE, (P1 - P5)/R5I)
```

which substitutes PSOURCE for inlet pressure PI and the expression for the outlet flow FO. The disadvantage of the concatenation approach is the inflexibility in naming convention and also the fact that any resulting symbol must be six characters or less. If three digits are used for the number N, then all concatenated symbols must have no more than three other characters. In practical problems one usually only has to allow for one digit to identify a component or at most two.

6.5 MACRO CALLS

Once a macro has been defined it must be invoked with specific arguments listed for substitution. The first form of call is to embed the macro name in an arithmetic expression. For this form only one output (a single number) should be produced by the macro - this has functional form

$$X = 5.0 * \text{SIN}(\text{DOT}(A, B) / 4.0)$$

where the DOT product macro is embedded in the argument of the SIN function. A and B in this case correspond to the second and third argument of the macro respectively. The output is the first argument.

Alternative form of the call is as a stand alone statement -

$$\text{DOT}(X = A, B)$$

or its exact equivalent

$$\text{DOT}(X, A, B)$$

The equals (=) sign in the first form is to *indicate* to the reader that X is an output. The program determines what are the actual inputs and outputs as it processes the statements produced by the macro, i.e., no error would result if the operator were invoked so

$$\text{DOT}(X, A = B)$$

but it would be a little misleading. Note especially that

$$X = \text{DOT}(A, B)$$

is an assignment statement and the name X will not be substituted for the first argument. Only a single numerical value can be passed across the equals sign of an assignment statement.

On the other hand, consider a matrix integration operator we might write:

$$\text{MATINT}(X, X_D = A, X_{IC})$$

In this set up two entire vectors are the output of the operator and have their values effectively passed across the equals sign.

The substitution of macro arguments is by replacement of the character string forming the argument with the substitutable name. Where expressions are used, the wrong answer can be obtained if parentheses are not placed around the argument; i.e., consider a macro to integrate a difference in flow rate so

```
MACRO ACCUM(TOT, W1, W2, IC)
TOT = INTEG (W1 - W2, IC)
MACRO END
```

At invocation time we use an expression for net flow out so

```
ACCUM(MASS = WIN, WP1 + WP2, MASSIC)
```

which would give the line of code

```
MASS = INTEG(WIN - WP1 + WP2, MASSIC)
```

which isn't really what we wanted, since the second flow WP2 has a plus sign in front of it. The answer is to surround the substitutable name - where operator precedence can cause a problem - with parentheses. The macro above should have been defined by

```
MACRO ACCUM(TOT, W1, W2, IC)
TOT = INTEG(W1 - (W2), IC)
MACRO END
```

and now at substitution the executed statement is

```
MASS = INTEG(WIN - (WP1 + WP2), MASSIC)
```

It's not necessary to parenthesise the first parameter W1, since any expression substitution will give the correct answer. Trouble usually arises when arguments are negated, multiplied or divided by other variables or used as a divisor in the macro definition.

THIS PAGE INTENTIONALLY LEFT BLANK

7. PROGRAM DEBUGGING

One of the more important features of the ACSL language is the availability of tools that assist in pinpointing errors. The first thing is to establish a frame of mind that believes in the existence of errors. It is difficult, in general, for the average user who writes a model definition to believe that there are any errors. However, if somebody else wrote it, you know there would be something wrong. You must accept the fact that *all* programs have at least one error and part of the joy of coming up with a finished product will be in finding it.

As you write the program, prepare the first run for debugging. Set the stop condition (TERMT) for the first run to a small value (typically one communication interval will suffice) so that no time will be wasted calculating the incorrect values. Use the 'D' option (despite any errors)* in the translator so that the program will proceed to uncover as many errors as possible.

The first run through the translator will produce syntax error indications and probably error messages as well. The latter are listed in Appendix E with some further explanation. The translator analyzes each statement in turn and if an error occurs it will indicate this. The way the error is indicated is to write out again the statement in error, including any continuations, with a line of asterisks (*) underneath to indicate the acceptable section. The asterisk should stop just below where the error is located.

Example:

```
X = Y + (SIN(Y.Y))
***SYNTAX ERROR***THE LINE IS LISTED WITH A POINTER TO THE ERROR
X = Y + (SIN(Y.Y))
*****
```

which shows that the period (.) separating the two Y's is not allowed. It should be an asterisk (*) to indicate a multiply. Two points should be noted when these errors are indicated. The first is that only the *first* error in the statement will be indicated. If this error is corrected, it may need a second (or third) run to uncover other problems further into the statement. When you make a correction, take a long hard look at the rest of the statement.

The second is that the line listed may not look like the input text if continuation cards are used. The error listing gives the complete string to be analyzed after the trailing blanks have been squeezed from the end of any cards continued.

Next check for misspelling - variables you may have intended to be the same get keypunched wrongly. Names you intended to change get overlooked. To check these, look at the symbol cross-reference tables listed at the end of the translator output. Any variables listed under 'VARIABLES NOT SPECIFIED IN ANY BLOCK' will be misspellings, constants you forgot to specify, or correct variables that had their name misspelled at the statement defining them. They should have been defined.

Next, take note of any unsatisfied external references from the load map. These will usually correspond to arrays you forgot to declare in an ARRAY statement - without this they look just like functions.

The first run-time command should set up a debug action and usually over the first five or ten derivative evaluations will suffice. Include the following card at run-time:

```
SET NDBUG = 10
```

Alternatively an action can be scheduled that will ensure a debug printout after every START until CLEARED

* See addendum for a description of the local control card sequence and options.

ACTION 'VAR' = 0.0, 'VAL' = 10, 'LOC' = NDEBUG

NOTE: While the system variable NDEBUG is greater than zero the complete set of user variables is printed out and the value of NDEBUG is reduced by one.

This output is probably the most important data to help in debugging; the previous set of tools was merely to ensure you had the mechanics correct - commas in the right place, spellings consistent, etc. This debug output gives you the actual numbers calculated for every one of the state derivatives and intermediate variables. Look at the numbers carefully and check for reasonableness using your knowledge of the system you are trying to model. It is a good idea to start with initial conditions nonzero. If there are too many zero values, the arithmetic calculations can conceal errors. For preference, pick conditions so the derivatives all have a nonzero value which can be checked. Check the values that are listed for the constants. Any that have been preset in a CONSTANT statement and where the decimal point has been left off will be listed as having a value of 0.0. This problem is a very common error. Some arrays may be missing from this printout if they happen to be longer than the integer contained in the system variable MALPRN (maximum array limit for print out). See system variable summary for the default value.

Now the time comes to try the first full run. Plan what significant output variables will enable you to deduce correct model operation. Specify these in an OUTPUT command; increase the termination time and START.

It is at this point that the modeller's skill comes in, in order to rationalize the behavior of the simulation in terms of how the real world system is expected to behave. About the only help that can be offered is that once questionable areas have been uncovered, schedule debug printouts to cover the area of interest so that as much information is recorded as possible. Note that the debug output occurs every derivative evaluation. For Runge-Kutta fourth order integration four derivative evaluations are made for a time step (calculation interval), one at the beginning, two in the middle, and one at the end. Looking at the independent variable it will appear to advance in half-steps, with *two* derivative evaluations taking place each step. An extra evaluation will take place prior to each communication interval or trip through the DYNAMIC section.

7.1 MEANING OF DEBUG PRINT OUT

The debug output is generated by going through the user dictionary which points to all variables in the user common block, listing the values of each one by one. The first fifteen variables are ACSL control variables that are defined as follows (see Figure A4-7 for an example):

- a) T - Real; Independent variable. May have been renamed in a VARIABLE statement
- b) ZTICG - Real; Initial condition on the independent variable
- c) CINT - Real; Current communication interval. May have been renamed by CINTERVAL
- d) ZZIERR - Logical; Variable step error flag; May have been renamed by ERRTAG
- e) ZZNBLK - Integer; Number of DERIVATIVE and DISCRETE blocks in use
- f) ZZI - Integer; Distinguishes pre-initial (=0), START (=1) and CONTIN (=2)
- g) ZZST - Logical; Stop flag set by TERMT operator
- h) ZZFRFL - Logical; First flag set true at first derivative evaluation of every step
- i) ZZICFL - Logical; Initial condition flag set true at first derivative evaluation of every run - immediately after initial conditions have been transferred to states
- j) ZZRNFL - Logical; Reinitialize flag set true by REINIT. Used during initialization (ZZICFL = .TRUE.) and then turned false
- k) ZZNS - Integer array of length number of DERIVATIVE blocks giving number of state variables in each block

- l) MINT - Real array of length number of DERIVATIVE blocks giving minimum integration step size for each block. Name may be changed by global MINTERVAL statement
- m) MAXT - Real array of length number of DERIVATIVE blocks giving maximum integration step size for each block. Name may be changed by global MAXTERVAL statement
- n) NSTP - Integer array of length number of DERIVATIVE blocks giving communication interval divisor for each block. Name may be changed by global NSTEPS statement
- o) IALG - Integer array of length number of DERIVATIVE blocks giving integration algorithm number to be used for each block. Name may be changed by global ALGORITHM statement

Next in the debug print out comes the list of state variables in DERIVATIVE block order and in alphabetical order within each block, with their corresponding derivatives and initial conditions on the same line. If line width (see TCWPRN and HVDPRN) is sufficient (126) the corresponding values of absolute error (XERR) and relative error (MERR) are also listed on the same line. In general the derivatives will all be dummy variables (Z0nnnn form) except for those defined by the INTVC integration operator.

After the states come all the algebraic variables in alphabetical order. Any EQUIVALENCED variables are listed at the end. System variable ZZSEED contains the random number seed variable which will change (depends on machine type) with every call for a new random number. ZZTLXP is a logical variable present in some machine versions to request the reprieve/interrupt capability. If it is set false before the first START, normal system dumps can be obtained if desired.

THIS PAGE INTENTIONALLY LEFT BLANK

8. APPLICATION NOTES

A number of techniques used in simulation models and run control are included in this section.

8.1 PARAMETER SWEEP

It is possible to define a run which consists of a sequence of runs in which a parameter (P) is varied from a low limit (PMN) to a high limit (PMX) by a certain increment between runs (PDL). This can be programmed using the explicit mode and then plots can be made showing a sequence of curves with the parametric variation.

Example:

```
PROGRAM SWEEP
INITIAL
    P = PMN
    L1 .. CONTINUE
END $' OF INITIAL'
DYNAMIC
DERIVATIVE
    { MODEL DEFINITION
    { ... DEPENDS ON P
END $' OF DERIVATIVE'
    TERMT (T.GE.TSTOP)
END $' OF DYNAMIC'
TERMINAL
    CALL LOG
    P = P + PDL
    IF (P. LE. PMX) GO TO L1
END $' OF TERMINAL'
END $' OF PROGRAM'
```

The run-time control cards will be

```
...
PREPAR T, list - -
SET FTSPLT = .TRUE.
START
PLOT Y1, Y2, etc.
```

Ensure the independent variable is the first variable on the PREPAR list and then set FTSPLT (fly back trace suppression on plots) .TRUE.. This signals the plot program to lift the pen when the variable recorded on channel one of the PREPAR list is less than its previous value. At the same time, the symbol character is

bumped. Printer plots, of course, do not need to lift the pen, but the character change means it is a lot easier to pick out the curves for separate runs. Remember each entry from the INITIAL section into the DYNAMIC section will reset the independent variable to its initial value. Single runs can be easily generated, either by setting the maximum value (PMX) to the minimum value

```
SET PMX = PMN
```

or by making the increment very large

```
SET PDL = 1.0E100
```

8.2 PHASE AND GAIN PLOTS

It is often required to determine phase and gain characteristics of a model that is being forced by a sine wave. With this system, the excitation frequency can be varied logarithmically and the phase and gain characteristics determined and plotted as a function of this frequency. First of all, let us establish a name W for frequency. This will be swept from minimum (WMN) to maximum (WMX) by using a geometric progression with multiplier KW (= 1.2 to 1.5).

Example:

```
PROGRAM PHASE AND GAIN
INITIAL
    W = WMN
    L1 .. CONTINUE
END $' OF INITIAL'
DYNAMIC
DERIVATIVE
    { MODEL
END $' OF DERIVATIVE'
    TERMT(. . .)
END $' OF DYNAMIC'
TERMINAL
    PW = ALOG10(W)
    CALL LOG
    W = W*KW
    IF (W.LE.WMX) GO TO L1
END $' OF TERMINAL'
END $' OF PROGRAM.
```

The frequency will be varied geometrically and PW is calculated for the x-axis of the plot to be made - the actual scale will then be logarithmic.

Now to find the phase and gain. Assume that we inject our signal into the equation for the variable F, and want to know the gain and phase between F and the output variable X, i.e.,

$$\frac{X}{F} = G(j\omega)$$

The in-phase (P) and quadrature (Q) components will be given by

$$P = \frac{W}{\pi} \int_{TS}^T X \sin (WT) dt$$

$$Q = \frac{W}{\pi} \int_{TS}^T X \cos (WT) dt$$

where the integration is taken over any complete cycle.

The trick is to start the integration after sufficient time has elapsed so that the initial transients have decayed away and then just integrate over a complete cycle. This can be done by logic within either the DYNAMIC or DERIVATIVE sections and it's usually easier to set the communication interval to force the full cycle integration. The phase of the window chosen is immaterial relative to the drive sine wave - it's only necessary to integrate over whole cycles.

This sketched out technique requires a complete simulation run for each point i.e. the model code cycles from TERMINAL to INITIAL every time a new frequency point has been calculated. Another way of generating frequency response in a single run is described in section nine of Appendix A. While the implementation described there is fairly complicated, it reduces somewhat the time spent for settling which is wasted CPU seconds and also allows more direct control of the phase accuracy calculated.

8.3 SUMMARY OUTPUT

It is often useful to obtain a complete list of all simulation variable values in order to document the state of the simulation. Setting NDEBUG = 1 gives a picture at the very first derivative evaluation but we have found that this is not as useful as a picture obtained at the end of the run. In a final value debug dump, initial conditions are still available in the initial condition arrays but all other variables document the termination condition.

In order to easily obtain this final value listing, incorporate the following code in the TERMINAL section i.e.

```
TERMINAL
...
LOGICAL DUMP $ CONSTANT DUMP = .TRUE.
IF (DUMP) CALL DEBUG
END $' OF TERMINAL'
```

The call to subroutine DEBUG gives the picture but it's important to have it under the control of a logical variable that can be set at run-time in order to turn the output on or off.

8.4 IMPULSE AND STEP RESPONSE

Common methods used to check simulation models are the determination of the response to impulses and steps in the control variables. In most cases it is not necessary to use special operators as these forcing functions can usually be modelled by parameter changes.

An actual impulse is of infinite height and zero time width and so is impossible to generate directly. The effects however are felt at all integrators the impulse is fed to, and result in a unity jump in the output of these

integrators. The easiest way to implement this jump in practice is to apply a value to the integrator initial condition which models receiving the impulse immediately prior to time equals zero. When the simulation program starts to execute it will then follow a solution trajectory in response to this hypothetical impulse.

As an example consider a pendulum model so

$$\text{OMEGA} = \text{INTEG}(-G*\text{SIN}(\text{THETA})/L, \text{OMEGAZ})$$

$$\text{THETA} = \text{INTEG}(\text{OMEGA}, \text{THETAZ})$$

An impulse in force or momentum transferred to the pendulum ball at $T=0$ is modelled by specifying a non-zero value for the angular rate initial condition **OMEGAZ**. This may need some calculation since there are equations governing momentum transfer. These would apply if for example we struck the pendulum bob with a mallet and would correctly be placed in the **INITIAL** section.

If the impulse is applied at times other than $T=0$, then the integral equation must be modified to add in the net integrated impulse or

$$\text{OMEGA} = \text{INTEG}(-G*\text{SIN}(\text{THETA})/L, 0.0) + \text{DLOMEG}$$

Now the variable **DLOMEG** (delta **OMEGA**) is added in and becomes the initial condition on **OMEGA** (since the **INTEG** has an initial condition of zero). In this configuration we change **DLOMEG** and since it is always added to the state variable or output of the **INTEG** operator, **OMEGA** will jump discontinuously when this happens. If **DLOMEG** has been changed it must be reset in the **INITIAL** section prior to the start of each run. It itself becomes an effective state variable. Most cases can be handled by using the initial condition but when true impulses are applied during the simulation run, then the added variable becomes necessary (see the aspirin example in section 12 of Appendix A)

Step responses are a different excitation technique, usually handled by adding a constant (initialized to zero) at a summing junction but often loops must also be broken. A typical requirement is to examine the response of a missile - pitch rate, pitch angle and accelerometer reading - to a step in fin deflection. This would normally be applied at time equals zero and the dynamic response recorded. The trouble here is that a simple change of initial condition on the fin angle integrator is not sufficient since after the model starts, the fin deflection will change due to the dynamics built into the actuator model. The key now is to break the outer loop and prevent this fin motion and the ease whereby this can be done depends somewhat on the actual fin dynamics model itself. A simple fin model is a first order lag with a typical time constant of 5 to 50 msec, so motion can easily be stopped by setting temporarily the time constant to $1.0E30$. This very large value ensures that the output will remain constant irrespective of what the input does. Alternatively there may be a gain between torquer and velocity integrator which can be made zero, so ensuring zero derivatives or a constant output.

For these types of test cases it is important that model parameter values be given symbolic names, preset by **CONSTANT** statements. The symbolic name enters these into the ACSL dictionary and the **CONSTANT** statement is just a preset so values changed won't be changed back as they would if they were set via an assignment statement. It is bad practice to specify numbers within the code sequence, since no name can be assigned to the value and so the value itself is always fixed, requiring a re-edit of the model definition code in order to effect any changes.

As an example, with a first order lag fin model with a time constant of 20 msec we have two choices

a) **REALPL** (**DL** = 0.020, **DLC**, **DLIC**)

b) **CONSTANT TACT** = 0.20

$$\text{REALPL}(\text{DL} = \text{TACT}, \text{DLC}, \text{DLIC})$$

The second form is far better since now we can change the variable **TACT** at run-time by **SET** commands. For step responses we would use

```
SET TACT=10E30, DLIC=-0.010
START
```

which would generate the response for a fixed fin deflection of minus ten milliradians.

8.5 EXTERNALLY DEFINED VARIABLES

It is sometimes necessary to suppress the message relating to undefined variables which will be generated if names are found that never appear on the left hand side of an equals sign. This happens with external FORTRAN subroutines that are communicating with the ACSL program via the dollar sign (\$) in column one and described in section 1.5. In order to tell the ACSL system that they are calculated and at the same time enter the name in the ACSL model dictionary, just mention them in a dummy PROCEDURAL block which can be placed anywhere in the program.

Consider variables XF, YF and ZF which we know to be defined elsewhere. Add the following two lines

```
...
PROCEDURAL (XF, YF, ZF = )
END
...
```

which says that it is known that the variables XF, YF and ZF are defined somewhere. In this we assume that sorting problems are handled separately. Remember there is no check that the inside of a PROCEDURAL block agrees with the input/output list stated on the header and while it can be a useful feature, care is necessary in its use in actual simulation programs.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

ACSL EXAMPLE PROGRAMS

A number of sample problems have been programmed and run using the ACSL simulation system. In all cases the input consisted of two files or sets of cards. The first describes the model under examination and the second is the run-time executive driver. In most cases the model can be considered as being described in parallel, i.e., there is no necessity to have variables calculated prior to their use. The translator will rearrange the program until it is in correct sequence for execution. On the other hand, the run-time executive is definitely sequential and the input is a sequence of commands that tell the model what to do next. At least a START card is necessary to exercise these models.

All the output from these programs has been designed to fit on standard A size paper (eight and a half by eleven) where possible, for ease of reproduction. Normally the full width of the line printer is available (13") but we have restricted the use of this with the exception of line printer plots since reducing the grid size to 50 by 50 increases the coarseness, decreasing significantly the quality of the picture. The plots therefore have been produced on a 100 by 100 grid and the page reduced by 47% for publication.

The listings produced from the OUTPUT command are sized to fit on the terminal where the size is controlled by the ACSL system variable TCWPRN (terminal character width for print out). When this is changed to 72, the output will fit on an 8.5 inch wide sheet. The low volume or display output is at the same time routed to logical unit nine (SET DIS=9) leaving the high volume output on logical unit six, the output of which was used to make the figures. These logical unit assignments are machine type specific and are described in the addenda for a particular installation. The figures were made on a CYBER 175, NOS operating system.

1. LIMIT CYCLE

The equations

$$\begin{aligned}\dot{x} &= y + K x (1 - x^2 - y^2) / \sqrt{x^2 + y^2} \\ \dot{y} &= -x + K y (1 - x^2 - y^2) / \sqrt{x^2 + y^2}\end{aligned}$$

where

$$x(0) = xz; y(0) = yz$$

describe a limit cycle in the xy plane. The limit cycle is a circle of radius 1.0. That is, no matter what initial conditions are imposed on x and y (except $x = y = 0$), $x^2 + y^2 \rightarrow 1$ as $t \rightarrow \infty$

Assume that plotted time histories of x and y are required for t running from 0 to 10 in steps of 0.2 sec. Printed output need only be listed every 1.0 sec.

The coding for this problem is shown in Figure A1-1 which establishes the base model. Line 7 defines the root sum square value SQ as a named item so that it can be listed and plotted. Execution time is also saved by calculating the common subexpression once.

The next three cards specify the integration equations. Line 13 specifies the termination condition (TERMT). The final time, TF, specified in the constant statement is given as slightly less than ten. If it's exactly ten an extra step will be taken since equality of floating point numbers is never exactly obtained. The value of T obtained by summing many small increments will in general be slightly less than the exact value.

The run-time control cards follow - separated from the model description in the batch run by an end of record card (7/8/9 punch). First comes the output statements that determine what is to be listed (Figure

PROGRAM LIMIT CYCLE

```

"-----DEFINE ALL THE PRESET VARIABLES"
CONSTANT      XZ = 0.5          , YZ = 1.0
CONSTANT      K = 0.2          , TF = 9.99
CINTERVAL     CINT = 0.2
"-----GIVE NAME TO RMS VALUE"
SQ           = SQRT(X**2 + Y**2)
"-----ISOLATE COMMON FACTOR"
KK           = (1.0 - SQ**2)/SQ
"-----LIMIT CYCLE EQUATIONS"
X           = INTEG( Y + K*X*KK, XZ)
Y           = INTEG(-X + K*Y*KK, YZ)
"-----DEFINE STOPPING CONDITION"
TERMT(T .GE. TF)

```

END \$ OF PROGRAM "

1.718 CP SECONDS 2056 TABLE SPACE USED 3 TABLE MOVES

```

SPARE
SET TITLE = "LIMIT CYCLE PROBLEM"
S TCWPRN=72,DIS=9 $ " FORCE 3 COLUMN OUTPUT WIDTH "
OUTPUT T,X,Y,SQ,"NCIOUT"=5 $ "DEFINE LIST TO BE PRINTED DURING RUN"
PREPAR T,X,Y,SQ $ "DEFINE LIST TO BE SAVED FOR LATER USE"
START
PLOT X,Y,SQ $ "PLOT AS FUNCTION OF TIME"
PLOT "XAXIS"=X,Y $ "PHASE PLANE PLOT"
SET NPXPPL=60,NGXPPL=12 $ "SQUARE UP PHASE PLANE PLOT"
PLOT Y $ "ASSUMES XAXIS IS LAST VARIABLE EXPLICITLY DEFINED"
SPARE
STOP

```

Figure A1-1. Listing of Limit Cycle Model and Executive Command Cards

SPARE
 ACCUMULATED CP TIME 6.011 SEC. ELAPSED CP TIME 6.011 SEC.
 SET TITLE = "LIMIT CYCLE PROBLEM"
 S TCWPRN=72,DIS=9 \$" FORCE 3 COLUMN OUTPUT WIDTH "
 OUTPUT T,X,Y,SQ,"NCIOUT"=5 \$"DEFINE LIST TO BE PRINTED DURING RUN"
 PREPAR T,X,Y,SQ \$"DEFINE LIST TO BE SAVED FOR LATER USE"
 START

T 0.	X 0.50000000	Y 1.00000000
SQ 1.11803399		
T 1.00000000	X 1.07143048	Y 0.11524377
SQ 1.07761050		
T 2.00000000	X 0.65941072	Y-0.81887087
SQ 1.05136673		
T 3.00000000	X-0.32732337	Y-0.98097465
SQ 1.03414305		
T 4.00000000	X-0.99128128	Y-0.25178710
SQ 1.02275868		
T 5.00000000	X-0.74193795	Y 0.69293307
SQ 1.01519858		
T 6.00000000	X 0.18130838	Y 0.99375825
SQ 1.01016246		
T 7.00000000	X 0.93107118	Y 0.38308500
SQ 1.00680071		
T 8.00000000	X 0.82357251	Y-0.57520097
SQ 1.00455355		
T 9.00000000	X-0.03897837	Y-1.00229241
SQ 1.00305005		
T 10.00000000	X-0.86359260	Y-0.50823121
SQ 1.00204348		

PLOT X,Y,SQ \$"PLOT AS FUNCTION OF TIME"

Figure A1-2. Run-time Control and Output Stream of Limit Cycle Models

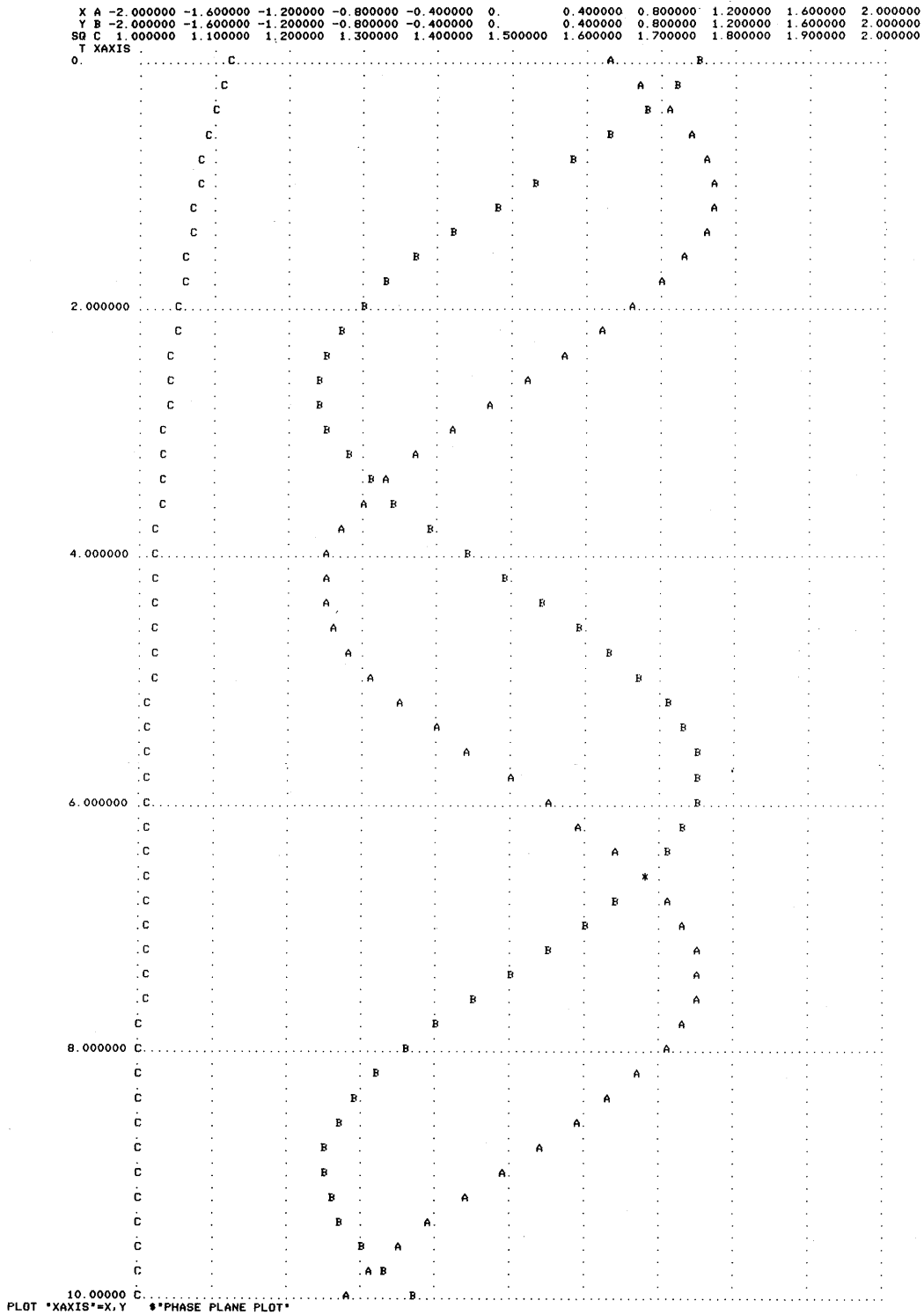
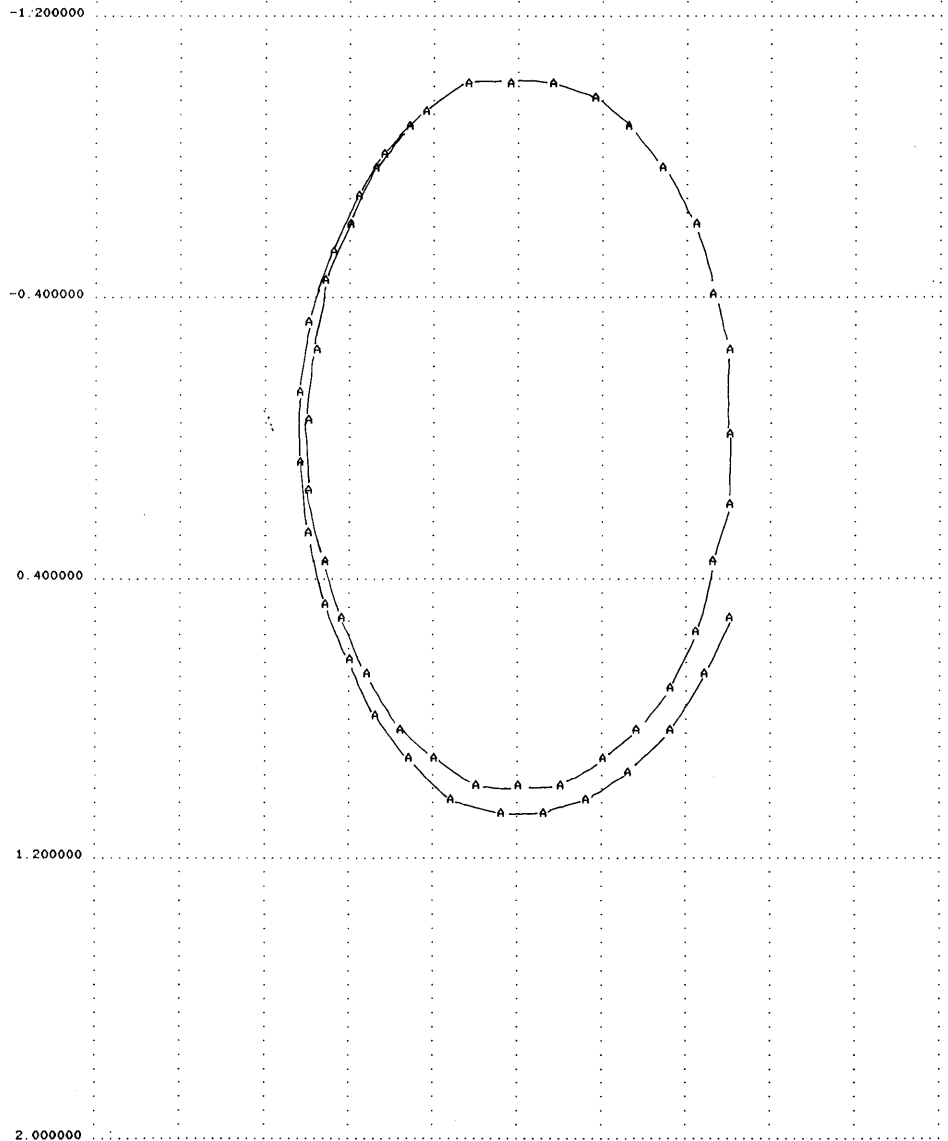


Figure A1-3. Plot Resulting from Last Command on Figure A-2 for Limit Cycle Model

Y A -2.000000 -1.600000 -1.200000 -0.800000 -0.400000 0. 0.400000 0.800000 1.200000 1.600000 2.000000
X AXIS -2.000000



2.000000
SET NPXPPL=60,NGXPPL=12 \$*SQUARE UP PHASE PLANE PLOT*
PLOT Y \$*ASSUMES X AXIS IS LAST VARIABLE EXPLICITLY DEFINED*

Figure A1-4. Plot Resulting from Last Command Figure A-3

Y A -2.000000 -1.600000 -1.200000 -0.800000 -0.400000 0. 0.400000 0.800000 1.200000 1.600000 2.000000
X AXIS
-2.000000

-1.200000

-0.400000

0.400000

1.200000

2.000000

SPARE
ACCUMULATED CP TIME 7.630 SEC. ELAPSED CP TIME 1.619 SEC.
STOP

Figure A1-5. Phase Plane Plot of Figure A-4 with Axes Squared

A1-2) during the run. The qualifier 'NCIOUT' is set to five to indicate output is only to occur every five communication intervals. Then START initiates the run and produces the listing. The next command is PLOT which produces the page plot shown in Figure A1-3. As can be seen, the root sum square value SQ tends to 1.0. A phase plane plot follows (Figure A1-4) where y is plotted against x. Note the distortion of the circle to an ellipse because the printer characters are longer than they are wide (ratio of 10:6). Next the x-axis had been changed by

SET NPXPPL = 60, NGXPPL = 6

(number of points in the x direction for printer plots). The grid is now square and the plot circular (Figure A1-5). Note the second plot does not specify the x-axis variable - the last one used is assumed.

Other values of the constants could have been tested by following this plot card with further SET statements followed by another START.

2. SPRING

A spring supporting a mass with a viscous damper attached can be modelled by a linear second order differential equation. The equation is derived by writing the expression for the force acting on the mass and then using Newton's Law (force equals mass times acceleration) to calculate the acceleration. If x is the linear displacement of the spring, the spring restoring force is -Ax lbs, where A is the spring constant (lbs/ft), and the viscous damping is proportional to the velocity and opposing it or -Kẋ lbs, where K is the coefficient of the viscous friction in lbs/(ft/sec). If W is the weight in lbs attached to the spring, then the mass is W/g slugs where g is the acceleration due to gravity. With this we can express Newton's Law so or:

$$\frac{W}{g} \ddot{x} = W - K\dot{x} - Ax$$

where W, the weight, is the force due to gravity tending to extend the spring. Dividing both sides of the above equation by W/g we obtain the expression for the highest derivative or:

$$\ddot{x} = (W - K\dot{x} - Ax) / (W/g)$$

Integrating this twice leads to the two statements:

$$\dot{x} = \int \ddot{x} dt$$

$$x = \int \dot{x} dt$$

Figure A2-1 shows the ACSL model definition statements that represent the dynamics of the mass when it is released. Note the alternate way of obtaining time by integrating a constant one. This is useful if it becomes necessary to change the initial condition on time. The run-time drive cards are shown in Figure A2-2 and output listings in Figure A2-3 through A2-6. At the bottom of Figure A2-4 the plot is set up with a specified Y-axis scale factor and Figure A2-5 shows the resulting limited plot. Out of bounds points are brought back to the edge of the plot with no attempt made to interpolate for correct intersection with the edge. Beware that the slope of the curve as it approaches the limit condition will therefore be incorrect.

The communication interval was defined in the model definition section to be 0.02 seconds and with a run time of 4 seconds this will save 201 data points. Since the x-axis size is a hundred points, two data points will be available for plotting in each x-axis box. When they fall in different y-axis boxes, the two points are shown on the same vertical line - see Figure A2-4 -, but this is only an artifact of the coarse plotting medium. When the rates of change are slow enough, the two points will coalesce into a single point as shown at the extrema of the curves.

3. CONTROL LOOP

This program was chosen to illustrate how MACROS are used to represent transfer functions from a system block diagram. The problem is the design of a lead-lag controller for a second order plant that has

```

PROGRAM SPRING
DERIVATIVE
"-----SPRING DAMPING PROBLEM.MODELS RELEASING"
"          A MASS FROM INITIAL CONDITIONS OF ZERO"
"          VELOCITY AND DISPLACEMENT"
CINTERVAL      CINT = 0.02
"-----DEFINE PRESET VARIABLES"
CONSTANT      XIC = 0.0      , XDIC = 0.0      , W = 1.0      ...
              , K = 0.02     , A = 1.0       , G = 32.2     ...
              , TSTP = 3.99
"-----ANOTHER WAY OF CHANGING THE INDEPENDENT"
"          VARIABLE"
TIME          = INTEG(1.0, 0.0)
XDD           =(W - K*XD - A*X)/(W/G)
"-----INTEGRATE ACCEL FOR VELOCITY AND POSITION"
XD           = INTEG(XDD, XDIC)
X            = INTEG(XD , XIC )
"-----SPECIFY TERMINATION CONDITION"
TERMT(T.GE.TSTP)
END $" OF DERIVATIVE "
END $" OF PROGRAM "
  
```

Figure A2-1. Listing of Model Definition for Spring Damping Problem

```

SET TITLE = "SPRING DAMPING PROBLEM"
S TCMPRN=72,DIS=9 $" FORCE 3 COLUMN OUTPUT WIDTH "
OUTPUT TIME,XDD,XD,X,"NCIOUT"=20
PREPAR XDD,XD,TIME,X
START
RANGE "ALL"
PLOT "XAXIS" = TIME      $" SET TIME AS X-AXIS FOR SUBSEQUENT PLOTS"
PLOT X,XD      $" USE AUTOMATIC SCALING"
PLOT X,"HI"=1.0,"LO"=0.0,XD
PLOT "XAXIS"=XD, X $" PHASE PLANE PLOT"
STOP
  
```

Figure A2-2. Run-time Drive Commands to Exercise Spring Damping Problem

SET TITLE = "SPRING DAMPING PROBLEM"
 S TCWPRN=72,DIS=9 \$" FORCE 3 COLUMN OUTPUT WIDTH "
 OUTPUT TIME,XDD,XD,X,"NCIOUT"=20
 PREPAR XDD,XD,TIME,X
 START

TIME 0. X 0.	XDD 32.2000000	XD 0.
TIME 0.4000000 X 1.52486284	XDD-19.3714113	XD 3.83668940
TIME 0.8000000 X 1.18166762	XDD-3.06640990	XD-4.32187508
TIME 1.2000000 X 0.38969003	XDD 18.4264342	XD 1.90302322
TIME 1.6000000 X 1.54707775	XDD-18.3865897	XD 1.19671746
TIME 2.0000000 X 0.85545031	XDD 6.46987709	XD-2.81890860
TIME 2.4000000 X 0.73999060	XDD 6.92302052	XD 2.25043816
TIME 2.8000000 X 1.40461313	XDD-12.7990980	XD-0.35628069
TIME 3.2000000 X 0.74518210	XDD 9.06683251	XD-1.33803765
TIME 3.6000000 X 0.97431479	XDD-0.32094098	XD 1.78261606
TIME 3.9920000 X 1.23345847	XDD-6.92515695	XD-0.91957557

RANGE "ALL"

XDD-27.0956534	32.2000000
XD-4.35499524	5.20222033
TIME 0.	3.99200000
X 0.	1.83607297

PLOT "XAXIS" = TIME \$" SET TIME AS X-AXIS FOR SUBSEQUENT PLOTS"
 PLOT X,XD \$" USE AUTOMATIC SCALING"

Figure A2-3. Run-time Control Card and Output Stream - Spring Damping Problem

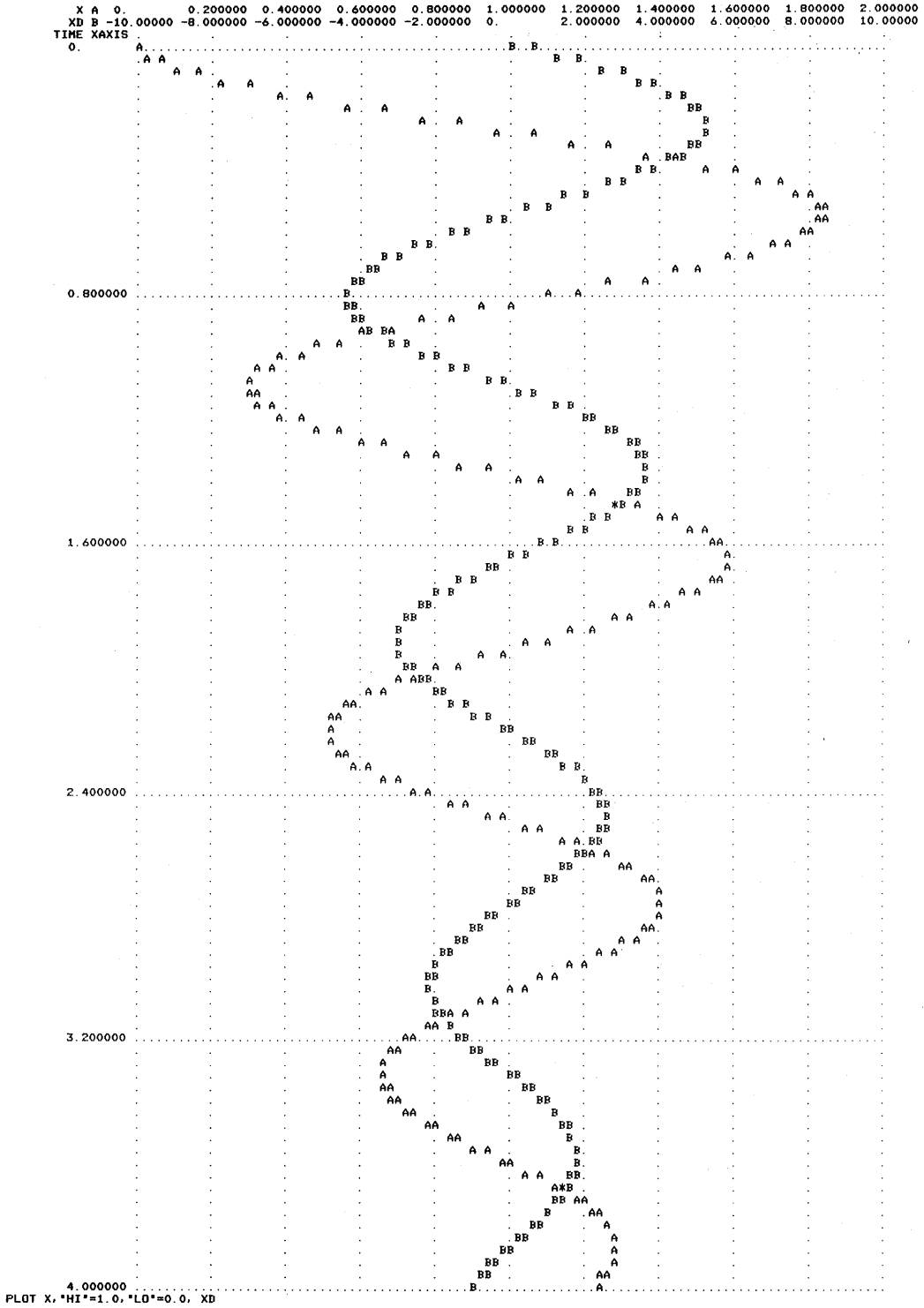


Figure A2-4. Plot with Automatic Scale Factor Selection - Spring Damping Problem

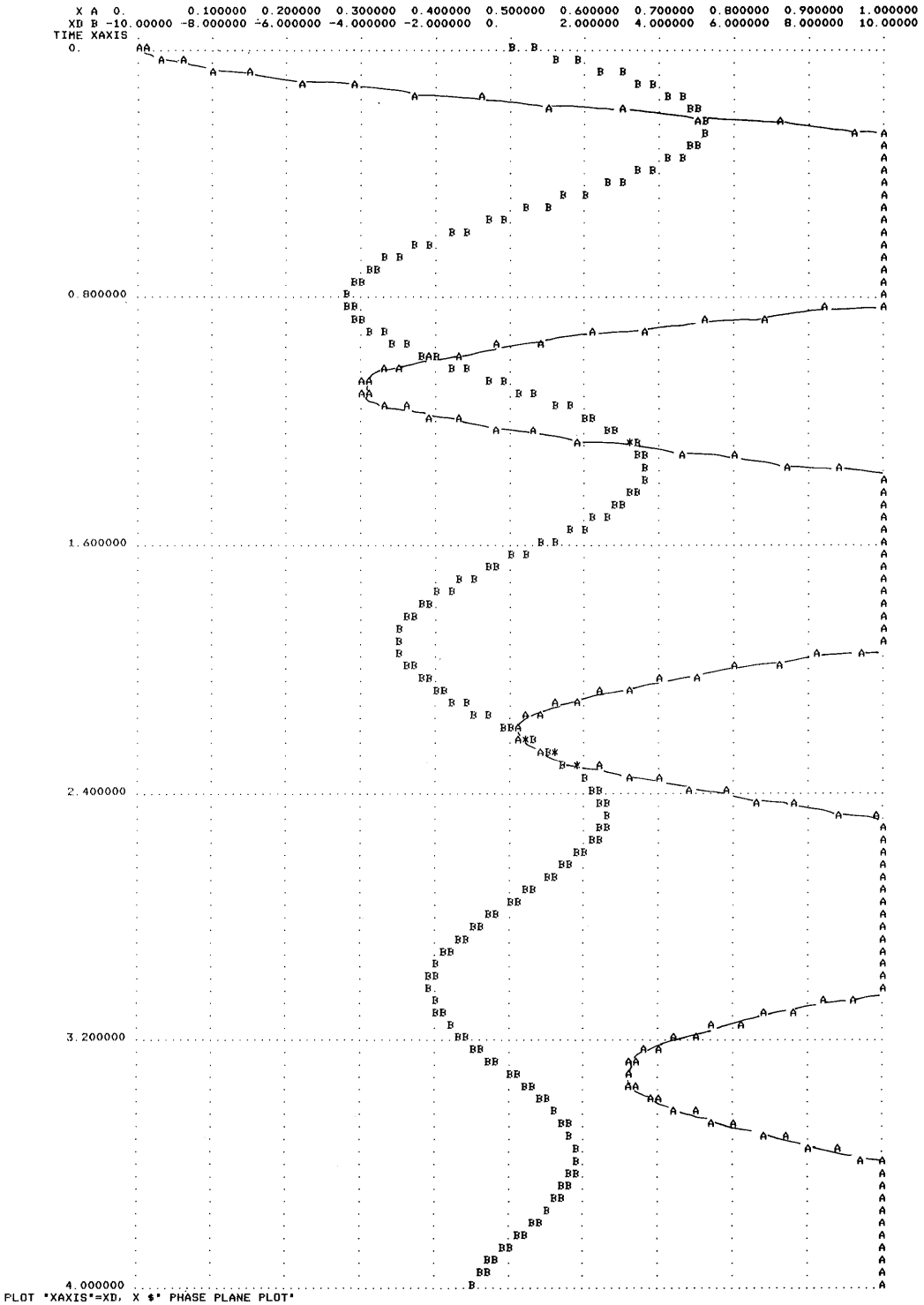


Figure A2-5. Forced Off-Scale Plot by Specifying Y-Axis Scale Factor for X-Spring Damping Problem

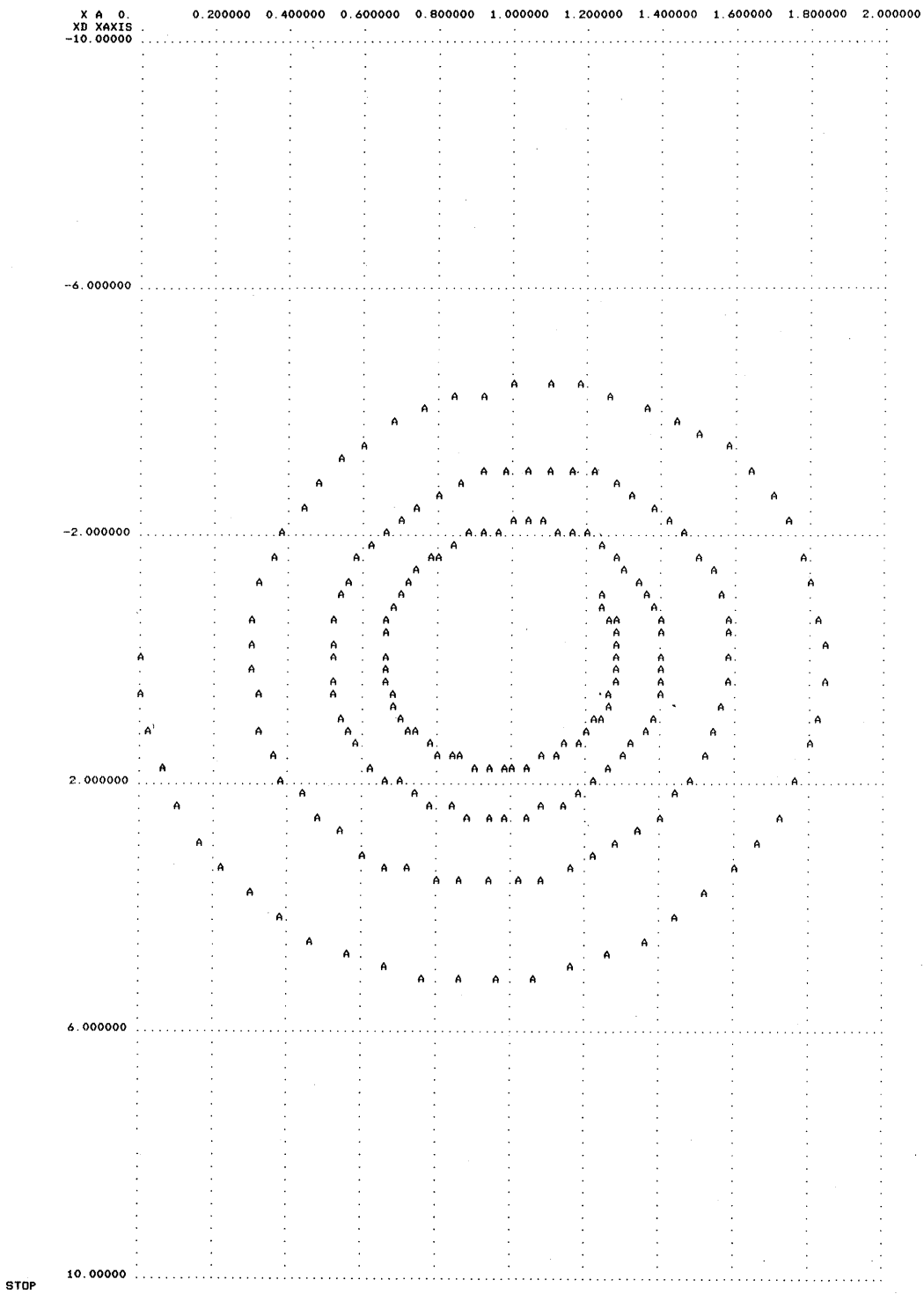


Figure A2-6. Phase Plane Plot - Spring Damping Problem

a measurement device containing a first order lag (real pole). Figure A3-1 shows the system block diagram. Constants in the model are as follows:

$$\begin{array}{ll} T_1 = 0.020 \text{ sec} & B = 0.200 \text{ sec} \\ T_2 = 0.005 \text{ sec} & K_1 = \text{to be determined} \\ T_3 = 0.002 \text{ sec} & K_2 = 0.5 \\ A = 0.012 \text{ (sec)}^2 & K_3 = 1.0 \end{array}$$

The model definition is shown in Figure A3-2 and the run-time drive sequence in Figure A3-3. In the model definition, the communication interval (CINT) is defined to be 5msec, a step in input is applied at 20msec (TZ) and the transient is allowed to run to the stop time (TSTP) of 499msec or 100 recorded data points. In the model listings, the transfer function operators, REALPL, CMPXPL and LEDLAG are embedded in the right hand side expressions. This is acceptable when the output is a single numeric quantity and in fact these can be nested to any depth desired. An alternate form is available for stand alone use if the operator is known to be a MACRO as follows for the three lines calculating XM, X and XP respectively.

```
REALPL (XM = TA3, K3*X, 0.0)
CMPXPL (X = A, B, K2*XP)
LEDLAG (XP = TA1, TA2, K1*E, 0.0)
```

This form of the MACRO invocation is sometimes preferable since it restricts generation of dummy names (or those starting with Z0 . . .). In changing to the above form we have used the property of linear operators that pre and post multiplication by scalars are equivalent.

In the run-time drive commands (Figure A3-3) the OUTPUT and PREPAR lists are specified and a procedure GO defined. The card images between the PROCED . . . END are saved and not executed. Then two runs are made using the procedure now as a new command, the first with K1 equals to 100.0 and the second 10.0. Each 'GO' invokes the START/PLOT sequence saved in the procedure. Figures A3-4 through A3-7 show the output stream generated as a result of the run-time drive commands. Notice that when the commands within the procedure are echoed back a trailing dollar sign (\$) is appended.

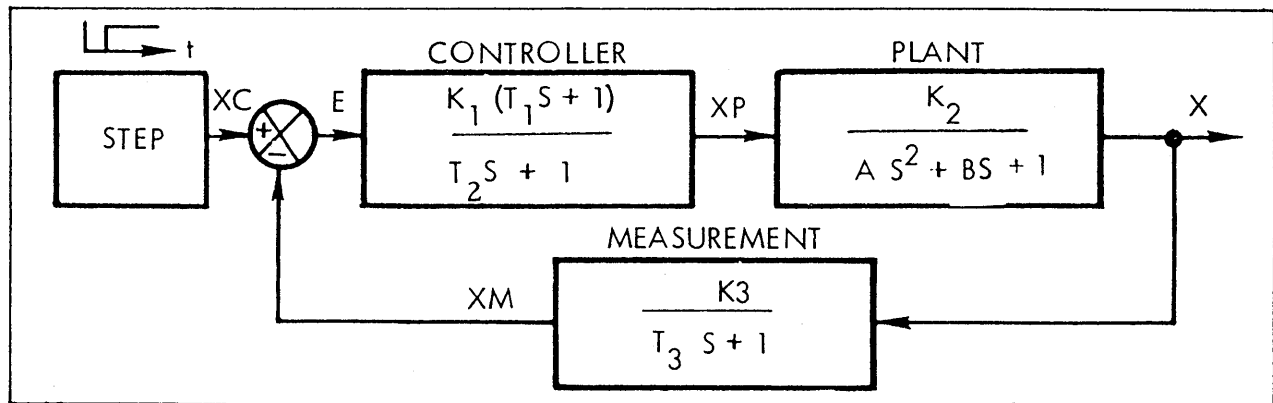


Figure A3-1. Control Loop Problem Block Diagram

```

PROGRAM LOOP
  "-----DEFINE PRESET VARIABLES"
  CONSTANT      K1 = 50.0      , K2 = 0.5      , K3 = 1.0      ...
                , TSTP = 0.499 , TA1 = 0.020   , TA2 = 0.005   ...
                , TA3 = 0.002  , A = 0.012    , B = 0.200    ...
                , TZ = 0.02
  CINTERVAL     CINT = 0.005
  "-----OUTPUT OF FIRST ORDER LAG IS MEASUREMENT"
  XM           = K3*REALPL(TA3, X, 0.0)
  "-----FORCING FUNCTION"
  XC           = STEP(TZ)
  E            = XC - XM
  "-----DEFINE 2-ND ORDER PLANT"
  X            = K2*CMPLXPL(A, B, XP, 0.0, 0.0)
  "-----CONTROLLER OUTPUT"
  XP           = K1*LEDLAG(TA1, TA2, E, 0.0)
  "-----SPECIFY TERMINATION CONDITION"
  TERMT(T.GE.TSTP)
END $" OF PROGRAM "
```

Figure A3-2. Listing of Control Loop Model Definition

```

SET TITLE = "CONTROL LOOP PROBLEM"
S TCWPRN=72,DIS=9 $" FORCE 3 COLUMN OUTPUT WIDTH "
OUTPUT T,XC,E,XP,X,"NCIOUT"=10
PREPAR T,XC,E,XP,X, XM
PROCED GO
START
"FORCE SAME SCALES FOR COMMAND AND MEASURED - FIRST VARIABLE IS REFNCE"
PLOT X,XC, XM, "SAME", E
END $" OF PROCEDURE "
"MAKE TWO RUNS AND PRODUCE PLOTS OF EACH"
SET K1=100.0 $ GO
SET K1=10.0 $ GO
STOP
```

Figure A3-3. Run-time Drive Commands for Control Loop Problem

```

SET TITLE = "CONTROL LOOP PROBLEM"
S TCWPRN=72,DIS=9 $ " FORCE 3 COLUMN OUTPUT WIDTH "
OUTPUT T,XC,E,XP,X,"NCIOUT"=10
PREPAR T,XC,E,XP,X,XM
PROCEED GO
START
"FORCE SAME SCALES FOR COMMAND AND MEASURED - FIRST VARIABLE IS REFNCE"
PLOT X,XC,XM,"SAME",E
END $ " OF PROCEDURE "
"MAKE TWO RUNS AND PRODUCE PLOTS OF EACH"
SET K1=100.0 $ GO
START$

```

T	XC	E
0.	0.	0.
0.05000000	1.00000000	-0.34243054
0.10000000	1.00000000	0.06252934
0.15000000	1.00000000	0.01896207
0.20000000	1.00000000	0.01921891
0.25000000	1.00000000	0.01964535
0.30000000	1.00000000	0.01960870
0.35000000	1.00000000	0.01960740
0.40000000	1.00000000	0.01960787
0.45000000	1.00000000	0.01960785
0.49950000	1.00000000	0.01960784

```

"FORCE SAME SCALES FOR COMMAND AND MEASURED - FIRST VARIABLE IS REFNCE"$
PLOT X,XC,XM,"SAME",E$

```

Figure A3-4. Run-time Drive Commands and Output Stream of Control Loop Problem

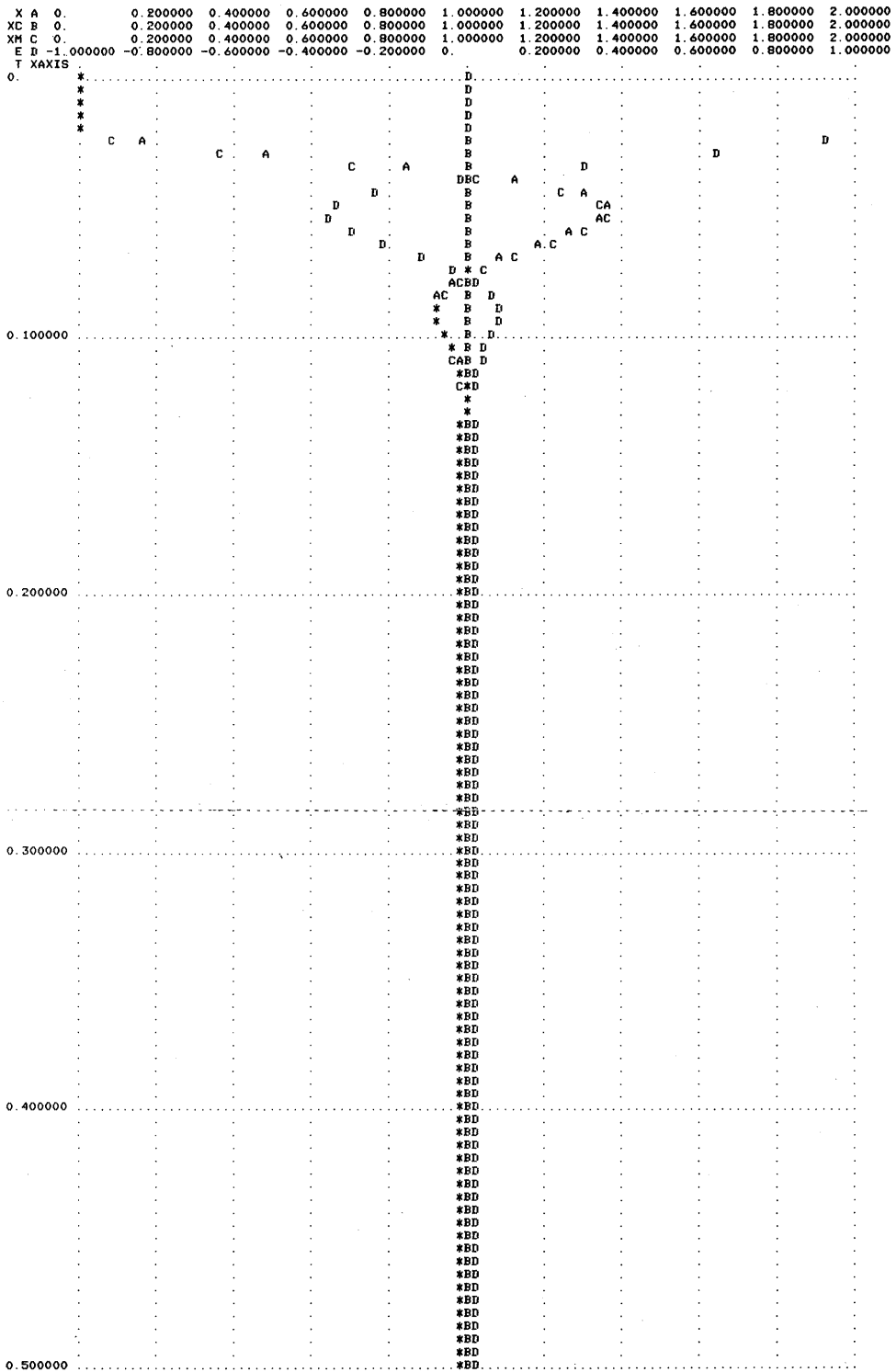


Figure A3-5. Plot of Input Step, XC, Output, X, and measurement, XM, to Same Scales with Error, E, for Control Loop Problem

SET K1=10.0 \$ GO
START\$

T 0.	XC 0.	E 0.
XP 0.	X 0.	
T 0.05000000	XC 1.00000000	E 0.75834817
XP 6.00718267	X 0.26483287	
T 0.10000000	XC 1.00000000	E 0.23491640
XP 1.07926131	X 0.78064589	
T 0.15000000	XC 1.00000000	E 0.02168535
XP-0.03417260	X 0.98055897	
T 0.20000000	XC 1.00000000	E 0.05569507
XP 0.80350815	X 0.94077047	
T 0.25000000	XC 1.00000000	E 0.14405774
XP 1.67195084	X 0.85305345	
T 0.30000000	XC 1.00000000	E 0.18779721
XP 1.94272303	X 0.81153849	
T 0.35000000	XC 1.00000000	E 0.18684210
XP 1.83522449	X 0.81366126	
T 0.40000000	XC 1.00000000	E 0.17242946
XP 1.68335989	X 0.82809089	
T 0.45000000	XC 1.00000000	E 0.16384387
XP 1.62368836	X 0.83631791	
T 0.49950000	XC 1.00000000	E 0.16308868
XP 1.63462643	X 0.83684777	

FORCE SAME SCALES FOR COMMAND AND MEASURED - FIRST VARIABLE IS REFNCE\$
PLOT X, XC, XM, *SAME*, E\$

Figure A3-6. Second Run - Control Loop Problem

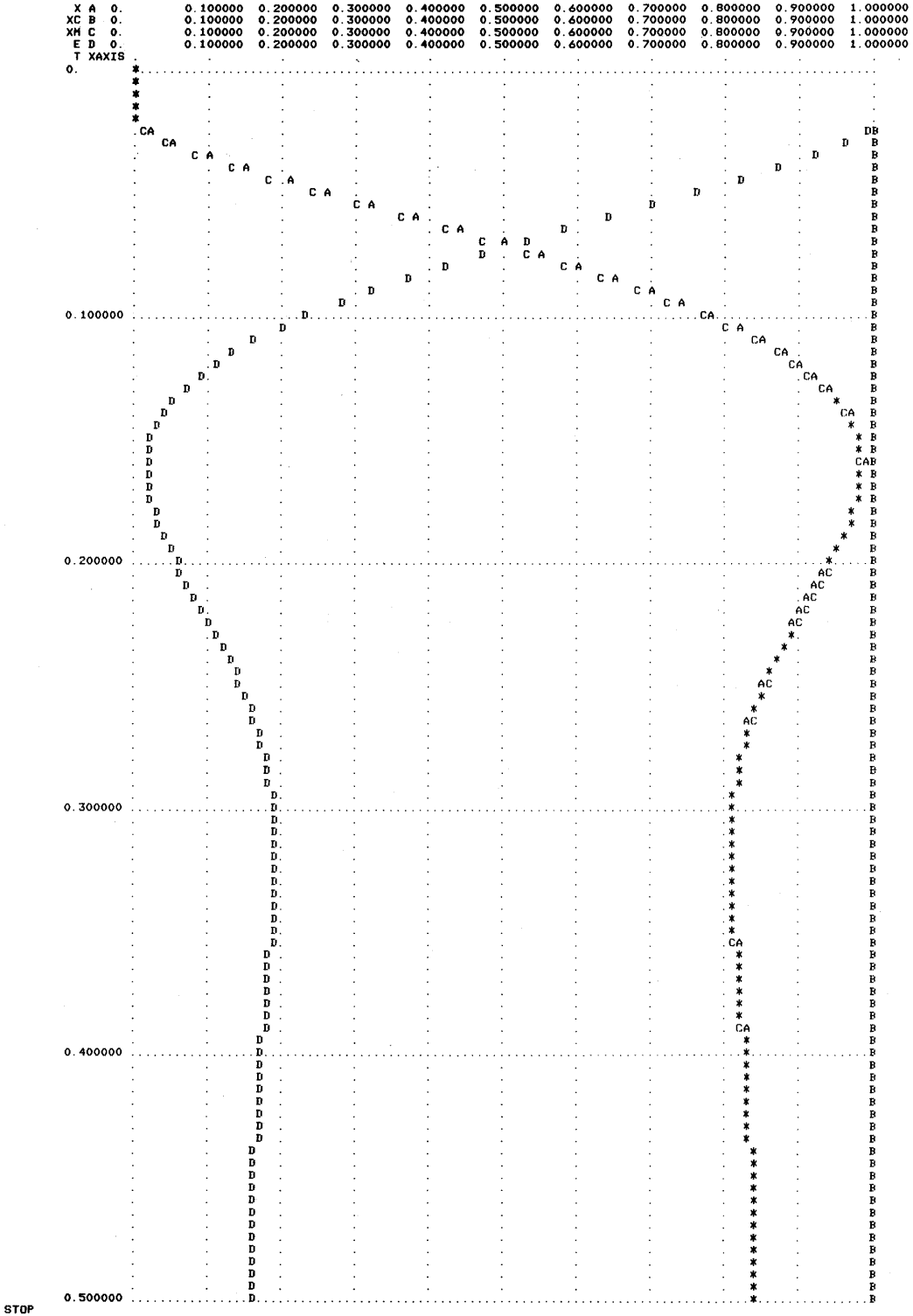


Figure A3-7. Plot of Step, Output and Error for Second Run - Control Loop Problem

4. PILOT EJECTION STUDY

The purpose of this investigation is to determine the trajectory of a pilot ejected from a fighter aircraft in order to ascertain whether he will strike the vertical stabilizer of the aircraft. Several combinations of aircraft speed and altitude will be investigated since the drag on the pilot, causing his relative horizontal motion with respect to the aircraft, is a function of air density and velocity (squared). The ejection system is devised so that it causes the pilot and his seat to travel along rails at a specified exit velocity, V_E , at an angle, θ_E , backward from vertical. The seat becomes disengaged from the rails at $Y = Y_1$. This first phase of the ejection is illustrated in Figure A4-1. Once the pilot and seat combination leaves the rails, it follows a ballistic trajectory which can be determined; however, since it is the *relative* motion of the pilot with respect to the aircraft (which is assumed to fly level at constant speed) that is important, we can formulate our equations to obtain this trajectory directly. This phase of the ejection is shown in Figure A4-2.

The governing equations are:

$$\begin{aligned} \dot{X} &= V \cos \theta - V_A \\ \dot{Y} &= V \sin \theta \\ \dot{V} &= 0 \end{aligned} \quad 0 \leq Y < Y_1$$

$$= \frac{-D}{M} - g \sin \theta \quad Y \geq Y_1$$

$$\begin{aligned} \dot{\theta} &= 0 \\ &= - (g \cos \theta) / V \end{aligned} \quad \begin{aligned} 0 \leq Y < Y_1 \\ Y \geq Y_1 \end{aligned}$$

$$D = \frac{1}{2} \rho C_D S V^2$$

Two cases will be run, viz:

Case 1: $V_A = 900$ ft/sec

$$\rho = 2.3769 \times 10^{-3} \text{ slugs/ft}^3 \text{ (sea level)}$$

Case 2: $V_A = 500$ ft/sec

$$\rho = 2.3769 \times 10^{-3} \text{ slugs/ft}^3 \text{ (sea level)}$$

Constants (for all cases)

$$m = 7 \text{ slugs} \quad g = 32.2 \text{ ft/sec}^2$$

$$C_D = 1 \quad S = 10 \text{ ft}^2$$

$$Y_1 = 4 \text{ ft} \quad V_E = 40 \text{ ft/sec}$$

$$\theta_E = 15 \text{ deg} (= 15/57.3 \text{ rad})$$

The initial values of V and θ (pilot's initial velocity vector at moment of leaving cockpit rails) are given by:

$$V(0) = (V_A - V_E \sin \theta_E)^2 + (V_E \cos \theta_E)^2$$

$$\theta(0) = \tan^{-1} \left[\frac{V_E \cos \theta_E}{V_A - V_E \sin \theta_E} \right]$$

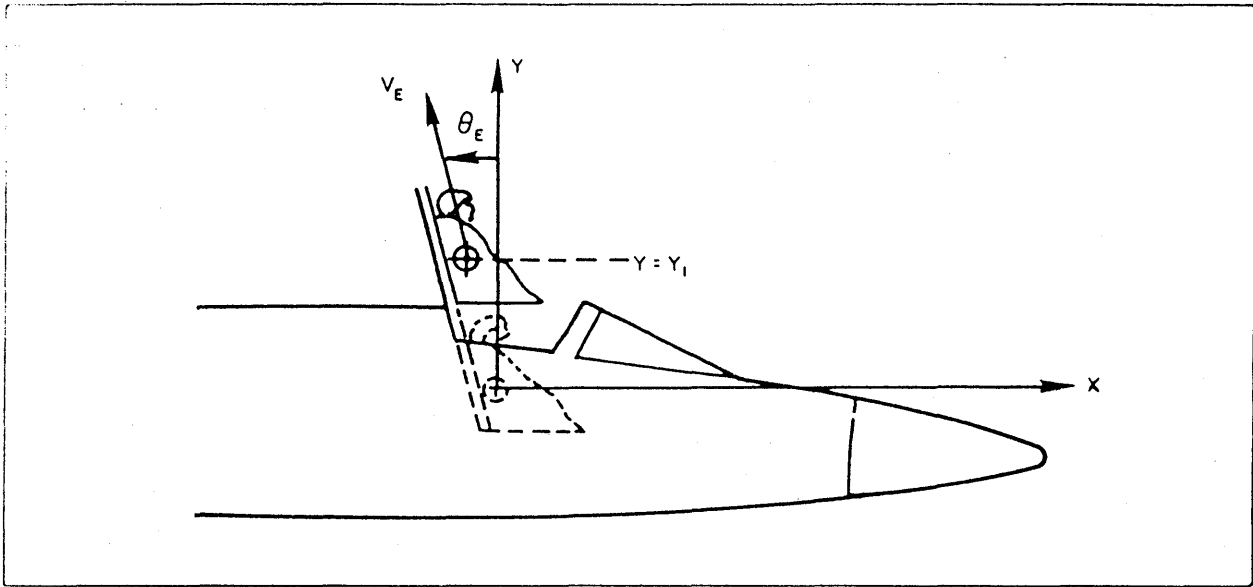


Figure A4-1. First Phase of Ejection

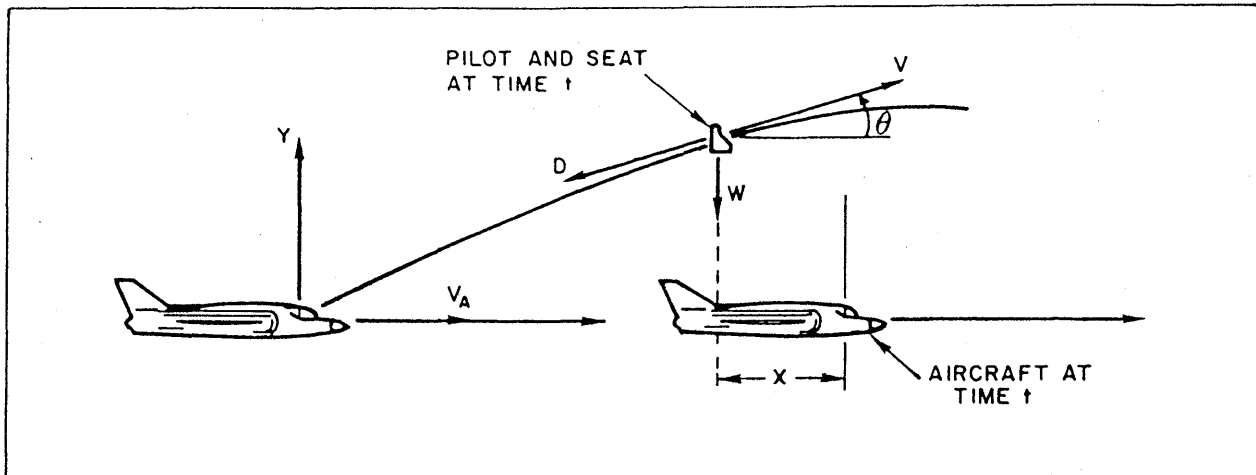


Figure A4-2. Space Trajectories, Vehicle and Pilot

$$X(0) = Y(0) = 0$$

A run is to be terminated when any one of these conditions occurs:

$X \leq -60$ ft (pilot beyond vertical stablizer)

$Y \geq 30$ ft (pilot well above 12 ft high tail)

$t \geq 4.0$ sec

Print t , V , \dot{V} , θ , X , and Y every 0.01 sec

Plot X versus Y

Plot X , Y , and V versus t

The coding for this example is given in Figure A4-3 with comments as follows. A procedural is defined to calculate the switch YGEI (Y greater than or equal to one). Note that the initial conditions on the pilot's velocity VIC and flight path THIC are calculated in the initial section. They must not be calculated in the derivative section since the initial condition table is transferred to the state table before the derivative section is evaluated. The terminate condition of X less than -60 ft or Y greater than 30 ft or time greater than 4 sec is placed in the DYNAMIC section to be interrogated every communication interval.

The output stream from running the model is shown in Figure A4-4 through A4-8. We have not separately listed the run-time drive cards since they are echoed as part of the normal output stream. In Figure A4-4, after establishing the title, the OUTPUT and PREPAR lists are defined and the first run is made (START) using the default parameter values defined in the model definition or aircraft velocity of 900 ft/sec. Output values are listed during the run every five communication intervals or 0.05 sec. The run stops at 0.44 seconds since x has become more negative than the minimum specified or -60 ft. The first plot produces time histories, where the x-axis variable is T, and then the trajectory plot with the x-axis taken as X, the relative distance of the pilot along the aircraft from the cockpit, negative towards the tail. At 60 ft. the pilot just clears the 12 ft. high tail. Note that this plot specifies 'XLO'=XMN or -60 ft. so that the plot runs from this value to zero. If the XLO subcommand had not been used, normal rounding would have caused the scales to run from -100 to zero, so wasting 40% of the plotting area.

In Figure A4-7, a second run is set up with a debug print out at the derivative evaluation with T equal to or greater than 0.1: The output rate is reduced to every ten communcation intervals or 0.1 sec but the same list is maintained: The aircraft speed is changed to 500 ft/sec and the model run again.

The debug output is obtained by using the ACTION command which is read as: When the independent variable ('VAR') is 0.1 take a value ('VAL') of 1 and store it into a location ('LOC') called NDBUG. The printout following the START card has the normal listing at the time equal to 0.0 and 0.1 (note the reduced frequency) and then all the variables in the model definition are listed out by the debug operation since the ACTION has made NDBUG positive. The time is actually at 0.101 sec which means that T was 0.099999999999 rather than 0.1 at the previous integration step. The order of the list is system variables (described in more detail in Section 7), state variables with their associated derivatives and initial conditions, followed by all the rest of the problem variables in alphabetical order.

Since the ACTION statement changed the system variable NDBUG to one, only one debug list is written out and the run continues normally from that point with output every 0.1 seconds. The plot of the second run, Figure A4-8, shows the pilot now clearing the tail by 5 ft. at the lower velocity.

PROGRAM EJECTION

INITIAL

```

*-----DEFINE ALL PRESET VARIABLES *
CONSTANT      THEDEG = 15.0      , DEGRAD = 57.3
CONSTANT      MASS = 7.0        , Y1 = 4.0
CONSTANT      CD = 1.0          , S = 10.0
CONSTANT      G = 32.2          , RO = 0.0023769
CONSTANT      VE = 40.0         , VA = 900.0
CONSTANT      XMN = -60.0        , YMX = 30.0
CONSTANT      TMX = 4.0
CINTERVAL     CINT = 0.01
  
```

-----EJECTION ANGLE IN RADIANS

THE = THEDEG/DEGRAD

-----SEAT INITIAL VELOCITY

VX = VA - VE*SIN(THE)

VY = VE*COS(THE)

VIC = SQRT(VX**2 + VY**2)

THIC = ATAN2(VY, VX)

END \$* OF INITIAL *

DYNAMIC

DERIVATIVE

-----RELATIVE POSITIONS

X = INTEG(V*COS(TH) - VA, 0.0)

Y = INTEG(V*SIN(TH), 0.0)

-----SPACE VELOCITY AND FLIGHT PATH ANGLE

V = INTEG(YGE1*(-D/MASS - G*SIN(TH)), VIC)

TH = INTEG(YGE1*(-G*COS(TH)/V), THIC)

-----COMPUTE DRAG

D = 0.5*RO*CD*S*V**2

-----USE PROCEDURAL FOR SWITCH TO KEEP SEAT

* CONSTRAINED TO GUIDE RAILS. THIS OPERATION IS BETTER DONE BY - *

* YGE1 = RSW(Y .GE. Y1, 1.0, 0.0) *

* BUT IS SHOWN HERE TO DEMONSTRATE USE OF A PROCEDURAL BLOCK *

PROCEDURAL(YGE1 = Y, Y1)

YGE1 = 1.0

IF(Y.LT.Y1) YGE1 = 0.0

END \$* OF PROCEDURAL *

END \$* OF DERIVATIVE *

-----SPECIFY TERMINATION CONDITIONS

TERMT(X.LE.XMN .OR. Y.GE.YMX .OR. T.GE.TMX)

END \$* OF DYNAMIC *

END \$* OF PROGRAM *

Figure A4-3. Listing of Model Definition Section for Pilot Ejection Problem

SET TITLE = "PILOT EJECTION"
 S TCWFRN=72,DIS=9 \$" FORCE 3 COLUMN OUTPUT WIDTH "
 OUTPUT T,TH,V,X,Y,D,"NCIOUT"=5
 PREPAR T,TH,V,X,Y
 START

T 0.	TH 0.04340252	V 890.486592
X 0.	Y 0.	D 9424.00882
T 0.05000000	TH 0.04340252	V 890.486592
X-0.51760084	Y 1.93186163	D 9424.00882
T 0.10000000	TH 0.04340252	V 890.486592
X-1.03520168	Y 3.86372327	D 9424.00882
T 0.15000000	TH 0.04167638	V 832.328907
X-2.92272617	Y 5.70253371	D 8233.24182
T 0.20000000	TH 0.03967529	V 777.339784
X-7.74560503	Y 7.33859116	D 7181.29398
T 0.25000000	TH 0.03753729	V 729.161781
X-15.1368159	Y 8.79210784	D 6318.71416
T 0.30000000	TH 0.03526236	V 686.603725
X-24.7874576	Y 10.0801944	D 5602.64656
T 0.35000000	TH 0.03285045	V 648.737032
X-36.4412100	Y 11.2170486	D 5001.70754
T 0.40000000	TH 0.03030154	V 614.827136
X-49.8830398	Y 12.2145814	D 4492.48846
T 0.44000000	TH 0.02816375	V 590.148541
X-61.8005731	Y 12.9191282	D 4139.07780

PLOT "XAXIS"=T, TH, V, X, Y \$"TIME HISTORIES"

Figure A4-4. Output Stream for Pilot Ejection Study

TH A	0.010000	0.014000	0.018000	0.022000	0.026000	0.030000	0.034000	0.038000	0.042000	0.046000	0.050000
V B	500.0000	550.0000	600.0000	650.0000	700.0000	750.0000	800.0000	850.0000	900.0000	950.0000	1000.0000
X C	-100.0000	-90.00000	-80.00000	-70.00000	-60.00000	-50.00000	-40.00000	-30.00000	-20.00000	-10.00000	0.00000
Y D	0.000000	2.000000	4.000000	6.000000	8.000000	10.00000	12.00000	14.00000	16.00000	18.00000	20.00000

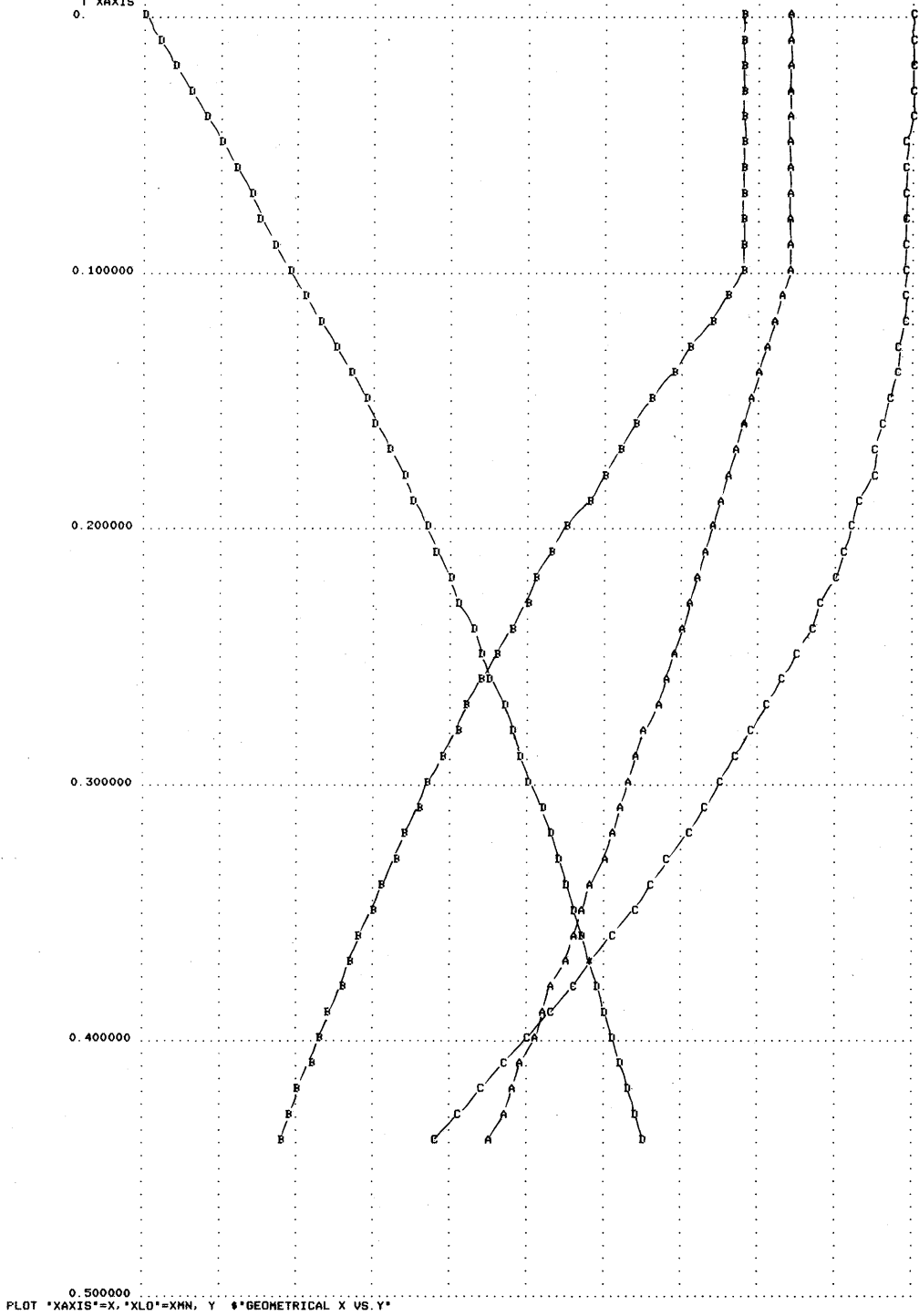


Figure A4-5. Time Plot - Pilot Ejection Study

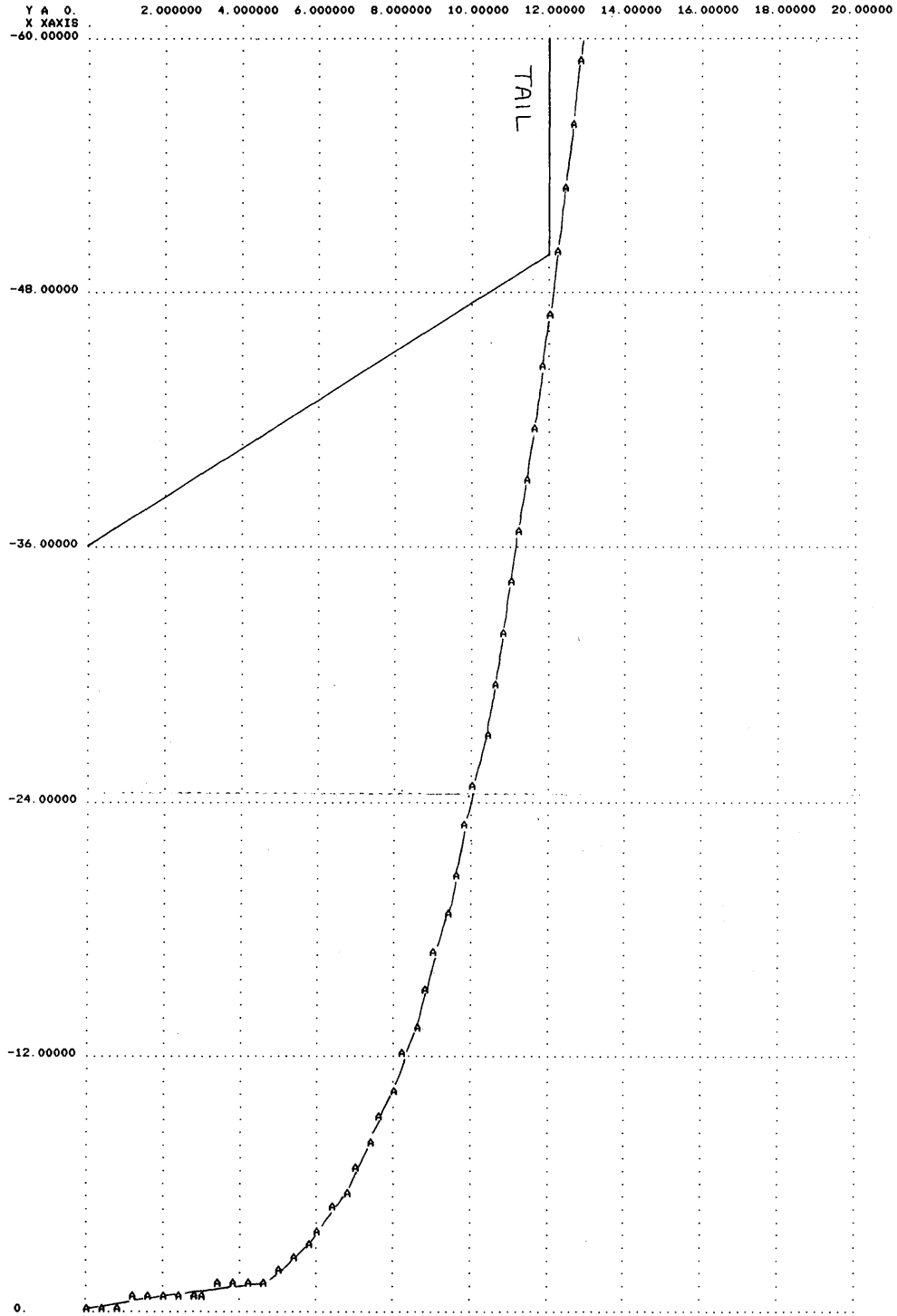


Figure A4-6. Trajectory Plot - Pilot Ejection Study

"SET UP FOR DEBUG LIST OF ALL VARIABLES WHEN T = 0.1 "
 ACTION "VAR" = 0.1, "VAL" = 1, "LOC" = NDBUG
 OUTPUT "NCIOUT"=10 \$ " SAME OUTPUT LIST, LOWER RATE "
 " LIST OF COMMANDS ON ONE CARD "
 SET VA = 500.0 \$ START \$ SET DIS=9 \$ PLOT "XAXIS" = X, "XLO" = XMN, Y
 T 0. TH 0.07874502 V 491.170015
 X 0. Y 0. D 2867.11166

 T 0.10000000 TH 0.07874502 V 491.170015
 X-1.03520168 Y 3.86372327 D 2867.11166

.... DEBUG DUMP - SYSTEM VARIABLES. NDBUG IS 1		
T 0.10100000	ZZTICG 0.	CINT 0.01000000
ZZIERR F	ZZNBLK 1	ZZI 1
ZZST F	ZZFRFL T	ZZICFL F
ZZRNFL F	ZZNS 4	MINT 1.0000E-10
MAXT 1.0000E+10	NSTP 10	IALG 5
STATE VARIABLES DERIVATIVES INITIAL CONDITIONS		
TH 0.07874502	Z09993 0.	THIC 0.07874502
V 491.170015	Z09994 0.	VIC 491.170015
X-1.04555369	Z09998-10.3520168	Z09997 0.
Y 3.90236050	Z09996 38.6372327	Z09995 0.
ALGEBRAIC VARIABLES		
CD 1.00000000	D 2867.11166	DEGRAD 57.3000000
G 32.2000000	MASS 7.00000000	RO 0.00237690
S 10.0000000	THE 0.26178010	THEDEG 15.0000000
TMX 4.00000000	VA 500.000000	VE 40.0000000
VX 489.647983	VY 38.6372327	XMN-60.0000000
Y1 4.00000000	YGE1 0.	YMX 30.0000000
Z09999 0	ZZSEED 5555555555	ZZTLXP T
XICITG 0.		

T 0.20000000	TH 0.07220484	V 454.489096
X-3.86295505	Y 7.44086099	D 2454.86634
T 0.30000000	TH 0.06486207	V 421.726827
X-10.1960489	Y 10.4422282	D 2113.70011
T 0.40000000	TH 0.05696625	V 393.365588
X-19.5501745	Y 12.9249537	D 1838.96577
T 0.50000000	TH 0.04851703	V 368.580106
X-31.5330481	Y 14.9354053	D 1614.52471
T 0.60000000	TH 0.03951440	V 346.740415
X-45.8242341	Y 16.5114590	D 1428.86054
T 0.69000000	TH 0.03093918	V 329.196798
X-60.4398634	Y 17.5845096	D 1287.92959

DEBUG
 OUTPUT

Figure A4-7. Set Debug Printout, Run and Plot - Pilot Ejection Study

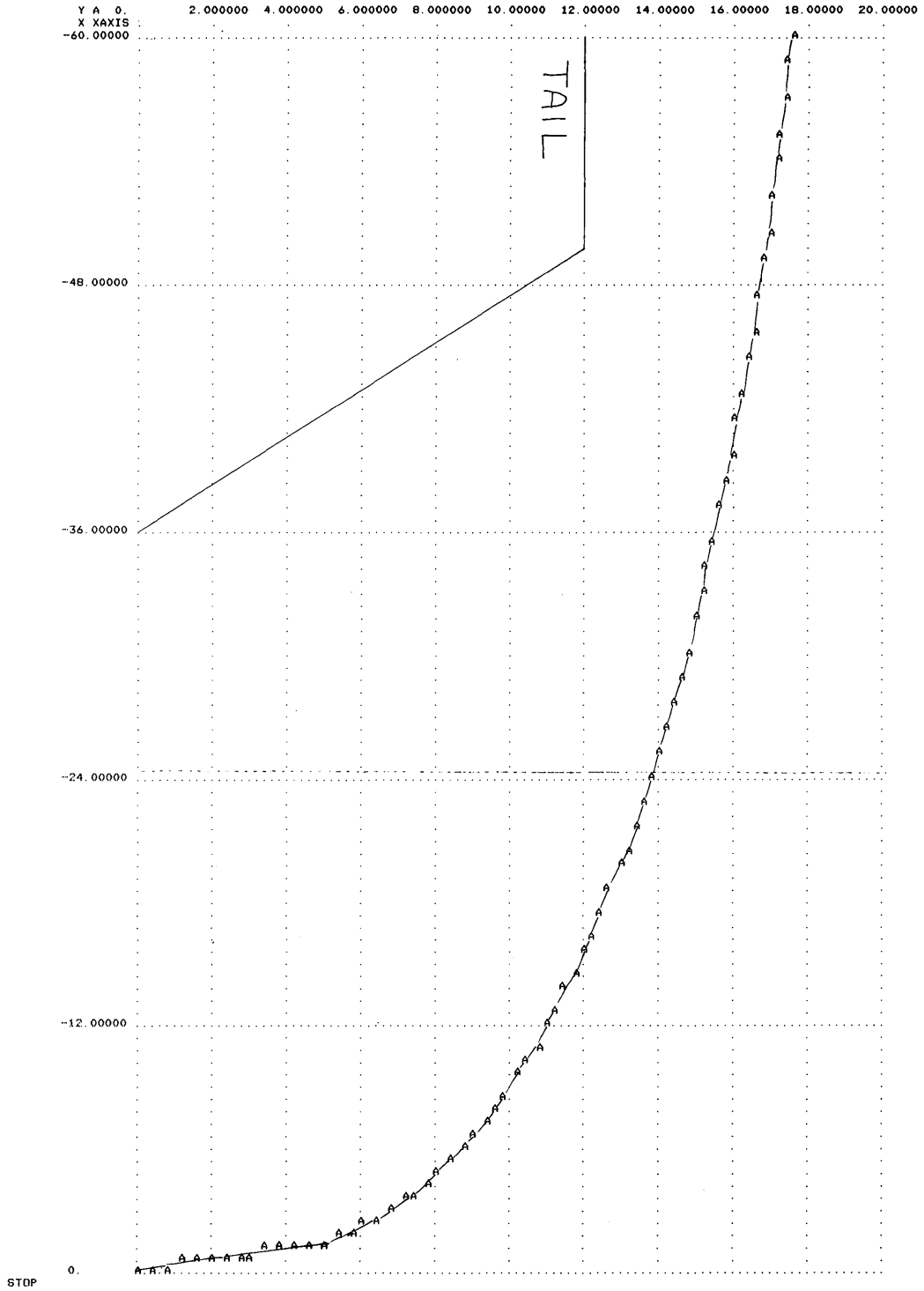


Figure A4-8. Last Trajectory - Pilot Ejection Study

5. TEMPERATURE DISTRIBUTION ALONG A RADIATING FIN

The only practical way of rejecting heat from a power plant operating in outer space is by thermal radiation. If the "working fluid" of the power plant passes through tubes, an efficient radiating surface could be devised by placing many tubes side by side. This would maintain the entire surface at the highest possible temperature. Such a system, though, is highly vulnerable to being punctured by a meteor fragment or some other particle, and this could lead to the loss of the vital working fluid.

A less efficient but also less vulnerable arrangement is shown in Figure A5-1. The number of tubes has been reduced and the space between the tubes has been filled by a fin of rectangular cross section. We want to determine the temperature profile across the fin for various tube spacings, fin thicknesses, fin material, etc. The temperature profile, in turn, can be used to calculate the efficiency of the radiating system. Keep in mind that this is really a problem in static temperature distribution. Time is not a factor because steady state conditions throughout the system are assumed.

5.1 Assumptions

Two views of the radiating fin are shown in Figure A5-2. In terms of this figure, the pertinent assumptions are as follows:

- 1) Steady-state conditions have been established.
- 2) Heat is transferred out of the fin only by radiation through a nonabsorbing medium.
- 3) Thermal properties of the material are constant.
- 4) There is no heat conduction in the y direction.
- 5) Heat loss from the two exposed side edges is small enough to consider the edges as being insulated.
- 6) Temperature is effectively constant across the fin thickness, $2H$, at all values of x which implies $2H \ll W$, $2H \ll L$

These assumptions reduce the problem to mathematical formulation for one-dimensional steady-state heat transfer under combined radiation and conduction.

5.2 Mathematical Formulation

Consider the heat balance for an element strip shown in Figure A5-3 where the width is W , half thickness is H , and length is Δx . The local rate of heat conduction through the cross-sectional area $2HW$ at position x is given by Fourier's equation to be

$$q_x = -K(2HW) \left. \frac{dT}{dx} \right|_x$$

where K is the thermal conductivity of the material.

The rate of heat conduction out of the segment is

$$q_{x+\Delta x} = -K(2HW) \left. \frac{dT}{dx} \right|_{x+\Delta x}$$

The rate of heat lost by radiation is the difference, namely

$$\Delta q = -K(2HW) \left[\left. \frac{dT}{dx} \right|_x - \left. \frac{dT}{dx} \right|_{x+\Delta x} \right]$$

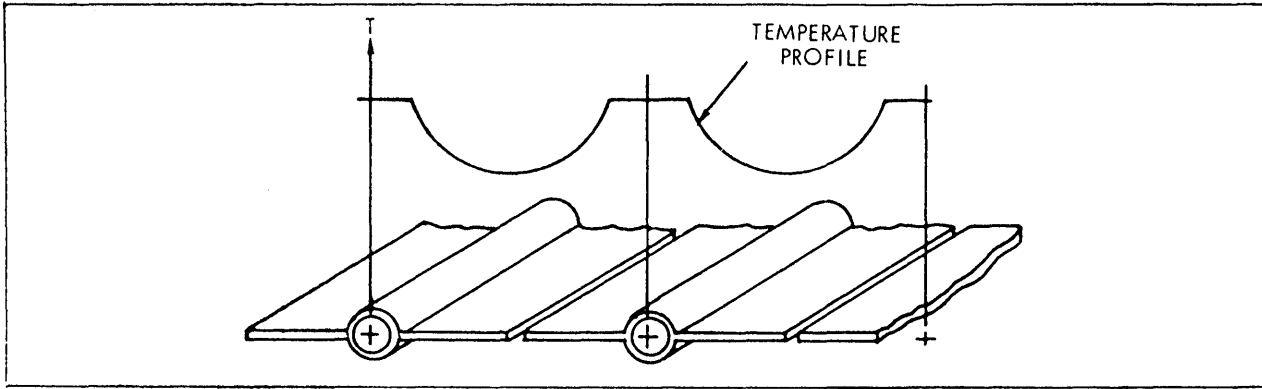


Figure A5-1. A Radiating Surface Using Tubes and Fins

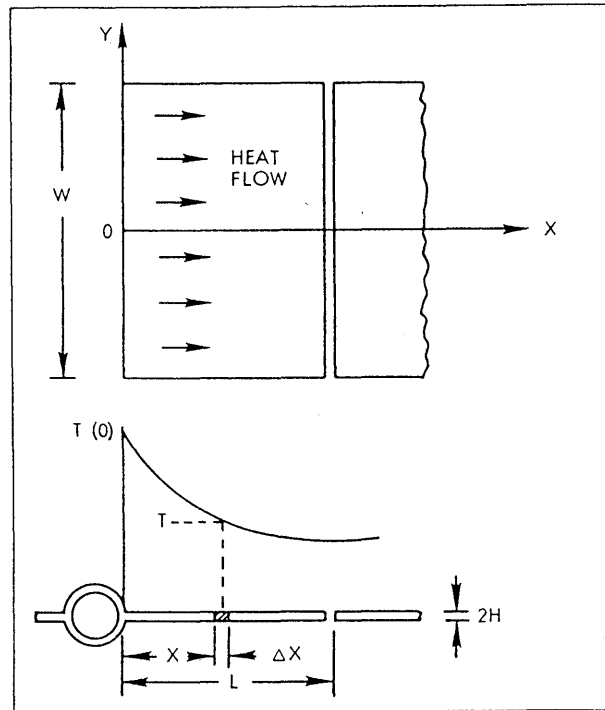


Figure A5-2. Geometry of the Radiating Fin

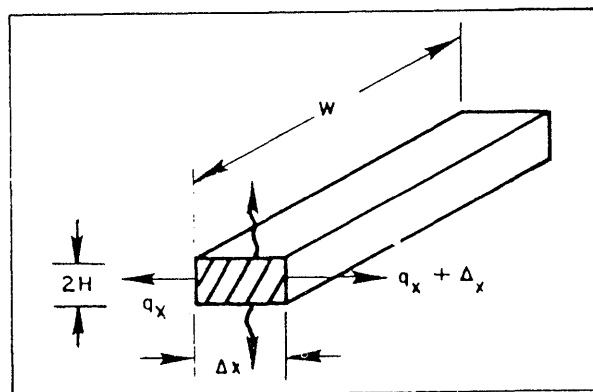


Figure A5-3. Heat Balance for an Elemental Strip

This can also be expressed by means of the Stephan-Boltzman law for heat radiation, considering both the top and bottom surfaces of the element;

$$\Delta q = \sigma \epsilon (T^4 - T_S^4) (2W\Delta x)$$

where

σ = Stephan-Boltzman constant for the material

ϵ = emissivity of the material

T = temperature of the segment, in °R

T_S = temperature of the surroundings, in °R

The heat balance equation, then becomes

$$-K(2HW) \left[\left. \frac{dT}{dx} \right|_x - \left. \frac{dT}{dx} \right|_{x+\Delta x} \right] = \sigma \epsilon (T^4 - T_S^4) (2W\Delta x)$$

Dividing both sides by $-K(2HW)\Delta x$ and inverting the order on the left side gives:

$$\frac{1}{\Delta x} \left[\left. \frac{dT}{dx} \right|_{x+\Delta x} - \left. \frac{dT}{dx} \right|_x \right] = \frac{\sigma \epsilon}{KH} (T^4 - T_S^4)$$

In the limit, as $\Delta x \rightarrow 0$, the left side becomes the second derivative of T with respect to x , so that the final equation becomes

$$\frac{d^2T}{dx^2} = \frac{\sigma \epsilon}{KH} (T^4 - T_S^4)$$

5.3 Initial Conditions and Parameters

Since Equation A5-1 is a second order differential equation, appropriate initial conditions must be specified for the temperature and the temperature gradient at $x = 0$. While the initial temperature is known, (the temperature where the fin joins the tube) the initial temperature gradient is not known. However, it is clear from the symmetry of the arrangement that the temperature gradient at the midpoint between tubes where $x = L$ is zero. Thus, the problem falls in the category of a two-point boundary value problem that must satisfy these conditions:

$$\begin{aligned} T(0) &= 2000 \text{ }^\circ\text{R} \\ \left. \frac{dT}{dx} \right|_{x=L} &= 0 \end{aligned}$$

A complete list of variables and parameters, along with suitable units, is as follows:

T = Temperature	°R
x = Distance	ft
$\sigma = 0.173 \times 10^{-8}$	Btu/(hr) (ft ²) (°R ⁴)
$\epsilon = 0.9$	
$K = 25$	Btu/(hr) (ft ²) (°R)

$$\begin{aligned}
 H &= 0.00125 && \text{ft} \\
 L &= 0.25 && \text{ft} \\
 T_s &= 0 && \text{°R}
 \end{aligned}$$

Notice that the temperature of the surroundings, T_s , has been taken to be at absolute zero for this example.

5.4 Solution

A solution is considered to be successful if

$$\left| \frac{dT}{dx} \right| \leq 0.2 \quad \text{when } x = L$$

The current run is terminated and a new run started if:

- 1) T exceeds its initial value by one percent or more.
- 2) T goes negative.
- 3) $\frac{dT}{dx} > 0.2$ when $x = L$

A new estimate of DTDXZ is computed using:

$$\text{new DTDXZ} = \text{old DTDXZ} - 0.07 * \text{DTDX}(x = L)$$

where $\text{DTDX}(x = L)$ is the final value of the temperature gradient. One case is to be run with an initial temperature of 2000 °R and an estimated temperature gradient of -20,000 °R./ft.

The listing of the model definition section is shown in Figure A5-4. Note that the terminal section checks the final value of temperature slope and if not within the specified tolerance recycles to the initial section for another run with a new value of initial slope. The independent variable is specified to be the variable X and the initial condition is set to 1.0E-10 to give it a lead on the round off error.

The call to subroutine LOG forces an OUTPUT operation and resets the counter for the “NCIOUT” divisor so that the last value of the run will always be listed.

The output stream is shown in Figure A5-5 through A5-12. After the START, the first three iterations find the intermediate temperature is more than 1 percent greater than the starting value, so triggering the terminate (TERMT) condition. For the fourth and subsequent iterations the x variable gets to 0.25 and now the final value of dt/dx is slowly reduced until at iteration number ten the value is -0.153, well within the ERROR tolerance of 0.2. A more detailed printout (PRINT) using data saved on the PREPAR list is started in Figure A5-7 and which goes on to Figure A5-11. Note the use of the flyback trace suppression flag FTSPLT which when true will resynchronize the line count to zero when the independent variable (first variable on the PREPAR list) steps back to the initial condition. Once the iteration has converged, a second START - Figure A5-11 - redoes the last run, since the initial condition on temperature slope (DTDXZ) is still the same. This makes sure that the PREPAR list data base only has one run to be plotted, which shows, in Figure A5-12, the temperature gradient (DTDX) and temperature (T) plotted against distance (X) along the fin. It is often easier and more cost effective merely to repeat the last run of an iteration rather than making complicated arrangements to save each run separately, before it's known that the convergence criteria have been satisfied. Plotting after the first START card would have shown the parametric set of curves as the iteration proceeded, since the PREPARed file doesn't get rewound between runs on cycling from TERMINAL to INITIAL sections. In this case the overstrikes cause rather an unattractive plot and don't provide much information, since it's the final profile that is needed.

PROGRAM RADIATING FIN

```

INTEGER          N
CONSTANT         SG = 1.73E-9      , EP = 0.9
CONSTANT         H = 0.00125      , K = 25.0
CONSTANT         L = 0.25         , TS = 0.0
CONSTANT         TZ = 2000.0      , ERROR = 0.2
CONSTANT         DTDXZ = -20000.0 , GAIN = 0.07
CINTERVAL       CINT = 0.0025
"-----OFF-SET INITIAL VALUE TO HANDLE ACCUMU-...
                LATED ROUND OFF - CHANGE NAME FROM T TO X"
VARIABLE         X = 1.0E-10

INITIAL
"-----NOTE N IS HANDLED AS AN INTEGER"
N                = 0
L1..CONTINUE
N                = N + 1      $* BUMP RUN COUNT *

END $* OF INITIAL *
"-----NOTE NO DYNAMIC SECTION SINCE NOT USED"
DERIVATIVE
"-----INTEGRATE FOR TEMPERATURE AND TEMP RATE"
"                NOTE T IS NOW TEMP NOT TIME"
T                = INTEG(DTDX, TZ)
DTDX             = INTEG(SG*EP*(T**4 - TS**4)/(K*H), DTDXZ)
"-----SPECIFY TERMINATION CONDITION"
TERMT(X.GE.L .OR. T.GE.1.01*TZ .OR. T.LT.0.0)

END $* OF DERIVATIVE *

TERMINAL
"-----FORCE OUTPUT AT END OF EVERY SWEEP"
CALL LOG
"-----IF CONVERGED OR TOO MANY TRIES"
IF((ABS(DTDX).LT.ERROR) .OR. N.GT.10) GO TO L2
"-----FIND NEW GUESS FOR INITIAL TEMP RATE"
DTDXZ           = DTDXZ - GAIN*DTDX
"-----TRY RUN AGAIN"
GO TO L1
L2..CONTINUE

END $* OF TERMINAL *

END $* OF PROGRAM *
```

Figure A5-4. Listing of Model Definition Section for Radiating Fin Problem

```

SET TITLE = "RADIATING FIN PROBLEM"
S TCWPRN=72,DIS=9 $" FORCE 3 COLUMN OUTPUT WIDTH "
OUTPUT X, T, N, DTDX, DTDXZ, "NCIOUT"=20
PREPAR X, DTDX, T, N, DTDXZ
START      $"PERFORM ITERATION"
      X 1.0000E-10          T 2000.00000          N      1
      DTDX-20000.0000      DTDXZ-20000.0000
      X 0.05000000        T 1655.08858          N      1
      DTDX 3125.64616      DTDXZ-20000.0000
      X 0.08400000        T 2022.12802          N      1
      DTDX 20882.2789      DTDXZ-20000.0000
      X 0.04750000        T 1549.94861          N      2
      DTDX-1064.29250      DTDXZ-21461.7595
      X 0.09750000        T 1903.29153          N      2
      DTDX 17906.0279      DTDXZ-21461.7595
      X 0.10350000        T 2023.47708          N      2
      DTDX 22336.6824      DTDXZ-21461.7595
      X 0.04750000        T 1446.93596          N      3
      DTDX-4337.96377      DTDXZ-23025.3273
      X 0.09750000        T 1481.25259          N      3
      DTDX 5876.63976      DTDXZ-23025.3273
      X 0.13925000        T 2023.07249          N      3
      DTDX 23828.7709      DTDXZ-23025.3273
      X 0.04750000        T 1339.47700          N      4
      DTDX-7612.48011      DTDXZ-24693.3413
      X 0.09750000        T 1108.29245          N      4
      DTDX-2310.61152      DTDXZ-24693.3413
      X 0.14750000        T 1078.97754          N      4
      DTDX 1076.41560      DTDXZ-24693.3413
      X 0.19750000        T 1228.43104          N      4
      DTDX 5269.23907      DTDXZ-24693.3413
      X 0.24750000        T 1702.91644          N      4
      DTDX 16044.3551      DTDXZ-24693.3413
      X 0.25000000        T 1744.37926          N      4
      DTDX 17143.5322      DTDXZ-24693.3413
      X 0.04750000        T 1263.59645          N      5
      DTDX-9844.85583      DTDXZ-25893.3885
      X 0.09750000        T 873.352393          N      5
      DTDX-6545.72840      DTDXZ-25893.3885
  
```

Figure A5-5. Run-time Commands and Output Stream from Radiating Fin Problem

X 0.14750000 DTDX-5823.09994	T 568.948130 DTDXZ-25893.3885	N	5
X 0.19750000 DTDX-5723.23787	T 281.268091 DTDXZ-25893.3885	N	5
X 0.24675000 DTDX-5720.17206	T-0.47550353 DTDXZ-25893.3885	N	5
X 0.04750000 DTDX-9110.73249	T 1288.78802 DTDXZ-25492.9765	N	6
X 0.09750000 DTDX-5244.32007	T 949.223751 DTDXZ-25492.9765	N	6
X 0.14750000 DTDX-4011.22327	T 723.295028 DTDXZ-25492.9765	N	6
X 0.19750000 DTDX-3607.81982	T 534.780029 DTDXZ-25492.9765	N	6
X 0.24750000 DTDX-3501.62433	T 357.711010 DTDXZ-25492.9765	N	6
X 0.25000000 DTDX-3499.68229	T 348.959417 DTDXZ-25492.9765	N	6
X 0.04750000 DTDX-8656.36174	T 1304.26264 DTDXZ-25247.9987	N	7
X 0.09750000 DTDX-4396.61310	T 996.832602 DTDXZ-25247.9987	N	7
X 0.14750000 DTDX-2704.41990	T 824.629724 DTDXZ-25247.9987	N	7
X 0.19750000 DTDX-1840.86108	T 713.083960 DTDXZ-25247.9987	N	7
X 0.24750000 DTDX-1329.91836	T 634.803277 DTDXZ-25247.9987	N	7
X 0.25000000 DTDX-1309.90090	T 631.503590 DTDXZ-25247.9987	N	7
X 0.04750000 DTDX-8485.25438	T 1310.06695 DTDXZ-25156.3056	N	8
X 0.09750000 DTDX-4068.36337	T 1014.89813 DTDXZ-25156.3056	N	8
X 0.14750000 DTDX-2166.72613	T 864.120545 DTDXZ-25156.3056	N	8

Figure A5-6. Output Stream from Radiating Fin Problem

X 0.19750000	T 785.914904	N	8
DTDX-1033.46037	DTDXZ-25156.3056		
X 0.24750000	T 756.432739	N	8
DTDX-168.016381	DTDXZ-25156.3056		
X 0.25000000	T 756.063639	N	8
DTDX-127.276801	DTDXZ-25156.3056		
X 0.04750000	T 1310.63128	N	9
DTDX-8468.59821	DTDXZ-25147.3963		
X 0.09750000	T 1016.66080	N	9
DTDX-4036.13740	DTDXZ-25147.3963		
X 0.14750000	T 868.006407	N	9
DTDX-2112.90756	DTDXZ-25147.3963		
X 0.19750000	T 793.197114	N	9
DTDX-949.677376	DTDXZ-25147.3963		
X 0.24750000	T 768.933252	N	9
DTDX-39.8412710	DTDXZ-25147.3963		
X 0.25000000	T 768.888072	N	9
DTDX 3.69604386	DTDXZ-25147.3963		
X 0.04750000	T 1310.61489	N	10
DTDX-8469.08197	DTDXZ-25147.6550		
X 0.09750000	T 1016.60960	N	10
DTDX-4037.07406	DTDXZ-25147.6550		
X 0.14750000	T 867.893440	N	10
DTDX-2114.47448	DTDXZ-25147.6550		
X 0.19750000	T 792.985106	N	10
DTDX-952.124666	DTDXZ-25147.6550		
X 0.24750000	T 768.568386	N	10
DTDX-43.6068998	DTDXZ-25147.6550		
X 0.25000000	T 768.513688	N	10
DTDX-0.15323730	DTDXZ-25147.6550		
X 0.25000000	T 768.513688	N	10
DTDX-0.15323730	DTDXZ-25147.6550		

S FTSPLT=.T. \$* SYNCHRONIZE OUTPUT WITH START OF EACH SWEEP *
 PRINT "NCIPRN"=5, X, DTDX, T, N, DTDXZ \$* DEMONSTRATE COLUMN PRINT *

LINE	X	DTDX	T	N	DTDXZ
0	1.000E-10	-20000.000	2000.0000	1	-20000.000
5	0.0125000	-11950.188	1803.7801	1	-20000.000
10	0.0250000	-6205.1661	1691.8472	1	-20000.000

Figure A5-7. Output Stream from Radiating Fin Problem

LINE	X	DTDX	T	N	DTDXZ
15	0.0375000	-1436.4184	1644.6516	1	-20000.000
20	0.0500000	3125.6462	1655.0886	1	-20000.000
25	0.0625000	8146.5127	1724.6749	1	-20000.000
30	0.0750000	14529.539	1864.3242	1	-20000.000
0	1.000E-10	-21461.760	2000.0000	2	-21461.760
5	0.0125000	-13559.972	1784.8628	2	-21461.760
10	0.0250000	-8190.9365	1650.6714	2	-21461.760
15	0.0375000	-4016.3286	1575.1924	2	-21461.760
20	0.0500000	-347.33759	1548.1848	2	-21461.760
25	0.0625000	3280.2301	1566.3385	2	-21461.760
30	0.0750000	7313.5237	1631.8588	2	-21461.760
35	0.0875000	12375.511	1753.3970	2	-21461.760
40	0.1000000	19621.831	1950.1666	2	-21461.760
0	1.000E-10	-23025.327	2000.0000	3	-23025.327
5	0.0125000	-15278.621	1764.6382	3	-23025.327
10	0.0250000	-10289.154	1606.7887	3	-23025.327
15	0.0375000	-6681.2732	1501.7414	3	-23025.327
20	0.0500000	-3799.7728	1436.7669	3	-23025.327
25	0.0625000	-1278.3675	1405.2621	3	-23025.327
30	0.0750000	1129.9274	1404.3409	3	-23025.327
35	0.0875000	3637.9960	1433.9221	3	-23025.327
40	0.1000000	6488.7176	1496.7040	3	-23025.327
45	0.1125000	10037.549	1599.0140	3	-23025.327
50	0.1250000	14913.851	1753.0901	3	-23025.327
55	0.1375000	22426.033	1982.6161	3	-23025.327
0	1.000E-10	-24693.341	2000.0000	4	-24693.341
5	0.0125000	-17108.433	1743.0743	4	-24693.341
10	0.0250000	-12499.304	1560.1524	4	-24693.341
15	0.0375000	-9424.8571	1424.2928	4	-24693.341
20	0.0500000	-7222.5452	1320.9377	4	-24693.341
25	0.0625000	-5550.1240	1241.5406	4	-24693.341
30	0.0750000	-4214.3992	1180.7917	4	-24693.341
35	0.0875000	-3097.9892	1135.2723	4	-24693.341
40	0.1000000	-2124.5781	1102.7492	4	-24693.341
45	0.1125000	-1240.9257	1081.7856	4	-24693.341
50	0.1250000	-406.72519	1071.5209	4	-24693.341
55	0.1375000	411.73022	1071.5521	4	-24693.341
60	0.1500000	1246.1259	1081.8803	4	-24693.341
65	0.1625000	2130.1939	1102.9113	4	-24693.341
70	0.1750000	3104.2982	1135.5086	4	-24693.341
75	0.1875000	4221.7835	1181.1131	4	-24693.341
80	0.2000000	5559.1505	1241.9639	4	-24693.341
85	0.2125000	7234.1135	1321.4885	4	-24693.341
90	0.2250000	9440.4962	1425.0116	4	-24693.341
95	0.2375000	12521.825	1561.1057	4	-24693.341
100	0.2500000	17143.532	1744.3793	4	-24693.341
0	1.000E-10	-25893.389	2000.0000	5	-25893.389
5	0.0125000	-18422.598	1727.5675	5	-25893.389
10	0.0250000	-14072.089	1526.7101	5	-25893.389
15	0.0375000	-11340.023	1369.1217	5	-25893.389
20	0.0500000	-9539.3127	1239.3712	5	-25893.389
25	0.0625000	-8316.0321	1128.2529	5	-25893.389
30	0.0750000	-7470.8513	1029.9053	5	-25893.389

Figure A5-8. Output Stream from Radiating Fin Problem

LINE	X	DTDX	T	N	DTDXZ
35	0.0875000	-6883.2429	940.41418	5	-25893.389
40	0.1000000	-6475.9178	857.07643	5	-25893.389
45	0.1125000	-6196.8071	777.98410	5	-25893.389
50	0.1250000	-6009.3789	701.77561	5	-25893.389
55	0.1375000	-5887.1889	627.47874	5	-25893.389
60	0.1500000	-5810.6988	554.40615	5	-25893.389
65	0.1625000	-5765.3527	482.08202	5	-25893.389
70	0.1750000	-5740.3654	410.18791	5	-25893.389
75	0.1875000	-5727.9112	338.52099	5	-25893.389
80	0.2000000	-5722.5337	266.96091	5	-25893.389
85	0.2125000	-5720.6688	195.44322	5	-25893.389
90	0.2250000	-5720.2230	123.93842	5	-25893.389
95	0.2375000	-5720.1727	52.436089	5	-25893.389
0	1.000E-10	-25492.976	2000.0000	6	-25492.976
5	0.0125000	-17984.320	1732.7409	6	-25492.976
10	0.0250000	-13548.881	1537.8585	6	-25492.976
15	0.0375000	-10706.228	1387.4808	6	-25492.976
20	0.0500000	-8778.8812	1266.4308	6	-25492.976
25	0.0625000	-7418.6188	1165.6658	6	-25492.976
30	0.0750000	-6430.3941	1079.4255	6	-25492.976
35	0.0875000	-5697.4258	1003.8480	6	-25492.976
40	0.1000000	-5145.9343	936.23706	6	-25492.976
45	0.1125000	-4727.1782	874.64868	6	-25492.976
50	0.1250000	-4407.7051	817.64524	6	-25492.976
55	0.1375000	-4163.7799	764.14214	6	-25492.976
60	0.1500000	-3978.0619	713.30881	6	-25492.976
65	0.1625000	-3837.5517	664.50264	6	-25492.976
70	0.1750000	-3732.2844	617.22347	6	-25492.976
75	0.1875000	-3654.4787	571.08133	6	-25492.976
80	0.2000000	-3597.9696	525.77293	6	-25492.976
85	0.2125000	-3557.8247	481.06403	6	-25492.976
90	0.2250000	-3530.0784	436.77576	6	-25492.976
95	0.2375000	-3511.5450	392.77378	6	-25492.976
100	0.2500000	-3499.6823	348.95942	6	-25492.976
0	1.000E-10	-25247.999	2000.0000	7	-25247.999
5	0.0125000	-17716.071	1735.9064	7	-25247.999
10	0.0250000	-13228.002	1544.6842	7	-25247.999
15	0.0375000	-10315.908	1398.7374	7	-25247.999
20	0.0500000	-8307.5240	1283.0626	7	-25247.999
25	0.0625000	-6857.3116	1188.7433	7	-25247.999
30	0.0750000	-5771.9082	1110.1200	7	-25247.999
35	0.0875000	-4935.7961	1043.4126	7	-25247.999
40	0.1000000	-4276.2819	985.99258	7	-25247.999
45	0.1125000	-3745.6431	935.97062	7	-25247.999
50	0.1250000	-3311.4293	891.95105	7	-25247.999
55	0.1375000	-2950.9094	852.87870	7	-25247.999
60	0.1500000	-2647.7512	817.93989	7	-25247.999
65	0.1625000	-2389.9584	786.49631	7	-25247.999
70	0.1750000	-2168.5485	758.03963	7	-25247.999
75	0.1875000	-1976.6806	732.15972	7	-25247.999
80	0.2000000	-1809.0654	708.52172	7	-25247.999
85	0.2125000	-1661.5577	686.84942	7	-25247.999
90	0.2250000	-1530.8693	666.91279	7	-25247.999

Figure A5-9. Output Stream from Radiating Fin Problem

LINE	X	DTDX	T	N	DTDXZ
95	0.2375000	-1414.3626	648.51867	7	-25247.999
100	0.2500000	-1309.9009	631.50359	7	-25247.999
0	1.000E-10	-25156.306	2000.0000	8	-25156.306
5	0.0125000	-17615.648	1737.0913	8	-25156.306
10	0.0250000	-13107.748	1547.2400	8	-25156.306
15	0.0375000	-10169.309	1402.9554	8	-25156.306
20	0.0500000	-8129.8790	1289.3028	8	-25156.306
25	0.0625000	-6644.7443	1197.4186	8	-25156.306
30	0.0750000	-5520.9489	1121.6888	8	-25156.306
35	0.0875000	-4643.1741	1058.3755	8	-25156.306
40	0.1000000	-3938.8075	1004.8902	8	-25156.306
45	0.1125000	-3360.1285	959.38361	8	-25156.306
50	0.1250000	-2874.6275	920.50009	8	-25156.306
55	0.1375000	-2459.4608	887.22573	8	-25156.306
60	0.1500000	-2098.1317	858.78983	8	-25156.306
65	0.1625000	-1778.4262	834.59941	8	-25156.306
70	0.1750000	-1491.0857	814.19460	8	-25156.306
75	0.1875000	-1228.9278	797.21754	8	-25156.306
80	0.2000000	-986.24658	783.39040	8	-25156.306
85	0.2125000	-758.39242	772.49970	8	-25156.306
90	0.2250000	-541.46852	764.38512	8	-25156.306
95	0.2375000	-332.10395	758.93154	8	-25156.306
100	0.2500000	-127.27680	756.06364	8	-25156.306
0	1.000E-10	-25147.396	2000.0000	9	-25147.396
5	0.0125000	-17605.890	1737.2064	9	-25147.396
10	0.0250000	-13096.059	1547.4884	9	-25147.396
15	0.0375000	-10155.050	1403.3654	9	-25147.396
20	0.0500000	-8112.5823	1289.9095	9	-25147.396
25	0.0625000	-6624.0172	1198.2626	9	-25147.396
30	0.0750000	-5496.4308	1122.8152	9	-25147.396
35	0.0875000	-4614.5151	1059.8339	9	-25147.396
40	0.1000000	-3905.6538	1006.7346	9	-25147.396
45	0.1125000	-3322.1126	961.67243	9	-25147.396
50	0.1250000	-2831.3585	923.29651	9	-25147.396
55	0.1375000	-2410.5157	890.59803	9	-25147.396
60	0.1500000	-2043.0452	862.81182	9	-25147.396
65	0.1625000	-1716.6796	839.35103	9	-25147.396
70	0.1750000	-1422.0937	819.76269	9	-25147.396
75	0.1875000	-1152.0229	803.69674	9	-25147.396
80	0.2000000	-900.66022	790.88431	9	-25147.396
85	0.2125000	-663.23160	781.12228	9	-25147.396
90	0.2250000	-435.68652	774.26241	9	-25147.396
95	0.2375000	-214.46367	770.20382	9	-25147.396
100	0.2500000	3.6960439	768.88807	9	-25147.396
0	1.000E-10	-25147.655	2000.0000	10	-25147.655
5	0.0125000	-17606.173	1737.2030	10	-25147.655
10	0.0250000	-13096.398	1547.4812	10	-25147.655
15	0.0375000	-10155.464	1403.3535	10	-25147.655
20	0.0500000	-8113.0847	1289.8919	10	-25147.655
25	0.0625000	-6624.6193	1198.2381	10	-25147.655
30	0.0750000	-5497.1432	1122.7825	10	-25147.655
35	0.0875000	-4615.3479	1059.7915	10	-25147.655
40	0.1000000	-3906.6175	1006.6811	10	-25147.655

Figure A5-10. Output Stream from Radiating Fin Problem

LINE	X	DTDX	T	N	DTDXZ
45	0.1125000	-3323.2179	961.60593	10	-25147.655
50	0.1250000	-2832.6171	923.21525	10	-25147.655
55	0.1375000	-2411.9401	890.50001	10	-25147.655
60	0.1500000	-2044.6493	862.69488	10	-25147.655
65	0.1625000	-1718.4787	839.21285	10	-25147.655
70	0.1750000	-1424.1055	819.60071	10	-25147.655
75	0.1875000	-1154.2674	803.50818	10	-25147.655
80	0.2000000	-903.16074	790.66612	10	-25147.655
85	0.2125000	-666.01515	780.87109	10	-25147.655
90	0.2250000	-438.78500	773.97449	10	-25147.655
95	0.2375000	-217.91490	769.87501	10	-25147.655
100	0.2500000	-0.1532373	768.51369	10	-25147.655
START	*REPEAT LAST CASE OF PREVIOUS RUN*				
	X 1.0000E-10		T 2000.00000	N	1
	DTDX-25147.6550		DTDXZ-25147.6550		
	X 0.05000000		T 1289.89191	N	1
	DTDX-8113.08470		DTDXZ-25147.6550		
	X 0.10000000		T 1006.68105	N	1
	DTDX-3906.61749		DTDXZ-25147.6550		
	X 0.15000000		T 862.694885	N	1
	DTDX-2044.64926		DTDXZ-25147.6550		
	X 0.20000000		T 790.666118	N	1
	DTDX-903.160736		DTDXZ-25147.6550		
	X 0.25000000		T 768.513688	N	1
	DTDX-0.15323730		DTDXZ-25147.6550		
	X 0.25000000		T 768.513688	N	1
	DTDX-0.15323730		DTDXZ-25147.6550		

PLOT *XHI*=L, DTDX, T *PLOT CONVERGED PROFILE. USE DEFAULT XAXIS*

Figure A5-11. Output Stream from Radiating Fin Problem. Repeat of Last Converged Run

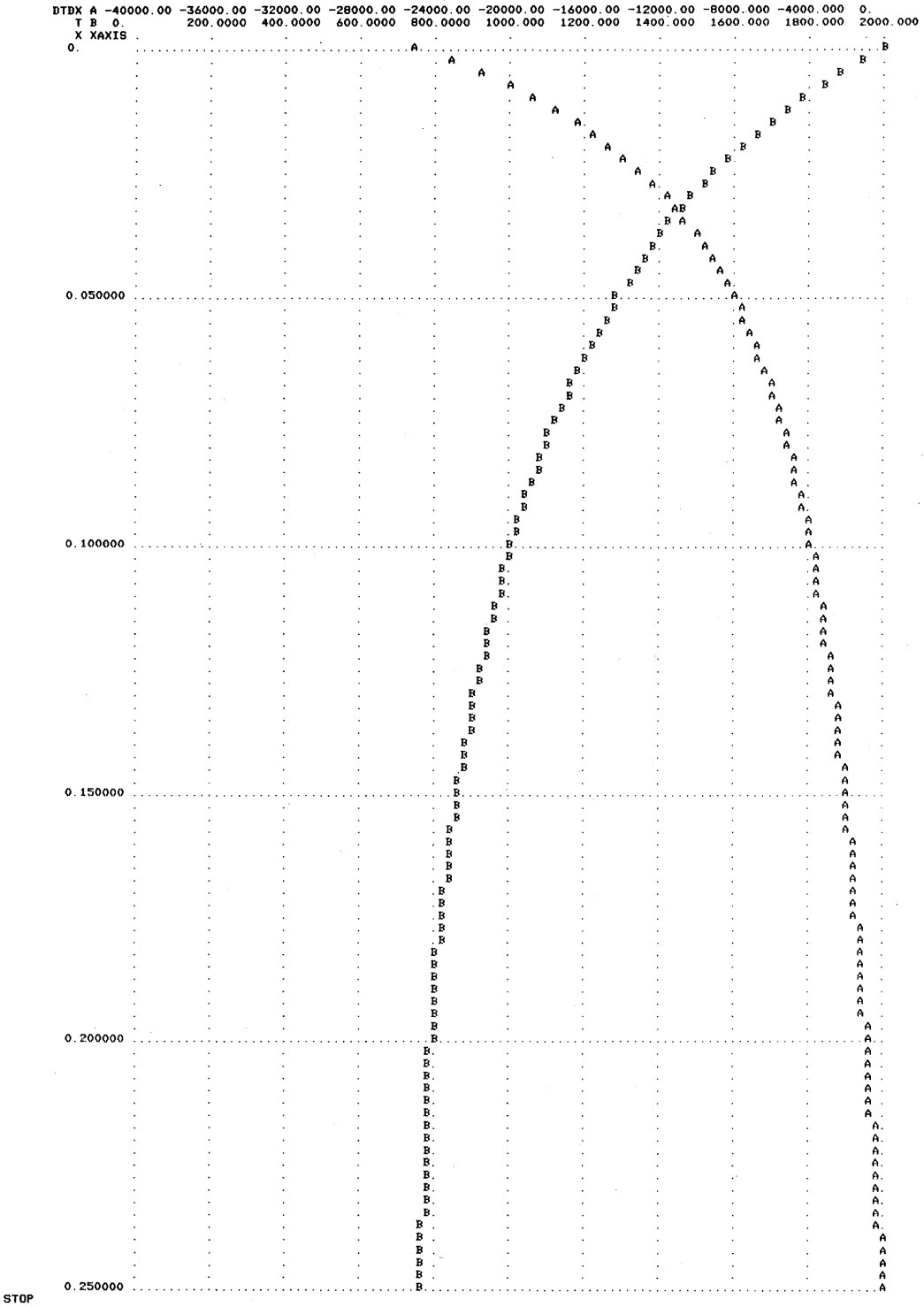


Figure A5-12. Plot of Temperature Profile and Gradient from Last Converged Run - Radiating Fin Problem

6. AIRCRAFT ARRESTING GEAR SYSTEM

The system investigated here is designed to halt a moving aircraft that would otherwise overrun the end of a runway. It is similar in principle to the gear used on aircraft carriers. The problem was selected for this manual because it employs a function generator and input relays. The equations listed below describe the system, and its geometry is shown in Figure A6-1. A plot of the "water squeezer" damping function, $f(y_3)$, is also shown on the figure. The aims of the investigation are to determine the range of aircraft weights and speeds that can be accommodated without exceeding the working limits of the cables or the allowable piston travel. Here, just two cases are investigated involving different aircraft velocities.

Differential equations describing the acceleration of the three masses are:

$$m_3 \frac{d^2 y_3}{dt^2} = f_{K2} - f_D; m_2 \frac{d^2 y_2}{dt^2} = wf_{K1} - f_{K2}$$

$$m_1 \frac{d^2 x}{dt^2} = -2f_{K1} \sin \theta$$

where the cable tension is given by:

$$f_{K1} = \begin{cases} k_1 (y_1 - 2y_2) & y_1 > 2y_2 \\ 0 & ; y_1 \leq 2y_2 \end{cases}$$

$$f_{K2} = \begin{cases} k_2 (y_2 - y_3) & y_2 > y_3 \\ 0 & y_2 \leq y_3 \end{cases}$$

Drag force from the water squeezer is given by

$$f_D = F(y_3) \left(\frac{dy_3}{dt} \right)^2$$

and geometrical constraints lead to

$$y_1 = \sqrt{(x^2 + h^2)} - h$$

$$\sin \theta = \frac{x}{h + y_1} = \frac{x}{\sqrt{(x^2 + h^2)}}$$

Constants for the problem are as follows:

$$\begin{array}{ll} m_1 = 1400 \text{ slugs} & k_1 = 4550 \text{ lb/ft} \\ m_2 = 45.28 \text{ slugs} & k_2 = 25,300 \text{ lb/ft} \\ m_3 = 20 \text{ slugs} & h = 125 \text{ ft} \end{array}$$

Initial conditions are given by:

$$\begin{array}{ll} \dot{y}_3(0) = 0 & y_3(0) = 0 \\ \dot{y}_2(0) = 0 & y_2(0) = 0 \\ \dot{x}(0) = 290 \text{ ft/sec (Run 1)} & x(0) = 0 \\ & = 200 \text{ ft/sec (Run 2)} \end{array}$$

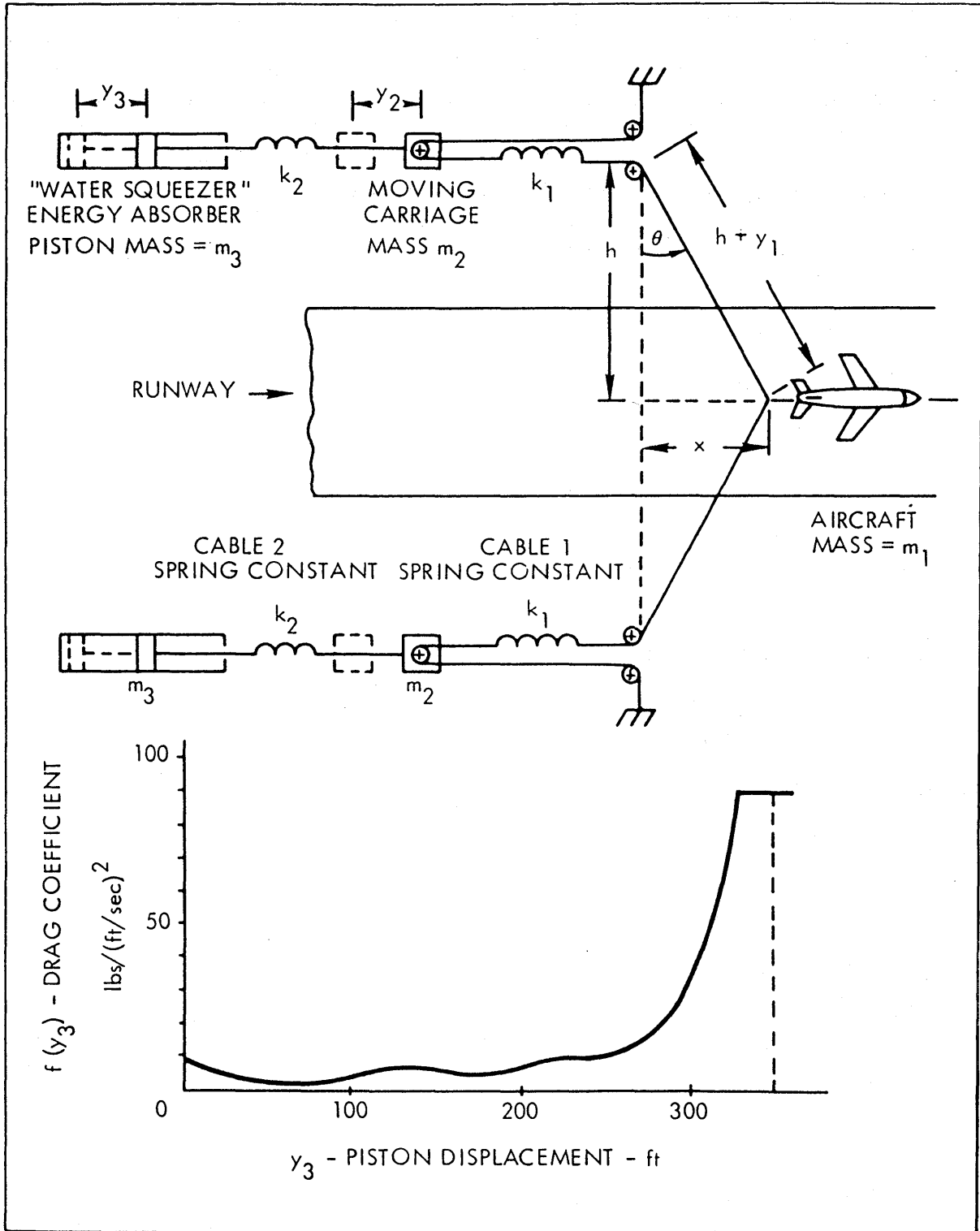


Figure A6-1. Aircraft Arresting Gear System

An arbitrary function of one variable is used for the squeezer drag coefficient.

$F(y_3) =$ A function of y_3 - The data points for the function are shown on the listing, Figure A6-1. Linear interpolation between points is desired.

This problem was programmed in the explicit mode with the INITIAL, DYNAMIC and DERIVATIVE regions being defined. The only reason is to show the use of standard FORTRAN output to write out the initial speed and case number at the beginning of each run. The DERIVATIVE section contains the descriptive equations modelling the system. The model definition listing is shown in Figure A6-2.

The cables can only transmit tensile forces and account must be taken of the fact that when the cable extension is negative, the force is zero. Expressing the force as proportional to the simple extension $k_2 (y_2 - y_3)$ is more representative of a spring. In the original formulation of this problem the function real switch operator RSW was used, i.e.

$FK2 = RSW(Y2. GE. Y3, K2*(Y2 - Y3), 0.0)$

When the first argument is true, the second argument is returned, else the third. A more elegant way of accomplishing the same thing is to use the FORTRAN function DIM (q.v.) that returns the positive difference between the two arguments - if the difference is negative (arg 1 minus arg 2) the result is zero. The force equation then becomes

$FK2 = K2*DIM(Y2, Y3)$

The damping coefficient, $f(y_3)$, was represented by a straight line approximation passing through 16 breakpoints (15 segments). A better fit to the given function could have been obtained by using more breakpoints.

Note the choice of termination conditions. The TERMT statement stops the problem after 10 sec, or when x reaches 1000 ft.

The run-time command sequence follows (Figures A6-3 through A6-7) that sets the TITLE and changes the grid spacing from the default value of ten to fifty. The effect can be seen on the clarity of the following plots. The integration step size had to be changed by the following statement (NSTP is the number of steps in a communication interval) so

$SET NSTP = 100$

makes the step size 1 msec since the communication interval is set to 0.2 sec. The reason for this is the small time constant associated with the third mass, m_3 , together with the nonlinearity - the function - that makes the program particularly susceptible to step size. The system becomes unstable when the step size is increased to 4 msec. A better way to control these types of models when using a fixed step algorithm is to set the maximum step size (MAXT) to control (i.e., SET MAX=0.001). In this case the step size will stay at this value (if NSTP=1) when the communication interval is changed, unless CINT is made less than MAXT, in which event the smaller value becomes the integration step size.

The next run was chosen to illustrate the two types of line plots available by setting both CALPLT and STRPLT to true. When system variable CALPLT true, the plot is shown in Figure A6-8. Here the axes are drawn side by side and all three plots occupy the same area. The strip chart plot (STRPLT true) is shown in Figure A6-9. Each curve occupies a separate slot, normally two inches by five inches, stacked in the order given from the bottom. Notice the addition of the XTAG string on both line plots, which adds a unit specification to the time variable, T, along the x-axis. Since PRNPLT is still true, the printer plot of Figure A6-7 is obtained at the same time. This plot could have been suppressed by making PRNPLT false.


```

PROGRAM AIRCRAFT ARRESTING GEAR
INTEGER      NCASE
*-----DEFINE TABLE FOR WATER SQUEEZER DRAG*
TABLE        FY3, 1, 16
             / -10.0, 0.0, 30.0, 60.0, 120.0
             , 150.0, 180.0, 210.0, 240.0, 270.0
             , 282.0, 294.0, 306.0, 312.0, 324.0
             , 350.0
             , 8.33, 8.33, 4.0, 1.6, 5.2
             , 5.2, 6.6, 8.3, 10.7, 16.0
             , 21.0, 28.0, 41.0, 50.0, 90.0
             , 90.0 /
*-----DEFINE PRESET VARIABLES*
CONSTANT     M1 = 140.0, M2 = 45.28, M3 = 20.0
             , K1 = 4550.0, K2 = 25300.0, M = 125.0
             , NCASE = 0, TSTP = 9.999, XMX = 1000.0
             , H = 125.0, SPEED = 250.0
CINTERVAL   DL = 0.10
INITIAL
NCASE = NCASE + 1 $"BUMP CASE NUMBER"
*-----TELL SYSTEM ABOUT TO PRINT FIVE LINES *
LINES(5)
WRITE(6, 200) SPEED, NCASE
200. FORMAT(//20X,17HAIRCRAFT SPEED - ,F6.2,3X,8HRUN NO. ,I3//)
END $"OF INITIAL"
DYNAMIC
DERIVATIVE
*-----COMPUTE SECOND DERIVATIVES*
Y3DD = (FK2 - FD)/M3
Y2DD = (2.0*FK1 - FK2)/M2
XDD  = -2.0*FK1*STH/M1
*-----INTEGRATE FOR FIRST DERIVATIVES*
Y3D  = INTEG(Y3DD, 0.0)
Y2D  = INTEG(Y2DD, 0.0)
XD   = INTEG(XDD, SPEED)
*-----INTEGRATE FOR POSITIONS*
Y3   = INTEG(Y3D, 0.0)
Y2   = INTEG(Y2D, 0.0)
X    = INTEG(XD, 0.0)
*-----CABLE TENSION BECOMES ZERO WHEN SLACK*
*
* CAN NEVER GO NEGATIVE - SO USE POSITIVE *
* DIFFERENCE FUNCTION *
FK1  = K1*DIM(Y1, 2.0*Y2)
FK2  = K2*DIM(Y2, Y3)
*-----WATER SQUEEZER DRAG*
FD   = FY3(Y3)*Y3D**2
*-----GEOMETRICAL RELATIONS*
Y1   = SQRT(X**2 + H**2) - H
STH  = X/(H + Y1)
END $"OF DERIVATIVE"
*-----SPECIFY TERMINATION CONDITION*
TERMT(T.GE.TSTP .OR. X.GE.XMX)
END $"OF DYNAMIC"
END $"OF PROGRAM"

```

Figure A6-2. Listing of Model Definition for Aircraft Arresting Gear Problem

```

SET TITLE = "AIRCRAFT ARRESTING GEAR PROBLEM"
S TCWPRN=72,DIS=9 $" FORCE 3 COLUMN OUTPUT WIDTH "
OUTPUT T, Y3DD, Y3D, Y3, FD, Y2DD, Y2D, Y2, FK2, XDD, XD, X
, FK1, Y1, "NCIOUT" = 20
PREPAR T, X, XD, XDD
SET NGXPPL = 50, NGYPPL = 50, NSTP = 100
SET SPEED = 200.0 $ START $ PLOT X, XD, XDD
SET SPEED = 290.0 $ START
SET CALPLT = .TRUE. $"TURN ON CALCOMP PLOTS"
S STRPLT=.T.,CALPLT=.T.
PLOT "XTAG"="(SEC)", X, XD, XDD
STOP

```

Figure A6-3. Run-time Drive Commands for Aircraft Arresting Gear Problem

SET TITLE = "AIRCRAFT ARRESTING GEAR PROBLEM"
 S TCWPRN=72,DIS=9 \$" FORCE 3 COLUMN OUTPUT WIDTH "
 OUTPUT T, Y3DD, Y3D, Y3, FD, Y2DD, Y2D, Y2, FK2, XDD, XD, X ...
 , FK1, Y1, "NCIOUT" = 20
 PREPAR T, X, XD, XDD
 SET NGXPPL = 50, NGYPPL = 50, NSTP = 100
 SET SPEED = 200.0 \$ START \$ PLOT X, XD, XDD

AIRCRAFT SPEED - 200.00 RUN NO. 1

T 0.	Y3DD 0.	Y3D 0.
Y3 0.	FD 0.	Y2DD 0.
Y2D 0.	Y2 0.	FK2 0.
XDD 0.	XD 200.000000	X 0.
FK1 0.	Y1 0.	
T 2.00000000	Y3DD-18.0123073	Y3D 45.9392902
Y3 96.2945828	FD 7972.47469	Y2DD-18.8087675
Y2D 45.8334561	Y2 96.5954614	FK2 7612.22854
XDD-44.4263306	XD 99.1682231	X 293.417443
FK1 3380.28378	Y1 193.933842	
T 4.00000000	Y3DD-6.57593866	Y3D 20.9752641
Y3 158.436671	FD 2461.01876	Y2DD-6.69116942
Y2D 20.9323673	Y2 158.528746	FK2 2329.49998
XDD-13.8850190	XD 43.5150304	X 424.248469
FK1 1013.26192	Y1 317.280186	
T 6.00000000	Y3DD-2.88914008	Y3D 12.1193707
Y3 190.313460	FD 1055.24285	Y2DD-2.88049125
Y2D 12.1043815	Y2 190.352885	FK2 997.460045
XDD-6.00098217	XD 24.9457250	X 490.111926
FK1 433.515700	Y1 380.801048	
T 8.00000000	Y3DD-1.45871043	Y3D 7.98094072
Y3 209.944687	FD 528.472294	Y2DD-1.45621945
Y2D 7.97479974	Y2 209.964422	FK2 499.298085
XDD-3.01290723	XD 16.3712078	X 530.447309
FK1 216.680234	Y1 419.976465	
T 10.00000000	Y3DD-0.85442820	Y3D 5.73461369
Y3 223.460384	FD 308.364524	Y2DD-0.85355226
Y2D 5.73175116	Y2 223.471897	FK2 291.275960
XDD-1.76086055	XD 11.7404810	X 558.145554
FK1 126.313557	Y1 446.971555	

Figure A6-4. Run-time Control and Output Stream for Aircraft Arresting Gear Problem

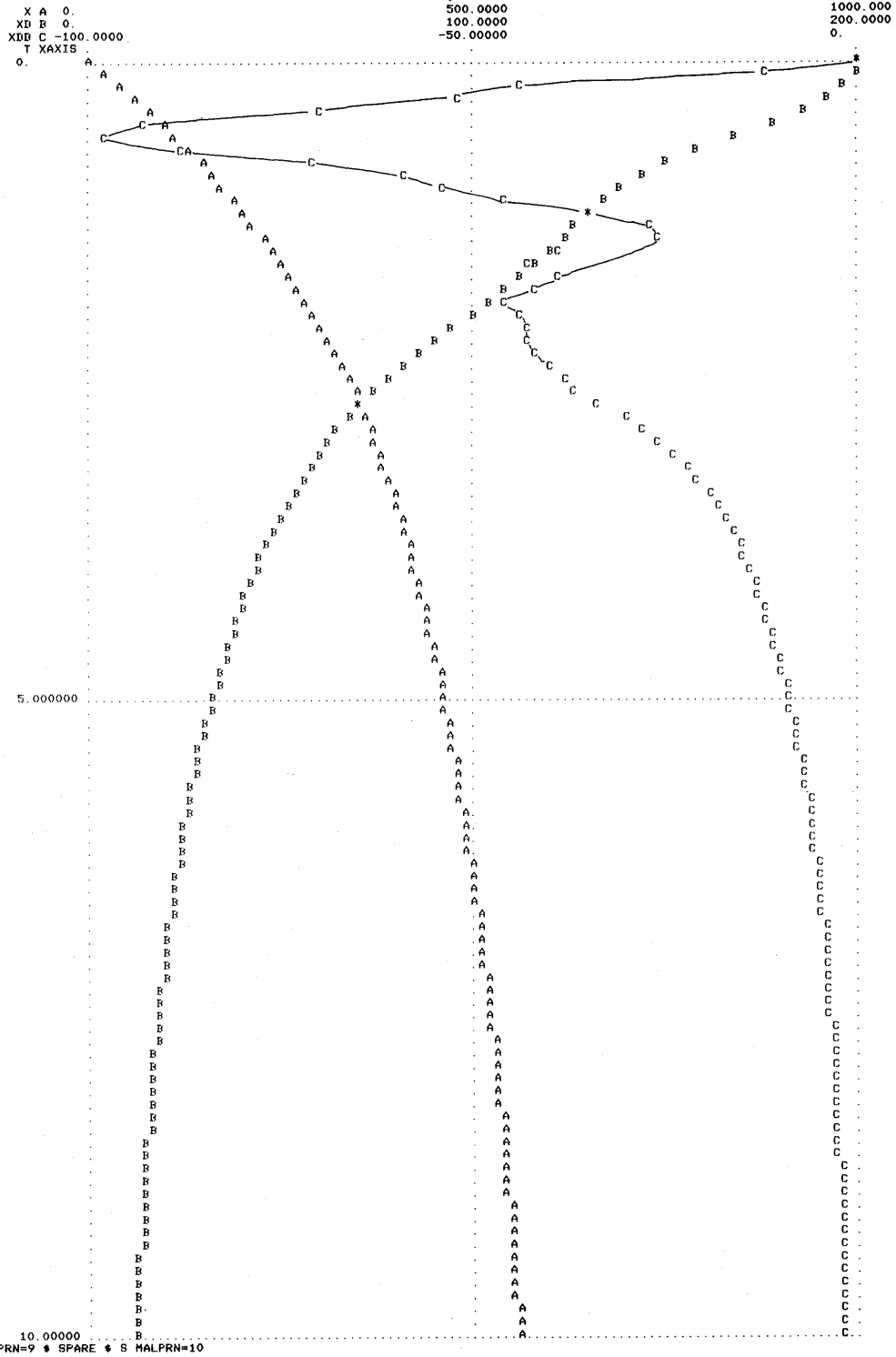


Figure A6-5. Plot of Displacement and its First and Second Derivatives for the Aircraft Arresting Gear Problem

SET SPEED = 290.0 \$ START

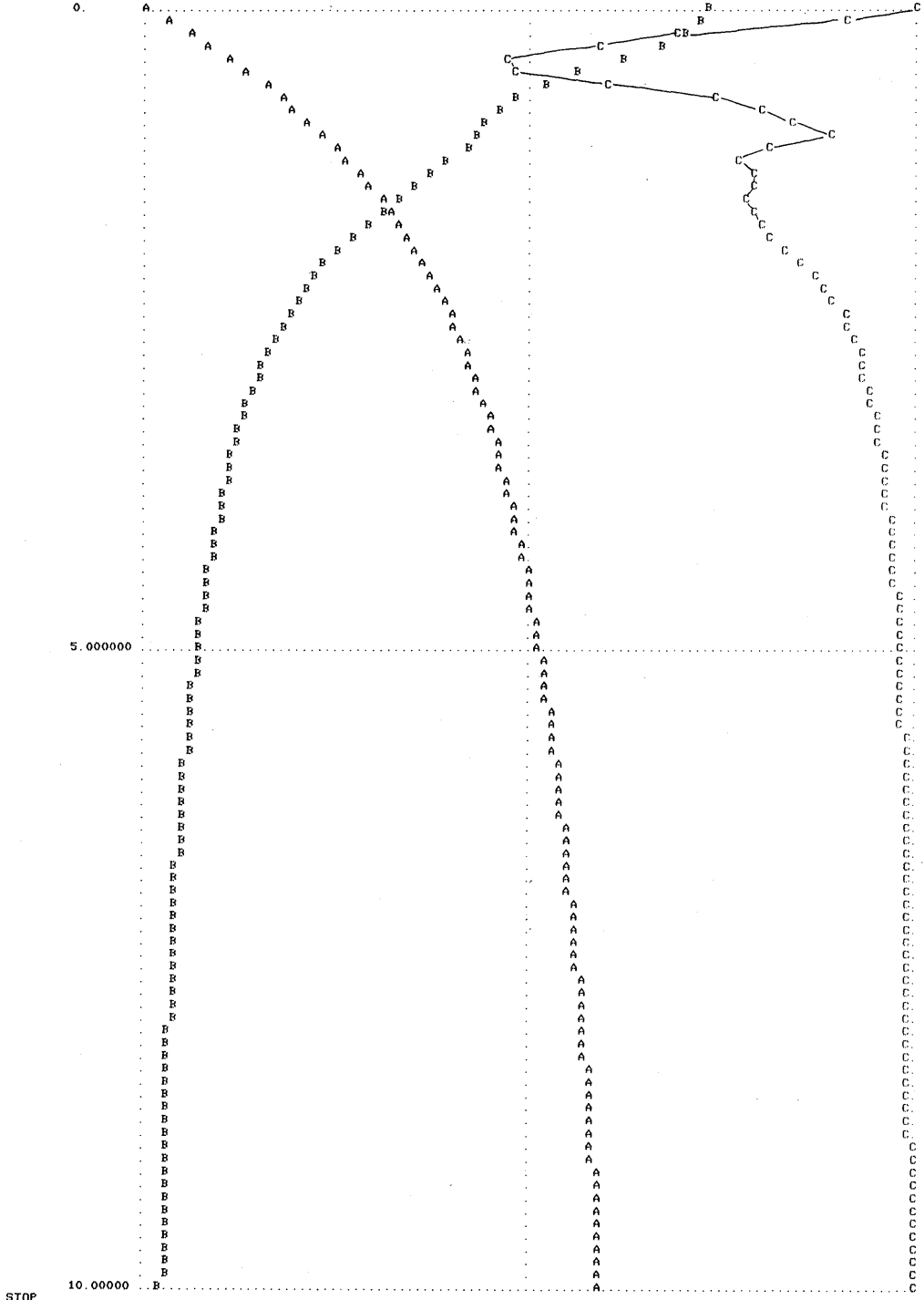
AIRCRAFT SPEED - 290.00 RUN NO. 2

T 0.	Y3DD 0.	Y3D 0.
Y3 0.	FD 0.	Y2DD 0.
Y2D 0.	Y2 0.	FK2 0.
XDD 0.	XD 290.000000	X 0.
FK1 0.	Y1 0.	
T 2.00000000	Y3DD-27.0599278	Y3D 45.5412403
Y3 128.365976	FD 10784.8237	Y2DD-29.7708033
Y2D 44.9787007	Y2 128.770862	FK2 10243.6252
XDD-60.0703751	XD 93.2928284	X 362.576923
FK1 4447.80160	Y1 258.519263	
T 4.00000000	Y3DD-6.47531783	Y3D 18.5078865
Y3 185.758816	FD 2372.55899	Y2DD-6.44284292
Y2D 18.4579190	Y2 185.847474	FK2 2243.05264
XDD-13.4898010	XD 38.0155459	X 480.930277
FK1 975.660354	Y1 371.909380	
T 6.00000000	Y3DD-2.51087770	Y3D 10.3046452
Y3 213.274391	FD 909.156899	Y2DD-2.49626943
Y2D 10.2910500	Y2 213.308341	FK2 858.939345
XDD-5.18935943	XD 21.0974966	X 537.351282
FK1 372.954133	Y1 426.698649	
T 8.00000000	Y3DD-1.24766712	Y3D 6.73035936
Y3 229.892031	FD 448.056336	Y2DD-1.24515397
Y2D 6.72508436	Y2 229.908754	FK2 423.102993
XDD-2.55891936	XD 13.7553728	X 571.343727
FK1 183.361211	Y1 459.857807	
T 10.00000000	Y3DD-0.70081981	Y3D 4.84281703
Y3 241.288925	FD 256.286240	Y2DD-0.70950784
Y2D 4.84067326	Y2 241.298501	FK2 242.269844
XDD-1.46891802	XD 9.88799079	X 594.623562
FK1 105.071665	Y1 482.620095	

SET CALPLT = .TRUE. \$"TURN ON CALCOMP PLOTS"
 S STRPLT=.T.,CALPLT=.T.
 PLOT *XTAG*=" (SEC)",X,XD,XDD

Figure A6-6. Second Run - Aircraft Arresting Gear Problem

X A 0. 500.0000 1000.000
 XD B 0. 200.0000 400.0000
 XDD C -400.0000 -200.0000 0.
 T XAXIS .



STOP 10.00000 B A
 END DISPLA -- 2792 VECTORS GENERATED IN 2 PLOT FRAMES.
 2362 WORDS TABLE SPACE USED
 4.900 CP SECONDS EXECUTION TIME.

Figure A6-7. Printer Plot to Match Line Plots in Next Two Figures

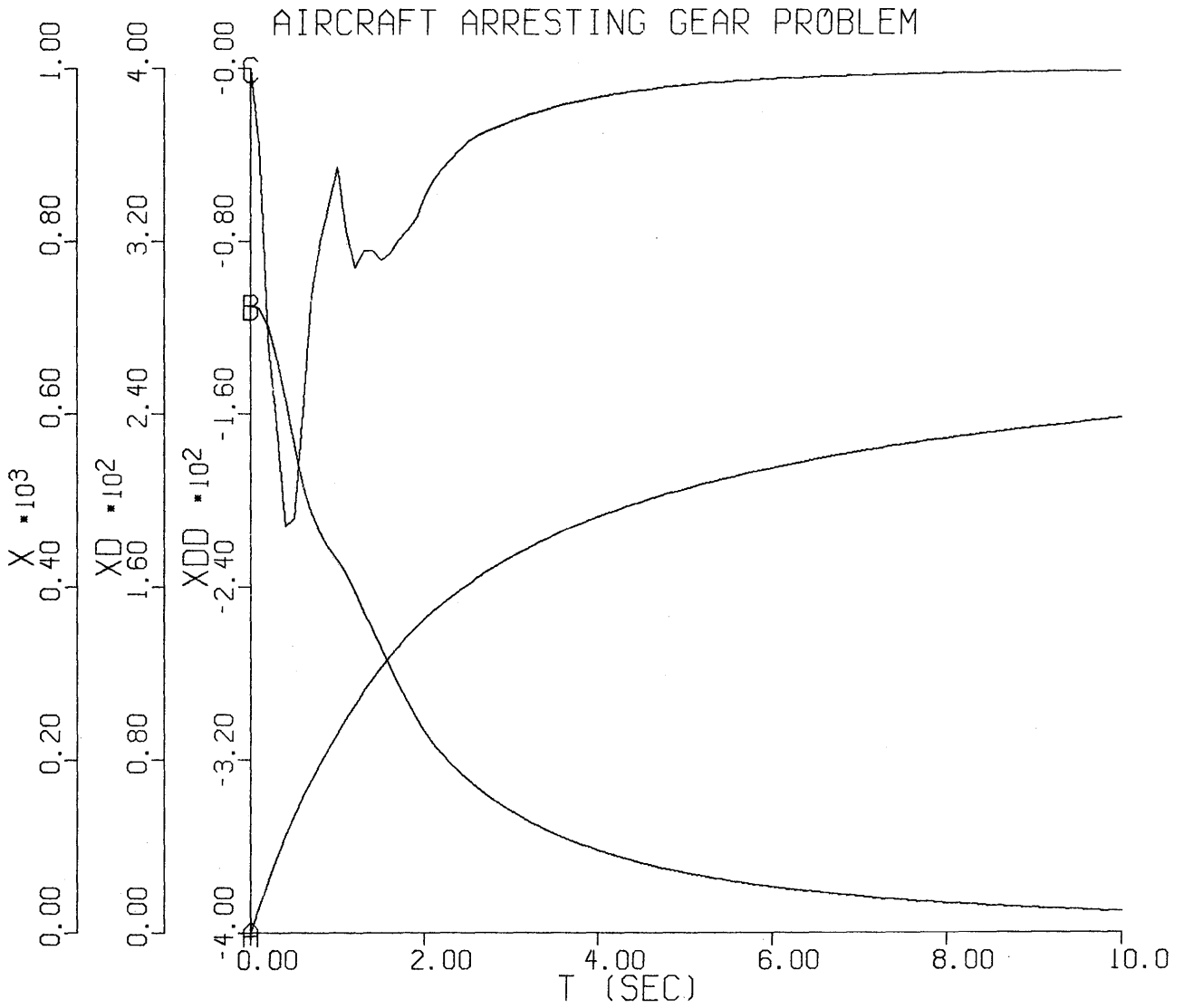


Figure A6-8. Line Plot Obtained with CALPLT = .TRUE. of Displacement, Velocity and Acceleration from the Aircraft Arresting Gear Problem

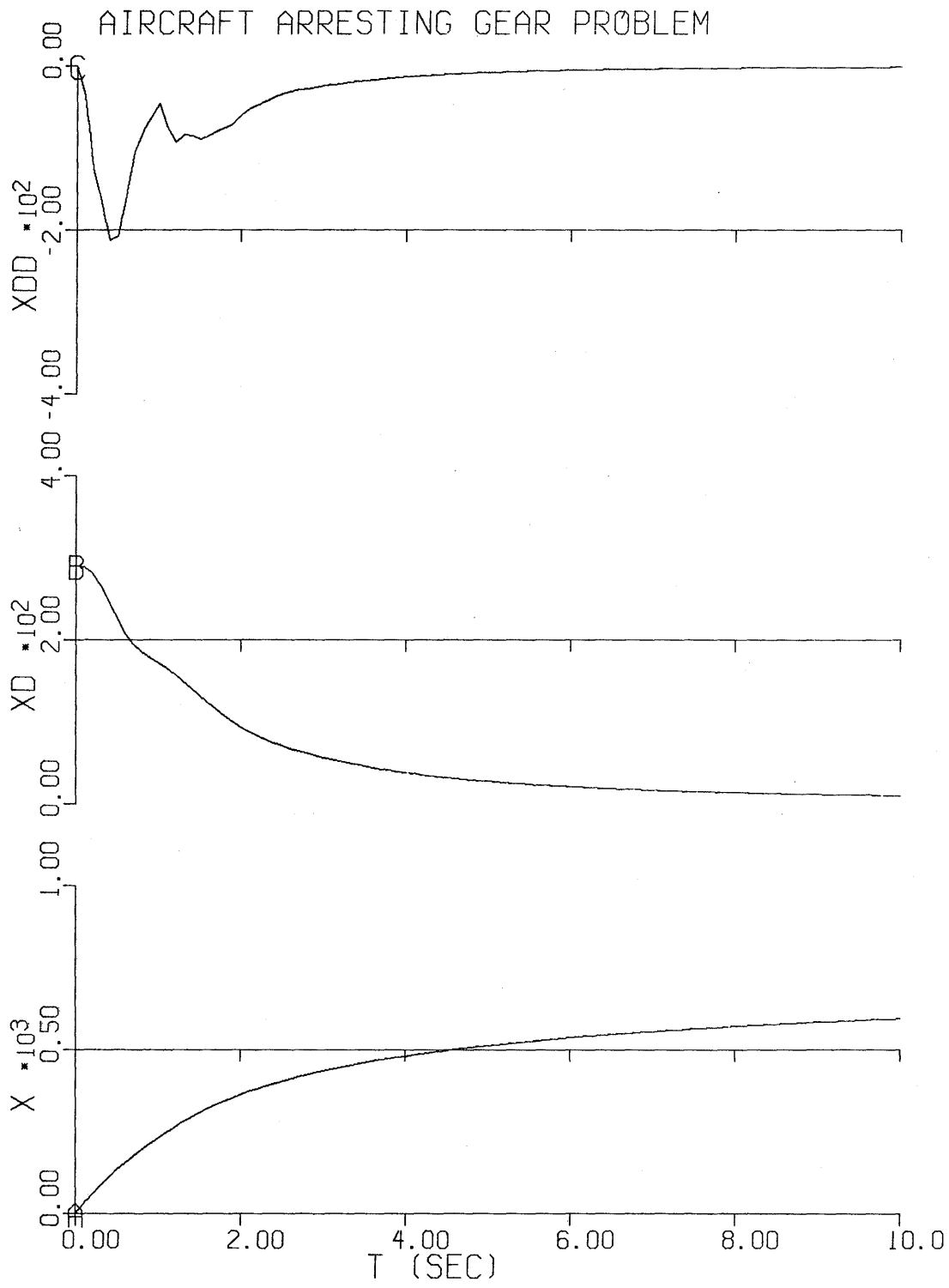


Figure A6-9. Line Plot Obtained with STRPLT = .TRUE. of Displacement, Velocity and Acceleration from the Aircraft Arresting Gear Problem

7. LONGITUDINAL STUDY

The equations describing the motion of an aircraft in three degrees-of-freedom in a longitudinal plane are given below:

Velocity

$$m\dot{V} = T \cos \alpha - D - W \sin \gamma$$

where

$$m = W/32.2 \text{ slugs}$$

$$D = qSC_D \text{ lbs}$$

$$q = 0.5 \rho V^2 \text{ lbs/sq ft}$$

$$C_D = C_{D0} + C_{D\alpha}\alpha + C_{D\sigma\epsilon} |\sigma_\epsilon|$$

Flight Path

$$mV\dot{\gamma} = L - W \cos \gamma + T \sin \alpha$$

where

$$L = qSC_L \text{ lbs}$$

$$C_L = C_{L0} + C_{L\alpha}\alpha + C_{L\sigma\epsilon}\sigma_\epsilon + (c/2V) (C_{L\dot{\alpha}}\dot{\alpha} + C_{L\dot{\theta}}\dot{\theta})$$

Pitch

$$I_y\ddot{\theta} = M$$

where

$$M = qS\bar{c}C_M \text{ ft-lbs}$$

$$C_M = C_{M0} + C_{M\alpha}\alpha + C_{M\sigma\epsilon}\sigma_\epsilon + (c/2V) (C_{M\dot{\alpha}}\dot{\alpha} + C_{M\dot{\theta}}\dot{\theta})$$

Angle of Attack

$$\dot{\alpha} = \dot{\theta} - \dot{\gamma}$$

Position

$$\dot{h} = V \sin \gamma$$

$$\dot{x} = V \cos \gamma$$

Constants

$$T = 2,000 \text{ lb}$$

$$S = 6,000 \text{ ft}^2$$

$$I_y = 27 \times 10^6 \text{ slug ft}^2$$

$$W = 500,000 \text{ lb}$$

$$\rho = 0.0023 \text{ slugs/ft}^3$$

$$\bar{c} = 30 \text{ ft}$$

Aerodynamic Coefficients

$C_{D0} = 0.14$	$C_{L0} = 1.1$	$C_{M0} = 0.05$
$C_{D\alpha} = 0.63/\text{rad}$	$C_{L\alpha} = 5.2/\text{rad}$	$C_{M\alpha} = 0.2/\text{rad}$
$C_{D\sigma\epsilon} = 0.003/\text{rad}$	$C_{L\sigma\epsilon} = 0.36/\text{rad}$	$C_{M\sigma\epsilon} = 1.4/\text{rad}$
	$C_{L\dot{\alpha}} = 2/\text{rad}/\text{sec}$	$C_{M\dot{\alpha}} = 8/\text{rad}/\text{sec}$
	$C_{L\dot{\theta}} = 5.5/\text{rad}/\text{sec}$	$C_{M\dot{\theta}} = 22/\text{rad}/\text{sec}$

Initial Conditions

$$V(0) = 200 \text{ ft/sec}$$

$$h(0) = 1500 \text{ ft}$$

$$\theta(0) = 0 \text{ rad/sec}$$

This example was chosen to illustrate an iteration to establish correct initial conditions (trim) and solution of an implicit loop.

Values of $\theta(0)$, $\gamma(0)$ and σ_ϵ are to be computed so that the aircraft is flying in a trimmed condition, i.e., the angular accelerations $\ddot{\theta}$, flight path rate $\dot{\gamma}$ and longitudinal acceleration V should all be zero. It is more usual to adjust throttle setting as thrust, T , to maintain a given flight path angle rather than the other way about.

This iteration is mechanized (see program listings, Figure A7-1 and A7-2) by choosing starting values in the INITIAL section for $\theta(0)$, $\gamma(0)$ and σ_ϵ (zero seems good enough) and then in the DYNAMIC, after the first evaluation of derivatives, checking if a weighted mean square error

$$E = (V)^2 + (20 \ddot{\theta})^2 + (100 \dot{\gamma})^2$$

is less than a maximum allowable error (0.1).

If not the initial guess is corrected by

$$\gamma(0) = \gamma(0) + 0.02 \dot{V}$$

$$\theta(0) = \theta(0) - 1.0 \dot{\gamma}$$

$$\sigma_\epsilon(0) = \sigma_\epsilon(0) + 2.0 \ddot{\theta}$$

A more precise (and faster) iteration would calculate the derivative of the error vector (V, γ, θ) with respect to the control vector $(\gamma, \theta, \sigma_\epsilon)$ three by three Jacobian matrix resulting - say (J) is then inverted and the new control vector is given by

$$(\gamma, \theta, \sigma_\epsilon)^T = (\gamma, \theta, \sigma_\epsilon)^T - J^{-1} (\dot{V}, \dot{\gamma}, \ddot{\theta})^T$$

However, the simple iteration given suffices for this fixed example.

Once convergence has been satisfied, then the variable START is set false and the error check is never examined again. Note that in order to recompute the derivatives the program cycles from the DYNAMIC section back to the INITIAL section. The integration routine is set up and derivatives calculated at the transition from INITIAL to the DYNAMIC.

The other feature is the use of the IMPLICIT operator to calculate angle-of-attack rate. Angle-of-attack rate is the difference between body rate and flight path rate. Flight path rate depends on lift which in turn is dependent on the coefficient C_L . This coefficient includes angle-of-attack rate so forming an algebraic or implicit loop. To avoid the sorting problem the IMPL operator is used so

$$\dot{\alpha} = \text{IMPL}(\dot{\theta}, 0.001, 10, \text{ef}, \dot{\theta} - \dot{\gamma}, 0.01)$$

This means take a first guess for $\dot{\alpha}$ to be $\dot{\theta}$ (body rate). Then iterate using Newton-Raphson until differences in successive values of $\dot{\alpha}$ are less than 0.0001. Try at most ten cycles and set ef (error flag) nonzero if this number is exceeded. This iteration is a time consuming business and has to be performed at each derivative evaluation. In this case it would have been much better to solve algebraically for α before writing the model equations - since the equations are linear, this would have been possible. On the other hand, if C_L had been given in functional form as a table, i.e., $C_L(\alpha, \dot{\alpha})$ then the implicit operator would have had to be used.

The output stream from executing the model is shown in Figure A7-3 through A7-7. Output is established at every ten communication intervals and a similar set of variables PREPARED for later plotting. The START initiates the run and the iteration is reported in the next eight lines of output during which the error, ERR, is reduced from 55.4 to 0.1. Then the run starts and the block of OUTPUT variables is written out every 0.5 seconds. At the end of the run (TMX=6.49 seconds), two PRINT statements are used in order to list selected variables, the first at every four communication intervals (0.2 seconds) and the second at every eight (0.4 seconds). At the bottom of Figure A7-5 a strip plot is set up (STRPLT=.T.); the scale factor is reduced to a half (PSFSPL=0.5); the x-axis length is increased to ten inches (XINSPL=10.0) and the distance between tick marks is increased to two inches (XTISPL=2.0). With the scale factor of a half that keeps the x-axis length to five inches but reduces the strip width to one inch, allowing seven variables to be stacked on one frame. The first PLOT command only changes the 'XHI' value and ensures that the plot will cover the full extent of the x-axis. The second PLOT actually draws the picture of Figure A7-8. If the first PLOT command had not been used, normal rounding would have made the x-axis run from zero to ten so wasting 35% of the area. Of course in this case the tick marks are not whole numbers but that's a user choice between the two ways of forming the plot. We would prefer maximum area utilization.

The last sequence compares the standard Runge-Kutta fourth order integration routine with the Adams-Moulton variable order, variable step. The OUTPUT list is cleared ('CLEAR') and reestablished as TIME, CIOITG (current integration order) and CSSITG (current step size) - printing out every thirty communication intervals. The SPARE \$ START \$ SPARE sequence runs the simulation with the default algorithm (IALG = 5, Runge-Kutta fourth order) and shows an elapsed central processor time of about 6.57 seconds. Then IALG is changed to one (Adams-Moulton) and the program run again. The integration order rises to four and the step size to a maximum of 0.0601 seconds and since this is greater than the communication interval (0.05 seconds) it won't change from that value. Note that the actual step size will be equal to the communication interval value of 0.05 seconds, since these points are needed for data recording. The current step size variable CSSITG contains the step size the integration algorithm would like to take, in the absence of any other constraints such as an upcoming event or communication interval which may reduce it from this value. From the report at the end of the Adams-Moulton variable step run (Figure A7-6) it can be deduced that θ (TH) was the controlling integrator and the minus sign (-) indicates that the relative error never exceeded the absolute error specified. The allowable error was thus set at 0.0001. From the elapsed time listed of about 0.73 seconds, the Adams-Moulton integrator was nearly ten times faster. At the bottom of Figure A7-6, the results of the Adams-Moulton integrator run are printed out at one second intervals to be compared with the same variables printed in Figures A7-4 and A7-5. Agreement is within two or three decimal places.

In Figure A7-7 the use of the command ANALYZ is shown which prints out the Jacobian and its eigen values. The largest eigen value is 0.82 per second, which implies a rather slowly varying system and in actual fact promises good results with integration step sizes of the reciprocal of this value or about one second, a further factor of twenty increase in solution speed. In this case the constant constraint on the step size is the data recording interval chosen to produce acceptable plots.

PROGRAM AIRCRAFT LONGITUDINAL STABILITY STUDY

INITIAL

```

"-----DEFINE ALL PRESET VARIABLES"
CONSTANT      T = 2000.0   , IY = 27.0E6   , RO = 2.3E-3   ...
               , S = 6000.0   , W = 500000.0 , CB = 30.0   ...
               , G = 32.2
CONSTANT      CDZ = 0.14   , CDAL = 0.63   , CDDE = 0.003 ...
               , CLZ = 1.1    , CLAL = 5.2    , CLDE = 0.36 ...
               , CMZ = -0.5   , CMAL = -0.2   , CMDE = -1.4 ...
               , CLAD = 2.0   , CLTD = 5.5   , K1 = 0.02 ...
               , CMAD = -8.0  , CMTD = -22.0 , K2 = -1.0 ...
               , K3 = 2.0    , ERMX = 0.1   , TMX = 6.49 ...
CONSTANT      VZ = 200.0   , HZ = 1500.0  , QZ = 0.0    ...
               , XZ = 0.0    , TZ = 0.5
CINTERVAL     CINT = 0.05
"-----CHANGE NAME OF INDEPENDENT VARIABLE"
"          TO AVOID NAME CONFLICT"
VARIABLE      TIME = 0.0
LOGICAL       START
"-----NEED MASS FROM WEIGHT IN LBS"
MASS          = W/G
"-----SET INITIAL GUES FOR INITIAL CONDITIONS"
GAZ           = 0.0
THZ           = 0.0
DLZ           = 0.0
"-----START WILL BE MADE TRUE WHEN ITERATION"
"          CONVERGES"
START        = .FALSE.
I1..CONTINUE

```

END \$*OF INITIAL*

```

"-----THE TRANSITION FROM INITIAL TO DYNAMIC "
"          TRANSFERS ALL INITIAL CONDITIONS TO THE "
"          STATE VARIABLES AND EVALUATES THE CODE "
"          IN THE DERIVATIVE SECTION ONCE "

```

DYNAMIC

DERIVATIVE

```

"-----FIN DEFLECTION IS KICKED TO EXCITE SYSTEM"
DLE          = 0.1*STEP(TZ) + DLZ
"-----ANGLE OF ATTACK"
AL           = TH - GA
"-----DRAG COEFFICIENT"
CD           = CDZ + CDAL*AL + CDDE*ABS(DLE)
"-----LIFT COEFFICIENT"
CL           = CLZ + CLAL*AL + CLDE*DLE + (CB/(2*V))*((CLAD*ALD ...
               + CLTD*Q)
"-----DYNAMIC PRESSURE"
QP           = 0.5*RO*V**2
"-----DRAG AND LIFT"
D            = QP*S*CD

```

Figure A7-1. Listing of Model Definition for Aircraft Longitudinal Study

```

L      = QP*S*CL
"-----FLIGHT PATH RATE"
GAD    = (L - W*COS(GA) + T*SIN(AL))/(MASS*V)
"-----LONGITUTINAL VELOCITY RATE"
VD     = (T*COS(AL) - D - W*SIN(GA))/MASS
"-----PITCH MOMENT COEFFICIENT"
CM     = CMZ + CMAL*AL + CMDE*DLE + (CB/(2*V))*(CMAD*ALD
      + CMTD*Q)
"-----PITCHING MOMENT"
M      = QP*S*CB*CM
"-----NEED PITCH RATE DERIVATIVE EXPLICITLY "
"-----FOR ITERATION "
QD     = M/IY
"-----IMPLICIT LOOP FOR ANGLE OF ATTACK RATE"
ALD    = IMPL(Q, 0.0001, 10, EF, Q - GAD, 0.01)
Q      = INTVC(QD, QZ)  $" PITCH RATE "
TH     = INTEG(Q, THZ)  $" PITCH ANGLE "
V      = INTVC(VD, VZ)  $" VELOCITY "
GA     = INTVC(GAD, GAZ) $" FLIGHT PATH ANGLE "
H      = INTEG(V*SIN(GA), HZ) $" HEIGHT "
X      = INTEG(V*COS(GA), XZ) $" HORIZONTAL DISTANCE TRAVELLED "

END  $"OF DERIVATIVE"

"-----IF ITERATION CONVERGED"
IF(START) GO TO I1
"-----WEIGHTED ERROR FROM TRIM"
ERROR  = VD**2 + (20.0*QD)**2 + (100.0*GAD)**2
"-----IF WITHIN TOLERANCE"
START  = ERROR .LE. ERMX
"-----COMPUTE NEW TRIAL VALUES"
GAZ    = GAZ + K1*VD
THZ    = THZ + K2*GAD
DLZ    = DLZ + K3*QD
"-----PRINT ITERATION INFORMATION"
LINES(1) $" INFORM SYSTEM ABOUT TO PRINT ONE LINE "
PRINT 99, GAZ, THZ, DLZ, ERROR
99..FORMAT(5H GAZ ,E12.4,5H THZ ,E12.4,5H DLE ,E12.4,5H ERR ,F10.1)
"-----RETURN TO INITIAL REGION TO RESTART"
"-----THE INTEGRATION ALGORITHM"
GO TO I1
I1..CONTINUE

"-----EXPRESS STOPPING CRITERIA"
TERMT(TIME.GE.TMX)

END  $"OF DYNAMIC"

END  $"OF PROGRAM"

```

Figure A7-2. Listing of Model Definition for Aircraft Longitudinal Study

```

SET TITLE = "LONGITUDINAL AIRCRAFT STABILITY STUDY"
S TCWPRN=80 $* FORCE 3 COLUMN OUTPUT WIDTH *
OUTPUT TIME, M, Q, VD, GAD, DLE, X, V, TH, GA, AL, EF, ALD, "NCIOUT"=10
PREPAR TIME, M, Q, VD, GAD, DLE, V, TH, GA, AL
SET NSTP = 10
START
GAZ -.4719E-01 THZ .6241E-01 DLE -.3296E+00 ERR 55.4
GAZ -.8892E-01 THZ .8499E-01 DLE -.3750E+00 ERR 9.7
GAZ -.1183E+00 THZ .7919E-01 DLE -.3789E+00 ERR 2.5
GAZ -.1342E+00 THZ .6226E-01 DLE -.3782E+00 ERR 3.5
GAZ -.1397E+00 THZ .4547E-01 DLE -.3780E+00 ERR 2.9
GAZ -.1391E+00 THZ .3372E-01 DLE -.3785E+00 ERR 1.4
GAZ -.1362E+00 THZ .2761E-01 DLE -.3791E+00 ERR .4
GAZ -.1330E+00 THZ .2573E-01 DLE -.3796E+00 ERR .1
TIME 0. M-4483.15766 Q 0.
VD 0.11209712 GAD-4.7803E-04 DLE-0.37964391
X 0. V 200.000000 TH 0.02573051
GA-0.13304999 AL 0.15878051 EF 0.
ALD 4.7803E-04
TIME 0.50000000 M-2709.32654 Q-6.5554E-05
VD 0.11389061 GAD-3.0769E-04 DLE-0.37964391
X 99.1287839 V 200.056557 TH 0.02571276
GA-0.13324424 AL 0.15895699 EF 0.
ALD 2.4213E-04
TIME 1.00000000 M-819443.768 Q-0.01806435
VD 0.15529338 GAD-4.6372E-04 DLE-0.27964391
X 198.291893 V 200.119570 TH 0.02093651
GA-0.13252507 AL 0.15346158 EF 0.
ALD-0.01760063
TIME 1.50000000 M-588537.231 Q-0.03097716
VD 0.32194985 GAD-0.00587615 DLE-0.27964391
X 297.491969 V 200.233657 TH 0.00849834
GA-0.13405542 AL 0.14255376 EF 0.
ALD-0.02510101
TIME 2.00000000 M-425184.786 Q-0.04027717
VD 0.60696380 GAD-0.01205182 DLE-0.27964391
X 396.734299 V 200.461263 TH-0.00944102
GA-0.13852553 AL 0.12908451 EF 0.
ALD-0.02822535
TIME 2.50000000 M-308869.266 Q-0.04701386
VD 0.99482728 GAD-0.01819030 DLE-0.27964391
X 496.045171 V 200.857767 TH-0.03135335
GA-0.14609914 AL 0.11474580 EF 0.
ALD-0.02882471
TIME 3.00000000 M-225422.635 Q-0.05191891
VD 1.46906702 GAD-0.02383891 DLE-0.27964391
X 595.470475 V 201.470484 TH-0.05615080
GA-0.15663296 AL 0.10048216 EF 0.
ALD-0.02807999

```

Figure A7-3. Run-time Drive Commands and Output Stream for Aircraft Longitudinal Study

TIME 3.50000000	M-165020.639	Q-0.05550448
VD 2.01355780	GAD-0.02876969	DLE-0.27964391
X 695.072274	V 202.338536	TH-0.08305317
GA-0.16981764	AL 0.08676447	EF 0.
ALD-0.02673479		
TIME 4.00000000	M-120892.038	Q-0.05813081
VD 2.61327924	GAD-0.03289737	DLE-0.27964391
X 794.925011	V 203.493238	TH-0.11149598
GA-0.18526832	AL 0.07377233	EF 0.
ALD-0.02523344		
TIME 4.50000000	M-88350.2552	Q-0.06005320
VD 3.25474200	GAD-0.03622554	DLE-0.27964391
X 895.112100	V 204.958763	TH-0.14106705
GA-0.20258148	AL 0.06151442	EF 0.
ALD-0.02382766		
TIME 5.00000000	M-64151.8200	Q-0.06145434
VD 3.92621180	GAD-0.03881095	DLE-0.27964391
X 995.723088	V 206.752971	TH-0.17146258
GA-0.22136990	AL 0.04990732	EF 0.
ALD-0.02264339		
TIME 5.50000000	M-46048.3669	Q-0.06246668
VD 4.61780037	GAD-0.04074011	DLE-0.27964391
X 1096.85137	V 208.888317	TH-0.20245679
GA-0.24128298	AL 0.03882619	EF 0.
ALD-0.02172656		
TIME 6.00000000	M-32476.6366	Q-0.06318772
VD 5.32146229	GAD-0.04211360	DLE-0.27964391
X 1198.59233	V 211.372774	TH-0.23388084
GA-0.26201738	AL 0.02813654	EF 0.
ALD-0.02107412		
TIME 6.50000000	M-22335.8613	Q-0.06369065
VD 6.03092082	GAD-0.04303576	DLE-0.27964391
X 1301.04170	V 214.210737	TH-0.26560825
GA-0.28332137	AL 0.01771313	EF 0.
ALD-0.02065650		

* MAKE COLUMN PRINT OF SELECTED VARIABLES FROM PREPAR LIST *
 PRINT "NCIPRN"=4, TIME, M, Q, VD, GAD

LINE	TIME	M	Q	VD	GAD
0	0.	-4483.1577	0.	0.1120971	-4.780E-04
4	0.2000000	-3691.5678	-3.020E-05	0.1129881	-4.035E-04
8	0.4000000	-3012.0301	-5.497E-05	0.1136435	-3.377E-04
12	0.6000000	-1.072E+06	-0.0041394	0.1156370	0.0024458
16	0.8000000	-936967.82	-0.0115693	0.1247060	0.0012126
20	1.0000000	-819443.77	-0.0180643	0.1552934	-4.637E-04
24	1.2000000	-717322.81	-0.0237472	0.2068592	-0.0024664
28	1.4000000	-628518.65	-0.0287242	0.2786918	-0.0046991

Figure A7-4. Output Stream from Aircraft Longitudinal Study

LINE	TIME	M	Q	VD	GAD
32	1.6000000	-551229.33	-0.0330870	0.3699518	-0.0070821
36	1.8000000	-483899.01	-0.0369151	0.4797075	-0.0095506
40	2.0000000	-425184.79	-0.0402772	0.6069638	-0.0120518
44	2.2000000	-373927.76	-0.0432326	0.7506856	-0.0145433
48	2.4000000	-329128.86	-0.0458328	0.9098160	-0.0169912
52	2.6000000	-289923.07	-0.0481223	1.0832922	-0.0193691
56	2.8000000	-255568.70	-0.0501399	1.2700565	-0.0216568
60	3.0000000	-225422.63	-0.0519189	1.4690670	-0.0238389
64	3.2000000	-198931.41	-0.0534885	1.6793048	-0.0259047
68	3.4000000	-175617.33	-0.0548739	1.8997806	-0.0278466
72	3.6000000	-155068.31	-0.0560970	2.1295392	-0.0296604
76	3.8000000	-136928.80	-0.0571771	2.3676639	-0.0313440
80	4.0000000	-120892.04	-0.0581308	2.6132792	-0.0328974
84	4.2000000	-106693.28	-0.0589727	2.8655533	-0.0343223
88	4.4000000	-94104.044	-0.0597154	3.1236997	-0.0356217
92	4.6000000	-82927.073	-0.0603703	3.3869786	-0.0367996
96	4.8000000	-72992.043	-0.0609470	3.6546976	-0.0378608
100	5.0000000	-64151.820	-0.0614543	3.9262118	-0.0388110
104	5.2000000	-56279.255	-0.0618998	4.2009243	-0.0396559
108	5.4000000	-49264.402	-0.0622902	4.4782854	-0.0404021
112	5.6000000	-43012.123	-0.0626315	4.7577925	-0.0410559
116	5.8000000	-37440.018	-0.0629291	5.0389888	-0.0416242
120	6.0000000	-32476.637	-0.0631877	5.3214623	-0.0421136
124	6.2000000	-28059.936	-0.0634116	5.6048448	-0.0425308
128	6.4000000	-24126.843	-0.0636046	5.8888103	-0.0428825

PRINT *NCIPRN*=8, TIME, DLE, TH, GA, AL

LINE	TIME	DLE	TH	GA	AL
0	0.	-0.3796439	0.0257305	-0.1330500	0.1587805
8	0.4000000	-0.3796439	0.0257188	-0.1332120	0.1589308
16	0.8000000	-0.2796439	0.0239144	-0.1326063	0.1565207
24	1.2000000	-0.2796439	0.0167427	-0.1328135	0.1495563
32	1.6000000	-0.2796439	0.0052940	-0.1347031	0.1399971
40	2.0000000	-0.2796439	-0.0094410	-0.1385255	0.1290845
48	2.4000000	-0.2796439	-0.0267104	-0.1443399	0.1176295
56	2.8000000	-0.2796439	-0.0459412	-0.1520815	0.1061403
64	3.2000000	-0.2796439	-0.0666948	-0.1616093	0.0949145
72	3.6000000	-0.2796439	-0.0886336	-0.1727394	0.0841059
80	4.0000000	-0.2796439	-0.1114960	-0.1852683	0.0737723
88	4.4000000	-0.2796439	-0.1350784	-0.1989889	0.0639104
96	4.8000000	-0.2796439	-0.1592214	-0.2137009	0.0544796
104	5.2000000	-0.2796439	-0.1837990	-0.2292183	0.0454193
112	5.6000000	-0.2796439	-0.2087118	-0.2453730	0.0366612
120	6.0000000	-0.2796439	-0.2338808	-0.2620174	0.0281365
128	6.4000000	-0.2796439	-0.2592434	-0.2790253	0.0197819

S STRPLT=.T., PRNPLT=.F., PSFSPL=0.5, XINGSPL=10.0, XTISPL=2.0
 PLOT *XHI*=TMX \$* PLOT NOTHING - BUT ESTABLISH FULL RANGE FOR SCALE *
 PLOT TH, AL, GA, GAD, Q, M, DLE
 * CHANGE OUTPUT LIST FOR TIME TRIAL BETWEEN RK4 AND ADAMS-MOULTON *
 OUTPUT *CLEAR*, TIME, CIOITG, CSSITG, *NCIOUT*=30
 SPARE \$ START \$ SPARE
 ACCUMULATED CP TIME 21.259 SEC. ELAPSED CP TIME 21.259 SEC.
 GAZ -.4719E-01 THZ .6241E-01 DLE -.3296E+00 ERR 55.4
 GAZ -.8892E-01 THZ .8499E-01 DLE -.3750E+00 ERR 9.7

Figure A7-5. Output Stream from Aircraft Longitudinal Study

GAZ	-.1183E+00 THZ	.7919E-01 DLE	-.3789E+00 ERR	2.5
GAZ	-.1342E+00 THZ	.6226E-01 DLE	-.3782E+00 ERR	3.5
GAZ	-.1397E+00 THZ	.4547E-01 DLE	-.3780E+00 ERR	2.9
GAZ	-.1391E+00 THZ	.3372E-01 DLE	-.3785E+00 ERR	1.4
GAZ	-.1362E+00 THZ	.2761E-01 DLE	-.3791E+00 ERR	.4
GAZ	-.1330E+00 THZ	.2573E-01 DLE	-.3796E+00 ERR	.1
	TIME 0.	CIOITG	4	CSSITG 0.00500000
	TIME 1.50000000	CIOITG	4	CSSITG 0.00500000
	TIME 3.00000000	CIOITG	4	CSSITG 0.00500000
	TIME 4.50000000	CIOITG	4	CSSITG 0.00500000
	TIME 6.00000000	CIOITG	4	CSSITG 0.00500000
	TIME 6.50000000	CIOITG	4	CSSITG 0.00500000

ACCUMULATED CP TIME 27.829 SEC. ELAPSED CP TIME 6.570 SEC.

SET IALG = 1 \$" TRY ADAMS-MOULTON INTEGRATION "

START \$ SPARE \$" COMPARE TIMES "

GAZ	-.4719E-01 THZ	.6241E-01 DLE	-.3296E+00 ERR	55.4
GAZ	-.8892E-01 THZ	.8499E-01 DLE	-.3750E+00 ERR	9.7
GAZ	-.1183E+00 THZ	.7919E-01 DLE	-.3789E+00 ERR	2.5
GAZ	-.1342E+00 THZ	.6226E-01 DLE	-.3782E+00 ERR	3.5
GAZ	-.1397E+00 THZ	.4547E-01 DLE	-.3780E+00 ERR	2.9
GAZ	-.1391E+00 THZ	.3372E-01 DLE	-.3785E+00 ERR	1.4
GAZ	-.1362E+00 THZ	.2761E-01 DLE	-.3791E+00 ERR	.4
GAZ	-.1330E+00 THZ	.2573E-01 DLE	-.3796E+00 ERR	.1
	TIME 0.	CIOITG	1	CSSITG 0.00500000
	TIME 1.50000000	CIOITG	3	CSSITG 0.06012383
	TIME 3.00000000	CIOITG	3	CSSITG 0.06012383
	TIME 4.50000000	CIOITG	4	CSSITG 0.06012383
	TIME 6.00000000	CIOITG	4	CSSITG 0.06012383
	TIME 6.50000000	CIOITG	4	CSSITG 0.06012383

COUNT OF TIMES STATE CONTROLLED STEP SIZE

MINUS (-) REL ERR ALWAYS BELOW ABS ERR

GA PC FAIL	0	ERR CONTROL	15-
H PC FAIL	0	ERR CONTROL	0
Q PC FAIL	0	ERR CONTROL	7-
TH PC FAIL	0	ERR CONTROL	110-
V PC FAIL	0	ERR CONTROL	0
X PC FAIL	0	ERR CONTROL	10

ACCUMULATED CP TIME 28.557 SEC. ELAPSED CP TIME .728 SEC.

PRINT "NCIPRN"=20, TIME, M, Q, VD, GAD

LINE	TIME	M	Q	VD	GAD
0	0.	-4483.1577	0.	0.1120971	-4.780E-04
20	1.0000000	-818960.30	-0.0180910	0.1554642	-4.701E-04
40	2.0000000	-424940.72	-0.0402911	0.6075197	-0.0120617
60	3.0000000	-225297.56	-0.0519262	1.4699101	-0.0238476
80	4.0000000	-120825.69	-0.0581347	2.6143087	-0.0329036
100	5.0000000	-64115.362	-0.0614564	3.9273439	-0.0388147
120	6.0000000	-32456.280	-0.0631888	5.3226369	-0.0421155

Figure A7-6. Output Stream from Aircraft Longitudinal Study

ANALYZ "FREEZE"=X, "JACOB", "EIGEN"

ROW VECTOR NAMES

GA	1	H	2	Q	3
TH	4	V	5		

COLUMN VECTOR NAMES

GAD	1	Z09985	2	QD	3
Z09986	4	VD	5		

MATRIX ELEMENTS - ROWS ACROSS, COLUMNS DOWN

1	-0.5305362	0.	0.0493329	0.4890637	0.0011434
2	205.67325	0.	0.	0.	-0.2795804
3	-0.0341951	0.	-0.7293217	0.0260218	3.747E-05
4	0.	0.	1.0000000	0.	0.
5	-18.067358	0.	0.	-12.848579	-0.0289350

COMPLEX EIGEN VALUES IN ASCENDING ORDER

1	0.	0.
2	0.01093535	0.
3	-0.11219790	0.
4	-0.36600676	0.
5	-0.82152364	0.

STOP

Figure A7-7. Output Stream from Aircraft Longitudinal Study

Don't be misled by the ratio of execution for Runge-Kutta to Adams-Moulton. The artificially small step size chosen arbitrarily for the RK4 algorithm unfairly penalizes the comparison. In actual tests with typical non-linear systems, the second order Runge-Kutta method (IALG=4) did best in terms of CP seconds per simulated second. The penalty is that the user must choose the step size, normally by experimentation, but sometimes by familiarity with the model. As mentioned above, a good rule of thumb is to make the integration step size (MAXT) equal to the smallest time constant in the model. Of course if the controlling time constants change significantly, then the variable step algorithms are called for. Stiff systems with fast initial transients should use integration algorithm two (IALG=2).

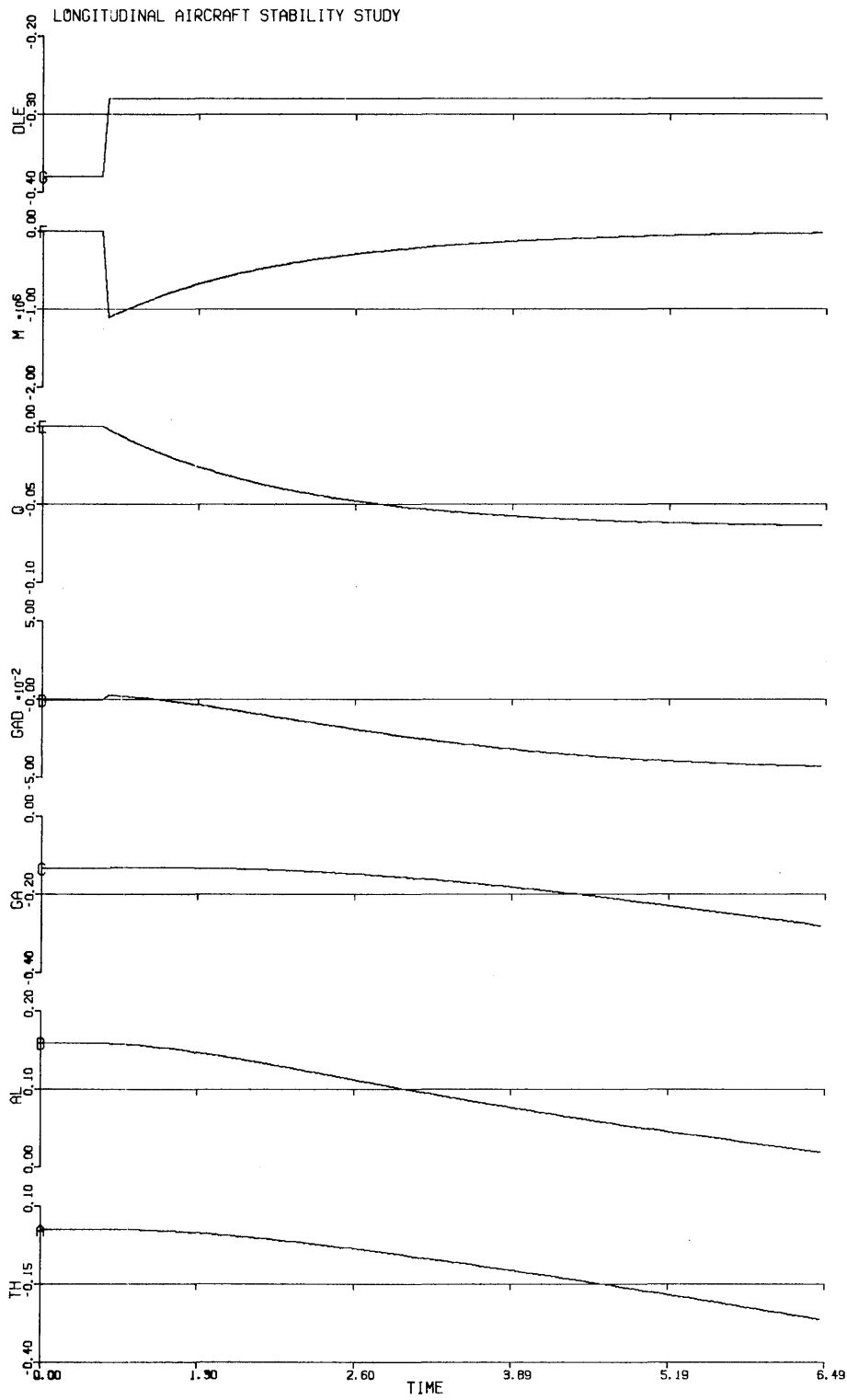


Figure A7-8. Strip Plot of Selected Variables from Aircraft Longitudinal Study

8. PHYSBE

A benchmark simulation model has been described* in order to demonstrate the different methods of solving simulation problems. Called PHYSBE, for physiological simulation benchmark experiment, this system models the blood flow around a human body, driven by two pumps - the right and left ventricles in the heart. Figure A8-1 shows the interconnection of the various components of the system which are to be considered as large bags or balloons that can be filled with blood. The characteristic is that the more blood that is forced in, the higher the pressure. Valves exist between the vena cava (VC) and the right ventricle (RV); the right ventricle and the lungs (LN); the lungs and the left ventricle (LV), and the left ventricle and the aorta (AO). These valves only allow blood flow in the forward direction.

The blood is driven around the loop by changing the compliance of the right and left ventricles as a function of time, so modeling the squeezing of the blood in the chambers by the heart muscle as it contracts and releases. Figure A8-2 shows this reciprocal compliance as a function of time; this function is repeated every second as the heart beats. Each lump contains mass balance and heat balance constraints which are described by the following equations.

The assumptions made are:

- 1) Physical parameters of the system are linear.
- 2) Blood flow within each area is influenced only by:
 - a) Inlet Pressure
 - b) Inflow Resistance
 - c) Compliance (Volume/Unit Pressure)
 - d) Outflow Resistance
 - e) Outlet Pressure
- 3) There is no resistance to blood flow within areas.
- 4) All endogenous heat will be absorbed by the blood and conducted by the blood.
- 5) Specific heat of blood and all body components is unity.
- 6) Temperature change within the arteries, ventricles, and great veins is negligible.

* McLeod, J., "PHYSBE: A Physiological Simulation Benchmark Experiment", *SIMULATION* 1. pp. 324-329, 1966.

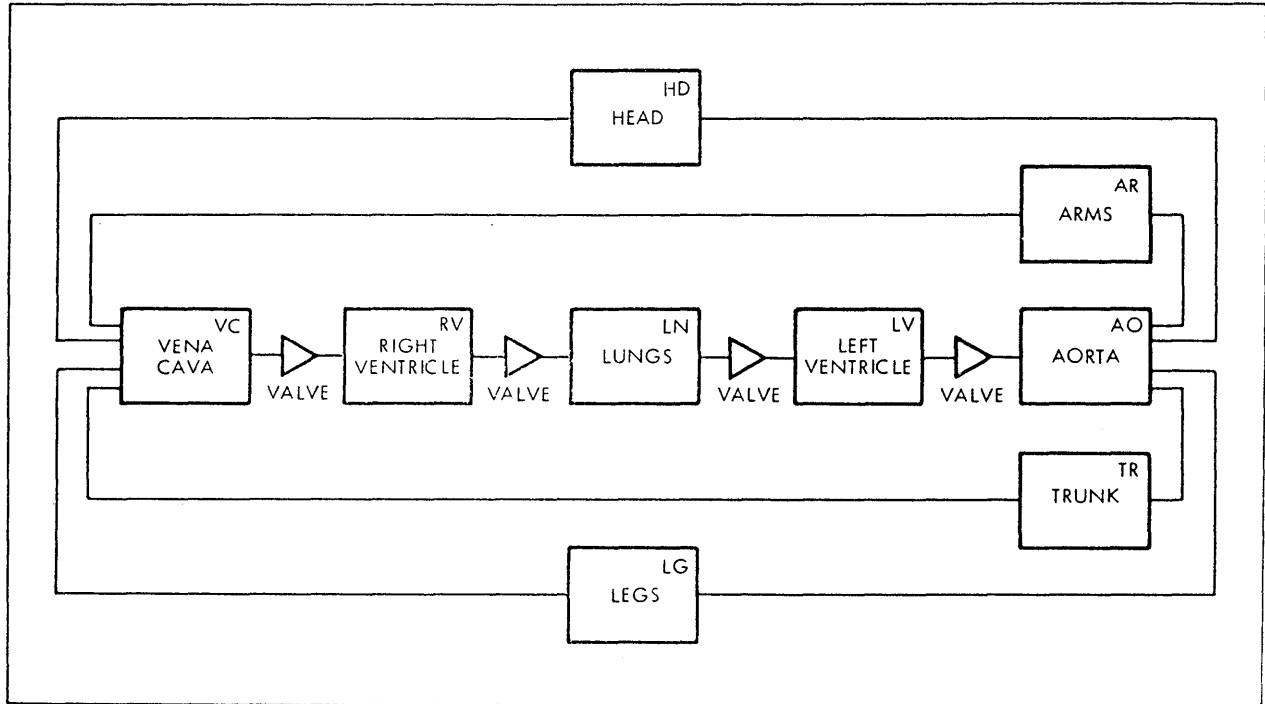


Figure A8-1. Interconnection of Lumps to Form Blood Distribution System

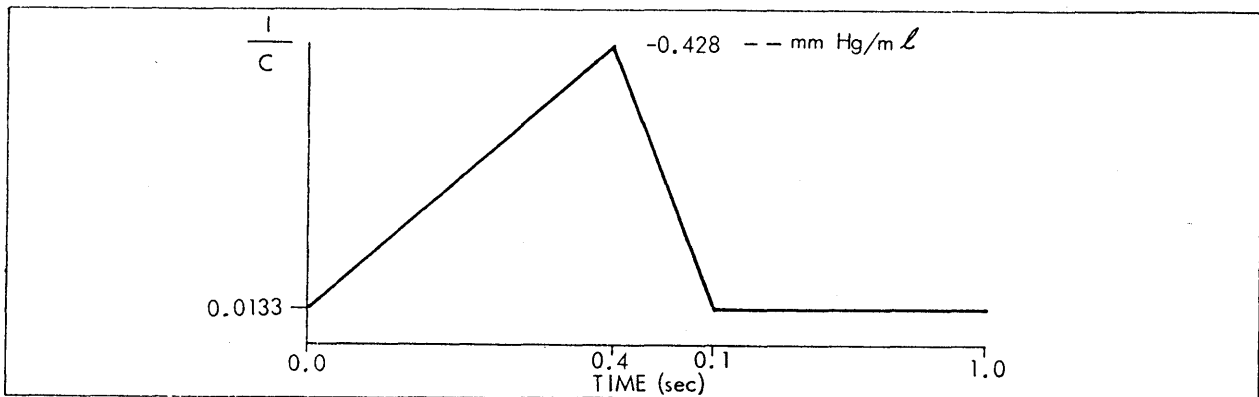


Figure A8-2. Reciprocal Compliance or Spring Constant (Pressure per Unit Volume) for Heart Chambers

Figure A8-3 shows the schematic representation of the mass and heat flow within each lump. Note the XX is to be replaced by the two character mnemonic when referring to one of the nine individual lumps. The input flow rate is given by the pressure differential and input resistance

$$F_{XXI} = \frac{P_{XXI} - P_{XX}}{R_{XXI}}$$

Pressure in the bag is volume by compliance

$$P_{XX} = \frac{V_{XX}}{C_{XX}}$$

Outlet flow rate is given by the pressure differential and output resistance

$$F_{XXO} = \frac{P_{XX} - P_{XXO}}{R_{XXO}}$$

The blood volume is the integrated mass flow rate

$$V_{XX} = \int_0^T (F_{XXI} - F_{XXO}) dt + V_{XXZ}$$

Heat (enthalpy) flow in is

$$Q_{XXI} = F_{XXI} * T_{XXI}$$

Temperature is total enthalpy in lump by mass.

$$T_{XX} = H_{XX}/W_{XX}$$

Heat flow out is

$$Q_{XXO} = F_{XXO} * T_{XX}$$

Heat dissipated to surroundings

$$Q_{XXB} = K * A_{XXO} * (T_{XX} - T_A)$$

and an accumulated heat is

$$H_{XX} = \int_0^T (Q_{XXI} - Q_{XXO} + Q_{XXE}) dt + H_{XXZ}$$

Now the macro is implemented by defining the macro name and identifying the substitutable parameter, i.e.,

MACRO LUMP (X)

identifies X as a substitutable parameter.

To make up the input flow equation, the following statement is used (→ implies concatenation)

$$F \rightarrow X \rightarrow O = (P \rightarrow X \rightarrow I - P \rightarrow X) / R \rightarrow X \rightarrow I$$

which means 'F' concatenated with the argument, concatenated with an 'O', etc. When the macro is invoked to describe the trunk, designated by TR

LUMP ('TR')

where the quotes imply the literal string TR which is not to be considered as a symbol.

The expansion of the macro for this invocation is shown in Figure A8-4.

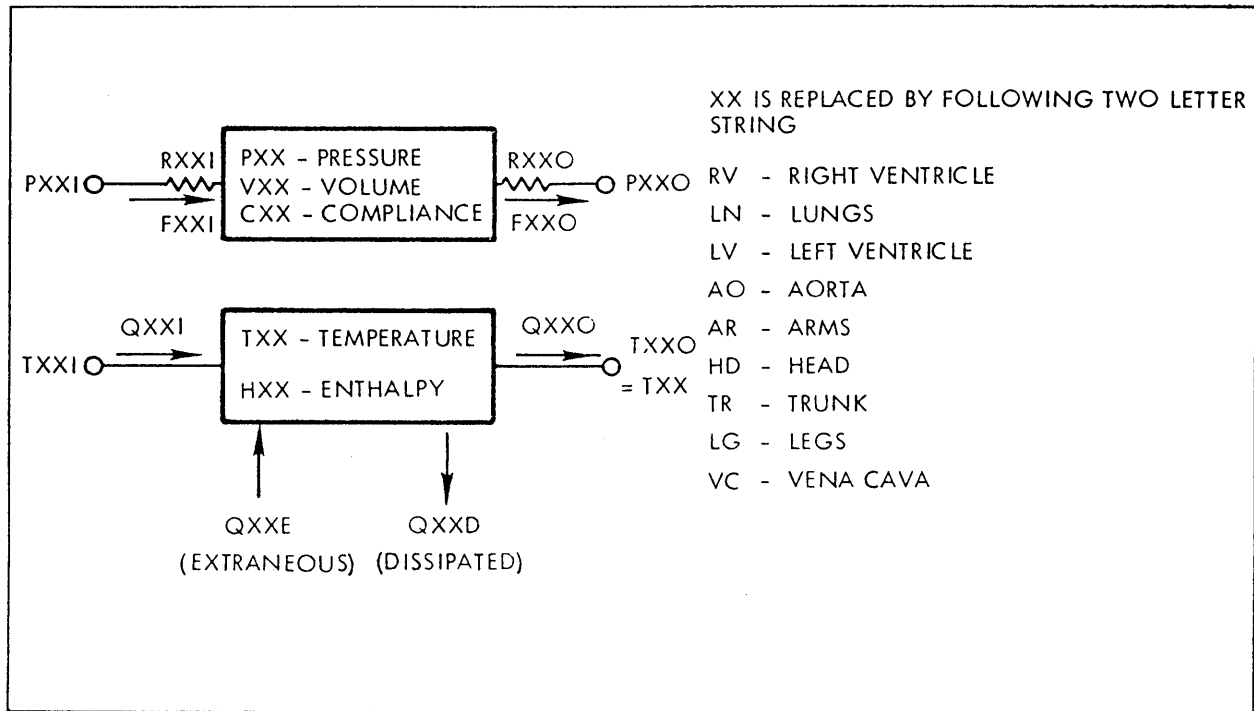


Figure A8-3. Lump Definition - Mass and Enthalpy Balance

```

LUMP('HD')    $ 'HEAD'
FHDI    =(PHDI - PHD)/RHDI
PHD     = VHD/CHD
FHDO    =(PHD - PHDO)/RHDO
VHD     = INTEG(FHDI - FHDO, VHDZ)
QHDI    = FHDI*THDI
THD     = HHD/WHD
QHDO    = FHDO*THD
QHDD    = K*AHD*(THD - TA)
HHD     = INTEG(QHDI - QHDO + QHDE - QHDD, HHDZ)
MACROEND

```

Figure A8-4. Main Invocation and Macro Expansion for Lump Associated with Head (HD)

The other macro used in the simulation is the valve so

MACRO VALVE (R, PO, PI, RZ)

which calculates a resistance R given outlet pressure, PO, inlet pressure PI, and valve open resistance, RZ. If the valve is closed the resistance will rise to a large value (10^{20}) to effectively shut off flow.

This VALVE macro is defined as a PROCEDURAL block, or one whose order must not be changed. The list states that R, the resistance, is the output variable, and that this is calculated as a function of outlet pressure, inlet pressure and open resistance.

Action of the valve is defined by

R = RZ;	PO ≤ PI
R = 10^{20}	PO > PI

Invocation to calculate the resistance into the right ventricle is

VALVE (RRVI = PRV, PVC, RRVIC)

which expands as shown in Figure A8-5. This says that the input resistance to the right ventricle is RRVIZ, if the pressure in the right ventricle PRV is less than the pressure in the vena cava, PVC. Otherwise, it is 10^{20} so preventing any backflow.

```
        VALVE(RRVI = PRV, PVC, RRVIZ)
PROCEDURAL(RRVI = PRV, PVC, RRVIZ)
RRVI      = RRVIZ
IF( PRV.GT. PVC) RRVI = 1.0E20
END
MACROEND
        VALVE(RRVO = PLN, PRV, RRVOZ)
PROCEDURAL(RRVO = PLN, PRV, RRVOZ)
RRVO      = RRVOZ
IF( PLN.GT. PRV) RRVO = 1.0E20
END
MACROEND
```

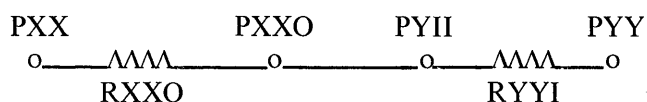
Figure A8-5. Macro and Macro Expansion for Inlet and Outlet Values in Right Ventricle

The program listing Figures A8-6 through A8-7 defines all the constants necessary to specify the system behavior. Since none of the heat flux constants were given in the problem definition they are set to zero, so the temperature equations are carried along to count in the time scale but don't calculate significant quantities. Since there is no feedback, the temperature merely goes along with the mass balance equation for the ride.

The compliance of the ventricles is the drive for the system that causes the heart to pump. The reciprocal compliance is given as a function of time that repeats every second and this is shown in Figure A8-2. Since the code order for calculating the compliance could have been important, it was bracketed in a PROCEDURAL block.

Then, the valves in the right and left ventricles (two in each lump) are defined.

The interconnections of each lump, Figure A8-1, define what the inlet and outlet pressures are. For the simple case of two elements in cascade, the outlet pressure is the effect of two resistances connected between two pressure sources.



$$PXXO = \frac{PXX * RYYI + PYY * RXXO}{RYYI + RXXO}$$

$$PYYI = PXXO$$

The aorta and vena cave are rather more difficult since the first feeds five other lumps and the second receives flow from the five. Writing down the flow balance - equivalent to Kirchoff's law - leads to the following for the aorta output pressure

$$PAOO = \frac{\frac{PAO}{RAOO} + \frac{PHD}{RHDI} + \frac{PAR}{RARI} + \frac{PLG}{RLGI} + \frac{PTR}{RTRI}}{\frac{1}{RAOO} + \frac{1}{RHDI} + \frac{1}{RARI} + \frac{1}{RLGI} + \frac{1}{RTRI}}$$

and similarly for the vena cava inlet pressure

$$PVC I = \frac{\frac{PVC}{RCVI} + \frac{PHD}{RHDO} + \frac{PAR}{RARO} + \frac{PLG}{RLGO} + \frac{PTR}{RTRO}}{\frac{1}{RCVI} + \frac{1}{RHDO} + \frac{1}{RARO} + \frac{1}{RLGO} + \frac{1}{RTRO}}$$

The nine lumps are defined by invoking the macro with the appropriate two character string and this ends the model definition included in a DERIVATIVE section. In the DYNAMIC section - interrogated every communication interval - is the termination condition and for four cycles the stop time TSTP is set at four. This completes the model description: the listing as shown took 13.2 seconds to translate on a Control Data CYBER 173. The run-time commands used to exercise this model are shown listed in Figure A8-8 and the output stream generated in Figures A8-9 through A8-12. Prior to the START, the communication interval (CINT) is set to 20 msec (0.020 sec), the NSTP divisor to one and the calculation interval or integration step specified via the maximum step size (MAXT) to be 10 msec (0.010 sec). After the START the OUTPUT list is printed out every half second, since the "NCIOUT" multiplier is 25.

PROGRAM PHYSBE

INITIAL

MACRO LUMP(X)

F_X_I = (P_X_I - P_X)/R_X_I
 P_X = V_X/C_X
 F_X_O = (P_X - P_X_O)/R_X_O
 V_X = INTEG(F_X_I - F_X_O, V_X_Z)
 Q_X_I = F_X_I*T_X_I
 T_X = H_X/W_X
 Q_X_O = F_X_O*T_X
 Q_X_D = K*A_X*(T_X - TA)
 H_X = INTEG(Q_X_I - Q_X_O + Q_X_E - Q_X_D, H_X_Z)

MACRO END

MACRO VALVE(R, PO, PI, RZ)

PROCEDURAL(R = PO, PI, RZ)

R = RZ
 IF(PO.GT.PI) R = 1.0E20

END

MACRO END

CONSTANT RAOI = 1.E-2 , RAOO = 1.E-2 , CAO = 1.01 , VAOZ= 80.8
 CONSTANT RARI = 5.15 , RARO = 10.0 , CAR = 4.25 , VARZ= 268.0
 CONSTANT RHDI = 2.58 , RHDO = 5.0 , CHD = 1.21 , VHDZ= 68.0
 CONSTANT RTRI = 0.67 , RTRD = 1.42 , CTR = 34.0 , VTRZ= 2180.
 CONSTANT RLGI = 2.58 , RLGO = 5.00 , CLG = 11.1 , VLGZ= 700.0
 CONSTANT RVCI = 1.E-2 , RVCO = 1.E-2 , CVC = 250.0 , VVCZ= 650.0
 CONSTANT RLNI = 1.E-2 , RLND = 0.1875, CLN = 10.0 , VLNZ= 200.0
 CONSTANT RRVIZ= 0.0030, RRVOZ= 0.0030
 CONSTANT RLVIZ= 0.0275, RLVOZ= 0.0060
 CONSTANT VLVZ = 319.0 , VRVZ = 120.0
 CONSTANT AAO = 0.0 , WAO = 1.0 , QAOE= 0.0 , HAOZ= 0.0
 CONSTANT AAR = 3670.0, WAR = 7000.0, QARE = 0.0 , HARZ= 0.0
 CONSTANT AHD = 1400.0, WHD = 4500.0, QHDE = 0.0 , HHZ= 0.0
 CONSTANT ATR = 6000.0, WTR = 53000., QTRE = 0.0 , HTRZ= 0.0
 CONSTANT ALG = 7000.0, WLG = 18500., QLGE = 0.0 , HLGZ= 0.0
 CONSTANT AVC = 0.0 , WVC = 1.0 , QVCE = 0.0 , HVCZ= 0.0
 CONSTANT ARV = 0.0 , WRV = 600.0 , QRVE = 0.0 , HRVZ= 0.0
 CONSTANT ALN = 50000., WLN = 1000.0, QLNE = 0.0 , HLNZ= 0.0
 CONSTANT ALV = 0.0 , WLV = 600.0 , QLVE = 0.0 , HLVZ= 0.0
 CONSTANT QARD = 0.0 , QHDD = 0.0 , QTRD = 0.0 , QLGD = 0.0
 CONSTANT QVCD = 0.0 , QRVD = 0.0 , QLND = 0.0 , QLVD = 0.0
 CONSTANT QAOD = 0.0
 CONSTANT K = 0.01 , TA = 0.0 , TSTP = 3.99
 CONSTANT TMX = 0.4 , TMN = 0.5,
 CIMX = 0.428 , CIMN = 0.0133

END \$* OF INITIAL *

DYNAMIC

DERIVATIVE

PROCEDURAL(CRV, CLV=T)

TF = AMOD(T, 1.0)
 IF(TF.LE.TMX) CI = (CIMX-CIMN)*TF/TMX + CIMN
 IF(TF.GT.TMX .AND. TF.LE.TMN) CI = (CIMX-CIMN)*(TMN-TF)...
 /(TMN-TMX) + CIMN
 IF(TF.GT.TMN) CI = CIMN
 CRV = 1.0/CI \$ CLV = CRV

END

Figure A8-6. Listing of Physiological Simulation Benchmark Experiment (PHYSBE) Model Definition

```

"VALVES IN RIGHT AND LEFT VENTRICLES"
VALVE(RRVI = PRV, PVC, RRVIZ)
VALVE(RRVO = PLN, PRV, RRVOZ)
VALVE(RLVI = PLV, PLN, RLVI Z)
VALVE(RLVO = PAO, PLV, RLVOZ)
"DEFINE THE INTERCONNECTIONS"
"VENA CAVA FEEDS RIGHT VENTRICLE"
PVCO   =(PVC*RRVI + PRV*RVCO)/(RRVI + RVCO)
PRVI   = PVCO           $"RIGHT VENTRICLE"
TRVI   = TVC
"RIGHT VENTRICLE FEEDS LUNGS"
PRVO   =(PRV*RLNI + PLN*RRVO)/(RLNI + RRVO)
PLNI   = PRVO           $"LUNG INPUTS"
TLNI   = TRV
"LUNGS FEED LEFT VENTRICLE"
PLNO   =(PLN*RLVI + PLV*RLNO)/(RLVI + RLNO)
PLVI   = PLNO           $"LEFT VENTRICLE INPUTS"
TLVI   = TLN
"LEFT VENTRICLE FEEDS AORTA"
PLVO   =(PLV*RAOI + PAO*RLVO)/(RAOI + RLVO)
PAOI   = PLVO           $"AORTA INPUTS"
TAOI   = TLV
"AORTA FEEDS HEAD, ARMS, LEGS AND TRUNK"
PAOO   =(PAO/RAOO + PHD/RHDI + PAR/RARI + PLG/RLGI + PTR/RTRI)...
        /(1.0/RAOO + 1.0/RHDI + 1.0/RARI + 1.0/RLGI + 1.0/RTRI)
PHDI   = PAOO           $"HEAD INPUTS"
THDI   = TAO
PARI   = PAOO           $"ARMS INPUTS"
TARI   = TAO
PLGI   = PAOO           $"LEG INPUTS"
TLGI   = TAO
PTRI   = PAOO           $"TRUNK INPUTS"
TTRI   = TAO
"VENA CAVA IS RETURN FROM HEAD, ARMS, LEGS AND TRUNK"
PVCI   =(PVC/RVCI + PHD/RHDI + PAR/RARO + PLG/RLGI + PTR/RTRO)...
        /(1.0/RVCI + 1.0/RHDI + 1.0/RARO + 1.0/RLGI + 1.0/RTRO)
TVCI   =(THD*FHDI + TAR*FARI + TLG*FLGI + TTR*FTRO)/FVCI
PARO   = PVCI $ PHDO = PVCI $ PLGO = PVCI $ PTRO = PVCI
"DEFINE EACH LUMP"
LUMP("RV")  $"RIGHT VENTRICLE"
LUMP("LN")  $"LUNGS"
LUMP("LV")  $"LEFT VENTRICLE"
LUMP("AO")  $"AORTA"
LUMP("AR")  $"ARMS"
LUMP("HD")  $"HEAD"
LUMP("TR")  $"TRUNK"
LUMP("LG")  $"LEGS"
LUMP("VC")  $"VENA CAVA"
END $" OF DERIVATIVE "
      TERMT(T. GE. TSTP)
END $" OF DYNAMIC "
END $" OF PROGRAM "

```

13.217 CP SECONDS 3649 TABLE SPACE USED 19. TABLE MOVES

Figure A8-7. Listing of PHYSBE Model Definition

```

SET TITLE="PHYSIOLOGICAL SIMULATION BENCHMARK EXPERIMENT (PHYSBE)"
S TCWPRN=72,DIS=9 $" FORCE 3 COLUMN OUTPUT WIDTH "
PREPAR T, PRV, PLN, PLV, PAO, PHD, PTR, PAR, PLG, PVC, FVCI
, FAOO, FARO, FHDO, FLGO, FLNO, FLVO, FRVO, FTRO, FVCO, CRV, CLV, CI
OUTPUT T, PRV, PLN, PAO, PHD, "NCIOUT"=25
SET CINT=0.02, NSTP=1, MAXT=0.010
START
PRINT "NCIPRN"=4, T, PRV, PLN, FLVO, FRVO
S STRPLT=.T., PRNPLT=.F., CALPLT=.F., GRDSPL=.T.
PLOT FRVO, PLN, PRV
S CALPLT=.T., GRDCPL=.T., PRNPLT=.T., STRPLT=.F.
PLOT PRV, PLN
STOP

```

Figure A8-8. Run-time Drive Commands to Exercise PHYSBE Model

After the simulation run a more detailed print out is obtained for two pressures, PRV and PLN, and two flows, FLV and FRV, every four communication intervals or every 80 msec. This runs through Figure A8-10. Next we obtain a strip chart plot by setting STRPLT true. The right ventricle flow and lung and right ventricle pressures are shown plotted in Figure A8-11. The grid was produced since GRDSPL was made true prior to the plot command. The next line (Figure A8-10) turns on normal line plots (CALPLT=.T.), asks for grid on these plots (GRDCPL=.T.), turns on print plots (PRNPLT=.T.) and turns off the strip plot (STRPLT=.F.). The following PLOT command produces the printer plot of Figure A8-12 and the line plot, actually on a Gould electrostatic plotter, of Figure A8-13.

SET TITLE="PHYSIOLOGICAL SIMULATION BENCHMARK EXPERIMENT (PHYSBE)"
 S TCWPRN=72,DIS=9 \$* FORCE 3 COLUMN OUTPUT WIDTH *
 PREPAR T, PRV, PLN, PLV, PAO, PHD, PTR, PAR, PLG, PVC, FVCI ...
 , FAOD, FARO, FHDO, FLGO, FLND, FLVO, FRVO, FTRO, FVCO, CRV, CLV, CI
 OUTPUT T, PRV, PLN, PAO, PHD, *NCIOUT*=25
 SET CINT=0.02,NSTP=1,MAXT=0.010
 START

T 0.	PRV 1.59600000	PLN 20.0000000
PAO 80.0000000	PHD 56.1983471	
T 0.50000000	PRV 0.85086878	PLN 25.4171240
PAO 100.880641	PHD 56.9120419	
T 1.00000000	PRV 1.58567986	PLN 20.9539454
PAO 74.2731870	PHD 56.8334996	
T 1.50000000	PRV 0.87358286	PLN 26.0932275
PAO 99.1535674	PHD 57.0084605	
T 2.00000000	PRV 1.62192528	PLN 21.4780036
PAO 73.5418443	PHD 56.7579594	
T 2.50000000	PRV 0.89433126	PLN 26.7111711
PAO 98.2354651	PHD 56.8377607	
T 3.00000000	PRV 1.65475157	PLN 21.9627005
PAO 73.0300194	PHD 56.5338005	
T 3.50000000	PRV 0.91342323	PLN 27.2808804
PAO 97.8142421	PHD 56.6051521	
T 4.00000000	PRV 1.68449157	PLN 22.4132055
PAO 72.6692007	PHD 56.2998931	

PRINT *NCIPRN*=4, T, PRV, PLN, FLVO, FRVO

LINE	T	PRV	PLN	FLVO	FRVO
0	0.	1.5960000	20.000000	-7.576E-19	-1.840E-19
4	0.0800000	11.579815	19.828992	-4.612E-19	-8.249E-20
8	0.1600000	21.410190	19.912234	-1.701E-19	115.22742
12	0.2400000	25.903454	21.978930	267.24770	301.88644
16	0.3200000	26.869540	24.074300	287.50230	215.01840
20	0.4000000	27.324480	25.476994	279.16393	142.11424
24	0.4800000	6.1052156	25.517478	-7.689E-19	-1.941E-19
28	0.5600000	0.9627592	24.817419	-9.213E-19	-2.385E-19
32	0.6400000	1.1002612	24.046745	-8.629E-19	-2.295E-19
36	0.7200000	1.2256486	23.307887	-8.146E-19	-2.208E-19
40	0.8000000	1.3401452	22.599530	-7.744E-19	-2.126E-19
44	0.8800000	1.4448501	21.920415	-7.410E-19	-2.048E-19
48	0.9600000	1.5407505	21.269336	-7.131E-19	-1.973E-19
52	1.0400000	6.5501247	20.766080	-5.589E-19	-1.422E-19
56	1.1200000	16.469192	20.757570	-2.778E-19	-4.288E-20
60	1.2000000	24.830554	21.463533	1.2038419	259.00168
64	1.2800000	27.213663	23.752947	281.51563	266.20886
68	1.3600000	27.831076	25.516663	276.73555	178.03178

Figure A8-9. Output Stream from PHYSBE Model

LINE	T	PRV	PLN	FLV0	FRV0
72	1.4400000	17.074435	26.202923	-3.574E-19	-9.128E-20
76	1.5200000	0.9124997	25.884333	-9.390E-19	-2.497E-19
80	1.6000000	1.0593256	25.070429	-8.772E-19	-2.401E-19
84	1.6800000	1.1930849	24.290124	-8.260E-19	-2.310E-19
88	1.7600000	1.3150978	23.542032	-7.835E-19	-2.223E-19
92	1.8400000	1.4265501	22.824822	-7.482E-19	-2.140E-19
96	1.9200000	1.5285069	22.137220	-7.187E-19	-2.061E-19
100	2.0000000	1.6219253	21.478004	-6.941E-19	-1.986E-19
104	2.0800000	11.772357	21.265876	-4.146E-19	-9.494E-20
108	2.1600000	21.901368	21.275058	-1.385E-19	48.177692
112	2.2400000	27.110493	23.155388	265.84205	304.23883
116	2.3200000	28.236425	25.315120	276.68971	224.71582
120	2.4000000	28.727497	26.786136	268.98179	149.33546
124	2.4800000	6.4187495	26.827306	-7.490E-19	-2.041E-19
128	2.5600000	1.0103165	26.073132	-8.985E-19	-2.506E-19
132	2.6400000	1.1527750	25.253196	-8.432E-19	-2.410E-19
136	2.7200000	1.2825971	24.467109	-7.972E-19	-2.318E-19
140	2.8000000	1.4010590	23.713473	-7.591E-19	-2.231E-19
144	2.8800000	1.5093069	22.990948	-7.273E-19	-2.148E-19
148	2.9600000	1.6083708	22.298251	-7.007E-19	-2.069E-19
152	3.0400000	6.8350133	21.753322	-5.495E-19	-1.492E-19
156	3.1200000	17.185497	21.737983	-2.730E-19	-4.552E-20
160	3.2000000	25.947598	22.457848	2.6812666	268.44232
164	3.2800000	28.458590	24.843446	277.11008	278.08799
168	3.3600000	29.107288	26.687059	272.46705	186.17147
172	3.4400000	17.857531	27.404736	-3.527E-19	-9.547E-20
176	3.5200000	0.9535880	27.060560	-9.264E-19	-2.611E-19
180	3.6000000	1.1050721	26.202135	-8.656E-19	-2.510E-19
184	3.6800000	1.2429967	25.379148	-8.153E-19	-2.414E-19
188	3.7600000	1.3687316	24.590135	-7.735E-19	-2.322E-19
192	3.8400000	1.4835075	23.833694	-7.387E-19	-2.235E-19
196	3.9200000	1.5884298	23.108481	-7.097E-19	-2.152E-19
200	4.0000000	1.6844916	22.413205	-6.855E-19	-2.073E-19

S STRPLT=. T. , PRNFLT=. F. , CALPLT=. F. , GRDSPL=. T.
 PLOT FRV0, PLN, PRV
 S CALPLT=. T. , GRDCPL=. T. , PRNFLT=. T. , STRPLT=. F.
 PLOT PRV, PLN

Figure A8-10. Output Stream from PHYSBE Model

PHYSIOLOGICAL SIMULATION BENCHMARK EXPERIMENT (PHYSBE)

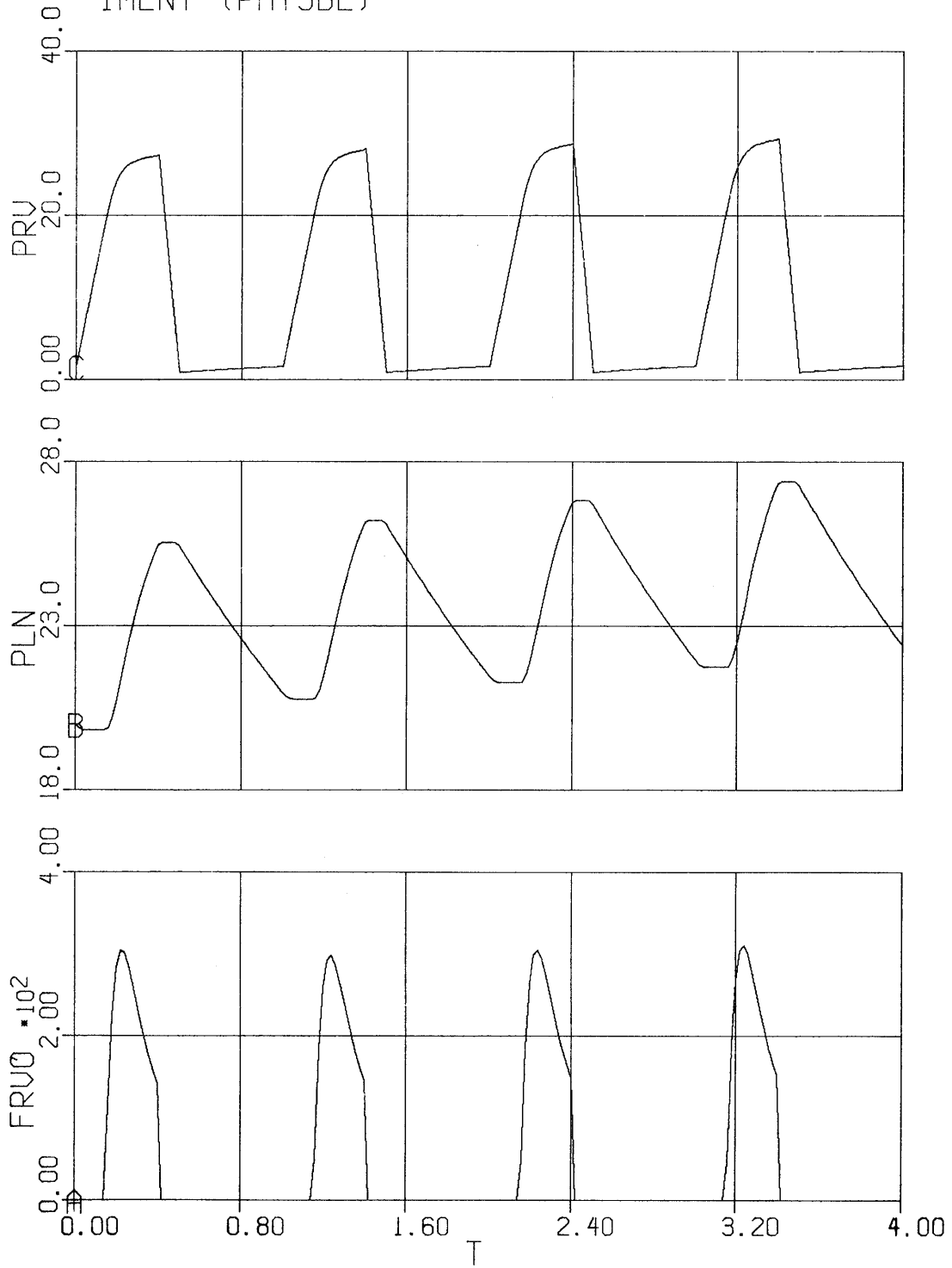
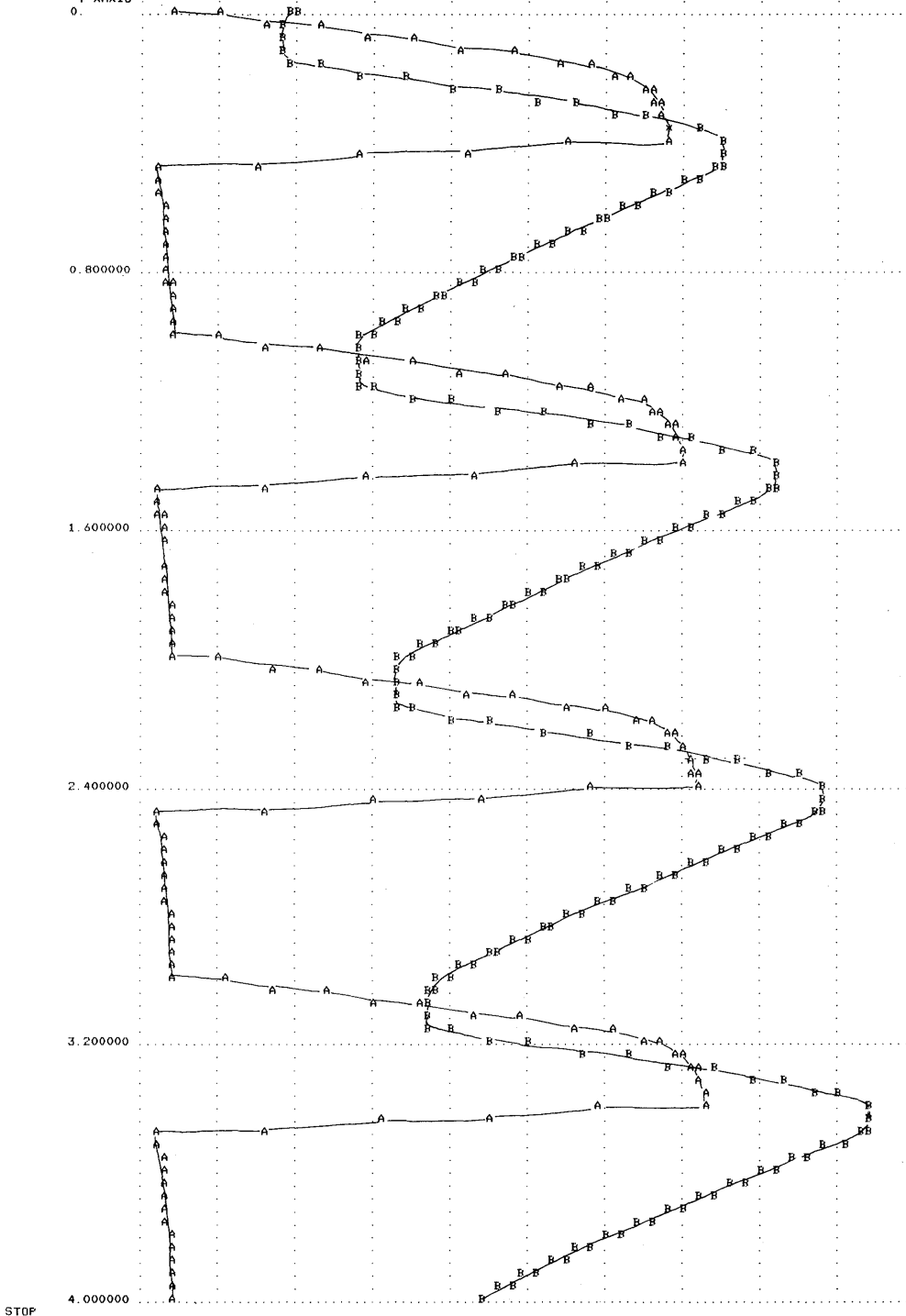


Figure A8-11. Strip Plot of Right Ventricle and Lung Pressure and Right Ventricle Flow

PRV A 0. 4.000000 8.000000 12.000000 16.000000 20.000000 24.000000 28.000000 32.000000 36.000000 40.000000
 PLN B 18.000000 19.000000 20.000000 21.000000 22.000000 23.000000 24.000000 25.000000 26.000000 27.000000 28.000000
 T X AXIS



END DISPLA -- 3039 VECTORS GENERATED IN 2 PLOT FRAMES.
 2353 WORDS TABLE SPACE USED

Figure A8-12. Plot of Right Ventricle and Lung Pressures from PHYSBE Model

PHYSIOLOGICAL SIMULATION BENCHMARK EXPERIMENT (PHYSBE)

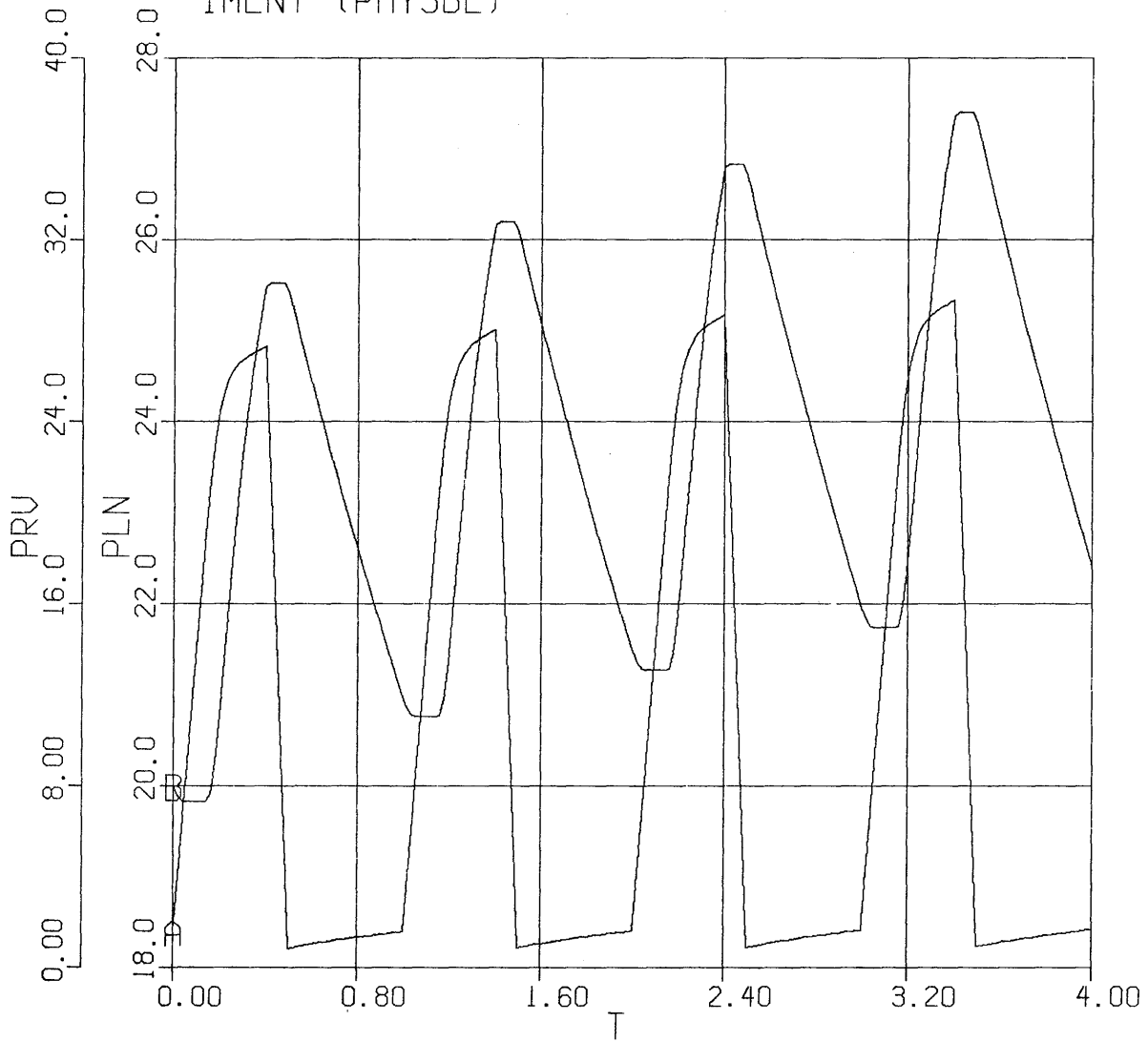


Figure A8-13. Line Plot of Right Ventricle and Lung Pressures from PHYSBE Model

9. PHASE AND GAIN

The accompanying program was put together to calculate and plot the phase and gain through the plant described by the following transfer function

$$\frac{X}{Y} = G(s) = \frac{0.5s + 1}{0.03s^2 + 0.1s + 1}$$

The characteristics were to be obtained by forcing the model with a sine wave and determining the in-phase and quadrature components, from which gain and phase can be calculated. Changing frequency slowly allows successive points to be calculated until, at the end of a single run, the complete Bode plot can be generated. This method is a relatively expensive way to obtain frequency response and is almost never justified for linear systems. For a non-linear plant though, this method is the best one available and can be used to match bench tests on hardware with a sinewave generator. Care must be taken to choose the excitation amplitude correctly and also make sure the model is in steady state while the measurements are made.

The program listing is given in Figure A9-1 and A9-2 and should be followed through the following discussion. The basic action is to establish a separate DERIVATIVE block, which will repeat every cycle of the current frequency, and which examines the inphase and quadrature components of a full cycle integration. When the change in phase from cycle to cycle is small enough, the data point is recorded and the frequency changed so that another point can be calculated. In the program, the INITIAL section sets the first frequency, W, and phase, FI. The frequency is started at the maximum value and will be reduced geometrically by:

$$W = \text{AMAX1}(\text{WMN}, \text{KW} * W)$$

so that the final logarithmic plot will have equally spaced points. For this application KW was defined to be 0.8 and the frequency sweep was from a maximum value (WMX) of 100.0 down to a minimum value (WMN) of 1.0. Next go to the second DERIVATIVE section listed in Figure A9-2 where the plant is defined - this is identified as DERIVATIVE CONTIN. First the name for the step size is defined (MAXTC) and set to zero to indicate that it is calculated elsewhere in the model. The frequency bounds, WMN and WMX, are specified together with the amplitude of the forcing function (XMAG) and the settling time (TSETTL=2.0). Since the plant is linear the value of the XMAG has no effect but for a real non-linear plant this value should be chosen to match the plant capability. The actual forcing function X is obtained from:

$$X = \text{XMAG} * \text{SIN}(W * T + \text{FI})$$

where FI is a parameter used to ensure continuity of X, when the frequency W is changed. The model is defined by the transfer operator, TRAN, and numerator and denominator polynomials. The numerator coefficients are 0.5 and 1.0; the denominator coefficients 0.03, 0.1 and 1.0. In addition we specify a maximum step size (MAXTZ) of 0.050 sec chosen conservatively from the plant roots of $(-1.5 \pm j6)$ and a minimum divisor (NSTPMN) of the period. This really says that no matter how fast the drive sine wave is, use a calculation interval of at least a tenth (NSTPMN=10) of the period. As the frequency get slower and slower, don't allow the step size to rise above MAXTZ (=0.050). This is performed by the line in the DISCRETE section above which calculates MAXTC or:

$$\text{MAXTC} = \text{AMIN1}(\text{PERIOD} / \text{NSTPMN}, \text{MAXTZ})$$

The output (Y) of the plant is obtained using the TRAN operator which can be used to represent a general polynomial transfer function i.e.

$$Y = \text{TRAN}(1, 2, A, B, X)$$

and then the in-phase (P) and quadrature (Q) components are obtained by multiplying the output (Y) by $\text{Sin}(W * T + \text{FI})$ and $\text{Cos}(W * T + \text{FI})$ respectively and integrating.

Returning to the DISCRETE code of Figure A9-1, the first calculation is to obtain the change in the in-phase and quadrature components since the last cycle i.e.

$$DLP = P - PP$$

$$DLQ = Q - QP$$

Since PP and QP are initialized to large numbers, the first time the difference will be large.

The termination condition (TERMT) is inserted in case a frequency is chosen such that the attenuation is too great for the machine precision. For 32 bit computers, EPM should probably be set to 0.0001 which will still give room for nearly 80 dB attenuation. If the change from cycle to cycle in P and Q is less than the machine precision, the increments will be zero and the logarithm will fail further on in calculating the gain (GDBN). Once the increment has been calculated, the current values of P and Q are saved to become the previous values (PP and QP) next time. The phase and gain over the last cycle are calculated from

$$PHASE = ATAN[\Delta Q/\Delta P]$$

$$GAIN = 10.0*LOG10[(\Delta P^2 + \Delta Q^2) (W/\pi)^2]$$

If the new phase differs from the last phase by too much (nominally 0.1 degrees) then the data save operation is skipped. Similarly if a settling time (TSETTL) has not passed, another cycle is taken. If both these tests are passed then the phase, gain and frequency are transferred to separately named variables PDG, GDB and WFR which will be used for plotting. Next the frequency W is decreased geometrically and the phase (FI) of the forcing function adjusted to give continuity in the output. This helps ensure a shorter settling time between frequency changes. The previous phase is set to a large number so that the integration over the first cycle will be rejected and lastly the data logging routine, LOG, is called in order to force an output and data recording action at this point. Recall that the communication interval CINT was set to 1000.0 in the beginning of the model definition with the intent that this will produce no output beyond the first time-equal-zero point. All the other output will be obtained every time LOG is called indicating a new frequency point is being recorded.

For all cases we then make the previous phase (PDGP) equal to the new phase (PDGN), ready for next time, recalculate the PERIOD and a new step size for the continuous section. These last two calculations only need to be done when the frequency changes but are put here in order to handle the initialization problem rather than repeat the code in the INITIAL section.

The output stream that results in exercising the model is shown in Figures A9-3. Note the frequency multiplier KW was changed to 0.9 from the nominal 0.8 in order to increase the point density to 23 per decade so producing smoother plots. The plot was modified to appear on both normal line plot (CALPLT=.T.) as well as the default printer plot using a logarithmic x-axis scale. With this logarithmic scale a minimum must be specified to avoid the rounding to zero which is expressed symbolically as the contents of WMN (=1.0 rad/sec) i.e.

PLOT 'XLOG, 'XLO' = WMN, PDG, GDB

The resulting pictures are shown in Figures A9-4 and A9-5.

These plots were made in one simulation run stopping every so often to change the drive frequency to a new value. Examination of the last value of time showed that it needed 242 seconds of simulated time to complete the sweep, which explains why it is usually considered an expensive picture to obtain. Individual transient studies will be over in three or four seconds or so whereas this run needs effectively sixty transient runs.

TABLE A9-1. Comparison of Measured and Theoretical Gain (dB) and Phase (degree)

W	GDBM	GDBT	PDGM	PDGT
90.00	-14.612	-14.617	-88.64	-89.14
31.38	-5.230	-5.232	-87.47	-87.37
10.94	5.916	+5.920	-77.47	-77.47
3.090	7.484	+7.484	+33.67	+33.67
1.000	1.188	+1.188	+20.68	+20.68

From an independent run with a listing of time, T, along with the frequency, phase and gain, it can be shown that the high frequency points can be obtained quickly. In fact it takes 35 seconds to sweep the frequency from 100 down to 10 radian/second and a further 186 seconds to go from 10 down to 1 radian/second. The reason for this is that we must wait at least one complete cycle and at 1 radian/second each measurement is taking over six seconds.

One caveat is in order in running this type of program and that is errors caused by non-steady-state operating conditions. At low attenuations, the drive signal dominates in the P and Q integrators but when the plant attenuation becomes high, the residual motion excited by the start up transient or frequency switch can become important. An idea of the magnitude of the effect can be obtained by looking at the irregularity in the phase plot when the frequency is over 20 radians/second. The plot should theoretically be smooth and monotonically negative as the phase asymptotically approaches -90 degrees. In the output stream of Figure A9-3, the phase at 72.9 radians/second is given as -87.8 degrees but at the next frequency point of 65.6 it has gone back to -89.3. This change of 1.5 degrees in the wrong direction must be an artifact of the measurement implementation.

We have done an empirical study of this phenomenon and found that better results are obtained with heavier damped systems (shorter settling times) and certainly when the attenuation is relatively small. If problems occur, experimentation with the phase change error parameter (EPDG) can help or else add a fixed settling time after each frequency change so that the system has time to get to a steady state before the measurement is taken. As a test of the accuracy, we evaluated the theoretical gain and phase at five selected frequency points and the comparison is listed in Table A9-1. Note the phase error of half a degree at high frequencies. A second run is made with the allowable phase error EPDG changed to 0.01 (from 0.1). This forced slightly longer settling times so that the overall sweep time was 269 seconds (up from 242 seconds) and the measured phase at a frequency of 90 radians/second was now -89.07 degrees (theoretical is -89.14 degrees). The increased settling time was seen in that the measurement at $W = 100$ was available at 2.51 sec (was 2.01) and that at $W = 90$ was produced at 3.84 seconds (was 2.29). At lower frequencies, the tighter constraint had little effect since settling has occurred within the first cycle anyway.

PROGRAM PHASE AND GAIN

```
"-----COMPUTE PHASE AND GAIN OF A GIVEN *
" TRANSFER FUNCTION BY INTEGRATING OVER A COMPLETE CYCLE. *
" CONTINUE TO INTEGRATE UNTIL PHASE CHANGE FROM CYCLE TO CYCLE *
" IS LESS THAN SOME PRESET MINIMUM *
```

```
CINTERVAL      CINT = 1000.0
NSTEPS         NSTP = 1

CONSTANT       RMN = 1.0E-30      , RMX = 1.0E30
```

INITIAL

```
"-----SET FIRST FREQUENCY AND PHASE *
W              = WMX
FI             = 0.0
"-----SET PREVIOUS *
PP            = RMX
QP            = RMX
PDGP          = RMX
"-----INITIALISE PLOT VARIABLES *
PDG           = 0.0
GDB           = 0.0
WFR           = 0.0
```

END \$* OF INITIAL *

DERIVATIVE DISCRETE

```
ALGORITHM      IALD = 0
MININTERVAL    PERIOD = 0.0 $* INDICATE PERIOD WILL BE CALCULATED*

CONSTANT       RADDEG = 57.3      , PI = 3.14159
CONSTANT       EPDG = 0.1         , EPM = 1.0E-7
CONSTANT       KW = 0.8           , TSTP = 10000.0
```

PROCEDURAL

```
"-----CHANGE IN IN-PHASE AND QUADRATURE INTEG *
" RALS OVER LAST CYCLE *
DLP            = P - PP
DLQ            = Q - QP
"-----IF RELATIVE CHANGE TOO SMALL FOR MACH ACC*
TERMT((DLP**2 + DLQ**2)/(P**2 + Q**2 + RMN) .LT. EPM**2)
"-----SAVE NEW INTEGRALS AS PREVIOUS *
PP            = P
QP            = Q
"-----CALCULATE NEW PHASE AND GAIN *
PDGN          = ATAN2(DLQ, DLP + RMN)*RADDEG
GDBN          = 10.0*ALOG10((DLP**2 + DLQ**2)*(W/(PI*XMAG))**2)
"-----IF CHANGE IN PHASE NOT SMALL ENOUGH YET *
IF(ABS(PDGN - PDGP) .GT. EPDG) GO TO SKIP1
"-----IGNORE RESULTS UNTIL AFTER SETTLING TIME *
IF(T .LT. TSETTL) GO TO SKIP1
"-----TERMINATE ON FREQUENCY SWEEP *
```

Figure A9-1. Model Definition Listing for Phase and Gain Study

```

TERMT(W .LE. WMN)
"-----SAVE VALUE IN SEPARATE NAME FOR PLOTTING "
PDG      = PDGN
GDB      = GDBN
WFR      = W
"-----ADVANCE FREQUENCY GEOMETRICALLY "
W        = AMAX1(WMN, KW*W)
"-----CALCULATE NEW PHASE FOR CONTINUITY "
"-----OF FORCING FUNCTION AT NEW FREQUENCY "
FI       = FI + T*(WFR - W)
"-----ENSURE PREVIOUS PHASE SET TO FORCE AT "
"-----LEAST TWO CYCLES "
PDGN     = RMX
"-----FORCE A DATA LOGGING ACTION "
CALL LOG
SKIP1..CONTINUE
"-----RESET PREVIOUS PHASE FOR NEXT TIME "
PDGP     = PDGN
"-----RECALCULATE NEW PERIOD AND STEP SIZE "
PERIOD   = 2.0*PI/W
MAXTC    = AMIN1(PERIOD/NSTPMN, MAXTXZ)
END $" OF PROCEDURAL "
TERMT(T .GT. TSTP)

END $" OF DISCRETE "

DERIVATIVE CONTIN

MAXTERVAL      MAXTC = 0.0
CONSTANT       WMN = 1.0           , WMX = 100.0
CONSTANT       XMAG = 1.0         , TSETTL = 2.0

X              = XMAG*SIN(W*T + FI)

"-----DEFINE MODEL "
REAL          A(2), B(3)
CONSTANT      A = 0.5, 1.0         , B = 0.03, 0.1, 1.0
CONSTANT      MAXTXZ = 0.050      , NSTPMN = 10.0
Y            = TRAN(1, 2, A, B, X)
"-----INTEGRATE FOR IN-PHASE AND QUADRATURE COM"
P            = INTEG(Y*SIN(W*T + FI), 0.0)
Q            = INTEG(Y*COS(W*T + FI), 0.0)

END $" OF CONTINUOUS SECTION "

END $" OF PROGRAM "

```

Figure A9-2. Model Definition Listing for Phase and Gain Study

S TITLE="PHASE AND GAIN OF A TRANSFER FUNCTION"
 SET TCWPRN=72 \$" FORCE OUTPUT WIDTH TO FIT ON PAGE "
 OUTPUT WFR, GDB, PDG
 PREPAR WFR, GDB, PDG, T, DLP, DLQ
 SET KW=0.9
 START

WFR 0.	GDB 0.	PDG 0.
WFR 100.000000	GDB-15.5339830	PDG-89.4828190
WFR 90.0000000	GDB-14.6119512	PDG-88.6436703
WFR 81.0000000	GDB-13.6880842	PDG-88.1866592
WFR 72.9000000	GDB-12.7655032	PDG-87.8266695
WFR 65.6100000	GDB-11.8405518	PDG-89.3292461
WFR 59.0490000	GDB-10.9121528	PDG-88.9555216
WFR 53.1441000	GDB-9.97898569	PDG-87.8230158
WFR 47.8296900	GDB-9.04066204	PDG-88.7937635
WFR 43.0467210	GDB-8.09906553	PDG-88.0034209
WFR 38.7420489	GDB-7.15376933	PDG-86.7765993
WFR 34.8678440	GDB-6.19762809	PDG-87.4969789
WFR 31.3810596	GDB-5.23020223	PDG-87.4719048
WFR 28.2429536	GDB-4.25508961	PDG-86.8380793
WFR 25.4186583	GDB-3.25076861	PDG-87.2079492
WFR 22.8767925	GDB-2.25563316	PDG-85.9499742
WFR 20.5891132	GDB-1.21329761	PDG-85.7098543
WFR 18.5302019	GDB-0.15270019	PDG-84.8512870
WFR 16.6771817	GDB 0.98163307	PDG-84.6636226
WFR 15.0094635	GDB 2.08723526	PDG-82.9781036
WFR 13.5085172	GDB 3.29087721	PDG-81.6269574
WFR 12.1576655	GDB 4.56177919	PDG-79.8609486
WFR 10.9418989	GDB 5.91640044	PDG-77.4747302
WFR 9.84770902	GDB 7.39443458	PDG-74.1437063
WFR 8.86293812	GDB 8.95048332	PDG-69.5784310
WFR 7.97664431	GDB 10.6294689	PDG-62.7842192
WFR 7.17897988	GDB 12.3176694	PDG-52.8597177
WFR 6.46108189	GDB 13.7611058	PDG-38.5498530
WFR 5.81497370	GDB 14.4616674	PDG-20.4125257
WFR 5.23347633	GDB 14.0973675	PDG-2.11206446
WFR 4.71012870	GDB 12.9260856	PDG 12.3714693
WFR 4.23911583	GDB 11.4640791	PDG 22.1337918
WFR 3.81520424	GDB 10.0113195	PDG 28.2251367
WFR 3.43368382	GDB 8.67507197	PDG 31.8013975
WFR 3.09031544	GDB 7.48419450	PDG 33.6716777
WFR 2.78128389	GDB 6.43282308	PDG 34.3740970
WFR 2.50315550	GDB 5.50769463	PDG 34.2452681
WFR 2.25283995	GDB 4.69641667	PDG 33.5226479
WFR 2.02755596	GDB 3.98711998	PDG 32.3719701
WFR 1.82480036	GDB 3.36962092	PDG 30.9191840
WFR 1.64232033	GDB 2.83474951	PDG 29.2622764
WFR 1.47808829	GDB 2.37411194	PDG 27.4796699
WFR 1.33027946	GDB 1.97983743	PDG 25.6343745
WFR 1.19725152	GDB 1.64446882	PDG 23.7766047
WFR 1.07752637	GDB 1.36094389	PDG 21.9455378
WFR 1.00000000	GDB 1.18775786	PDG 20.6805527
WFR 1.00000000	GDB 1.18775786	PDG 20.6805527

S CALPLT=.T., GRDCPL=.T.
 PLOT "XLOG", "XLO"=WMN, PDG, GDB

Figure A9-3. Output Stream from Phase and Gain Model

PDG A -100.0000 -80.00000 -60.00000 -40.00000 -20.00000 0. 20.00000 40.00000 60.00000 80.00000 100.0000
 GDB B -20.00000 -16.00000 -12.00000 -8.000000 -4.000000 0. 4.000000 8.000000 12.00000 16.00000 20.00000
 WFR XAXIS
 10**(0)

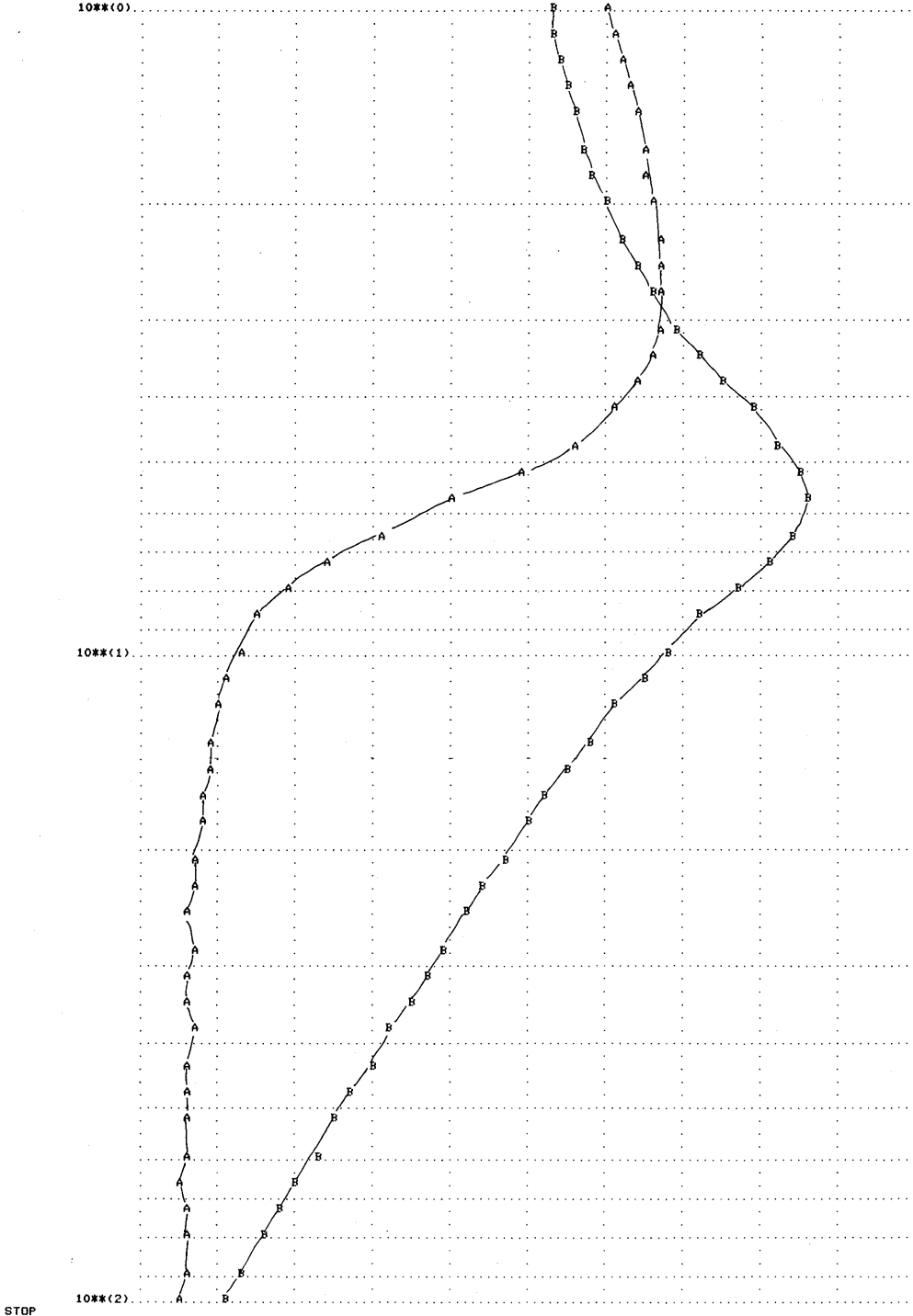


Figure A9-4. Plot of Phase and Gain against Frequency in Radians/second

PHASE AND GAIN OF A TRANSFER FUNCTION

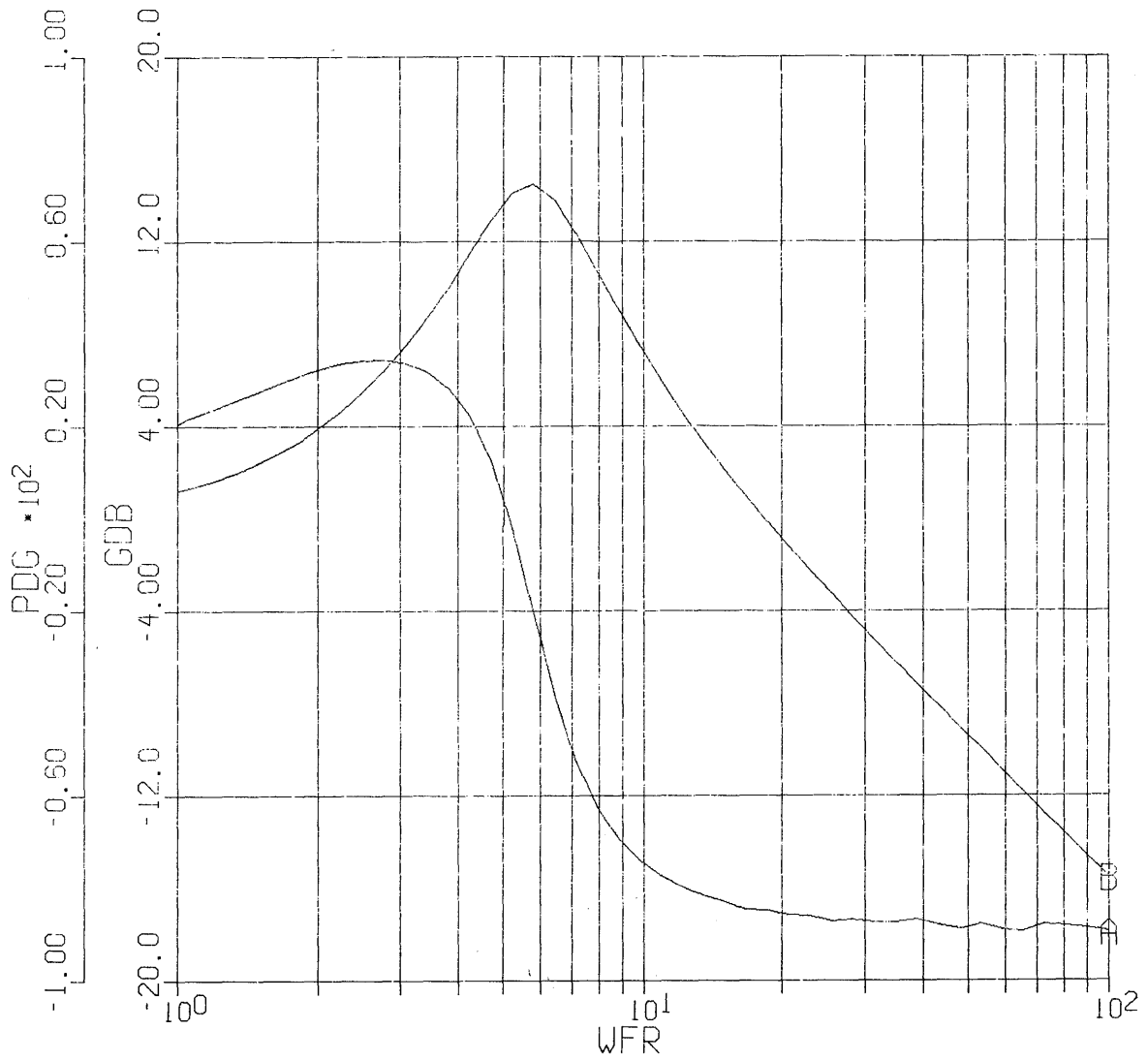


Figure A9-5. Line Plot of Phase and Gain against Frequency in Radians/second

10. MISSILE AIRFRAME MODEL

This example was chosen to show the use of vector operators, vector integration and to demonstrate how other standard FORTRAN subroutines can be incorporated into a simulation model. It is the six degree-of-freedom model of an uncontrolled ground-to-air missile with the simulation configured to produce transient responses to control surface deflections. For use in a missile system evaluation, the model would have to be extended to include a target, target sensor or seeker, guidance law and autopilot as well as expand the aerodynamic and motor descriptions. This model as it stands is complicated enough and any extensions describing actual hardware systems usually bear the burden of security classification.

In order to develop the missile model, we must define axis systems and the angles that transform between different frames. The axis system used in this model has the second or y-axis pointing up, and is found commonly in models developed for ground launched missiles (HAWK, PATRIOT). The other major school of thought orients the axes with third or z-axis pointing down, and this tends to be used by missile engineers who have graduated from airplane development. Models can be built in any system of axes, but for consistency, all frames in the simulation should become parallel when the orientation angles become zero.

For this example model, the reference or E frame has the E1 axis horizontal and down range, E2 or vertical (up) and E3 is crossrange, horizontal, out to the right to form a right hand set. The origin of the frame is normally on the ground but since we don't compensate the atmospheric density table look-up for ground altitude, in this case the origin is assumed to sit at sea level. All velocity and range vectors are normally expressed as components in this frame unless deliberately specified otherwise. The missile frame, shown in Figures A10-1 and A10-2 has M1 out of the nose along the center line, M2 along fin one (normally viewed vertically or up) and M3 out to the right along fin two. The two frames are connected by an euler sequence of rotations starting at the ground reference or E frame. ψ_M (SIM) to the left about E2, θ_M (THM) up about the new three axis E3' followed by ϕ_M (FIM) about the new one axis which should now be M1 to align the two and three axes with M2 and M3. The control fin deflections are shown in Figure A10-1 in their positive sense - trailing edge right for fins one and three, trailing edge down for fins two and four.

Units adopted in the model are derived from the slug as the unit of mass, the foot as the unit of length and the second as the unit of time. Recently the US Department of Defense has begun to require metric sizing with models developed using kilogram, meter, second as the fundamental units. Most of the existing missiles however are sized in English units.

In developing the simulation model it is important that all units be consistent and fundamental, since over half of all simulation errors can be said to be the fault of unit misconceptions, either in the equations or in the constant values. It is strongly recommended that non-basic units such as degrees or gees be eliminated from the model equations (use radians for angles and feet/second squared or meters/second squared for acceleration): Variables can be transformed in the DYNAMIC section for output into auxiliary units but keep the internal scaling within the model consistent.

The simulation model definition code is listed in Figures A10-3 through A10-6. In the INITIAL section, the integration algorithm is specified as second order, fixed step (IALG = 4); the step size is 10 msec (MAXT = 0.010); the communication interval divisor is unity (NSTP = 1) and a communication interval of 20 msec is defined. A subroutine INIT(A) is called to pass the current stability derivative matrix, A, to the aerodynamic table generators. Normally the aerodynamic tables would be real data stored in the external subroutines and the INITIAL section would be concerned with launch angles, taking into account the position and velocity of a target.

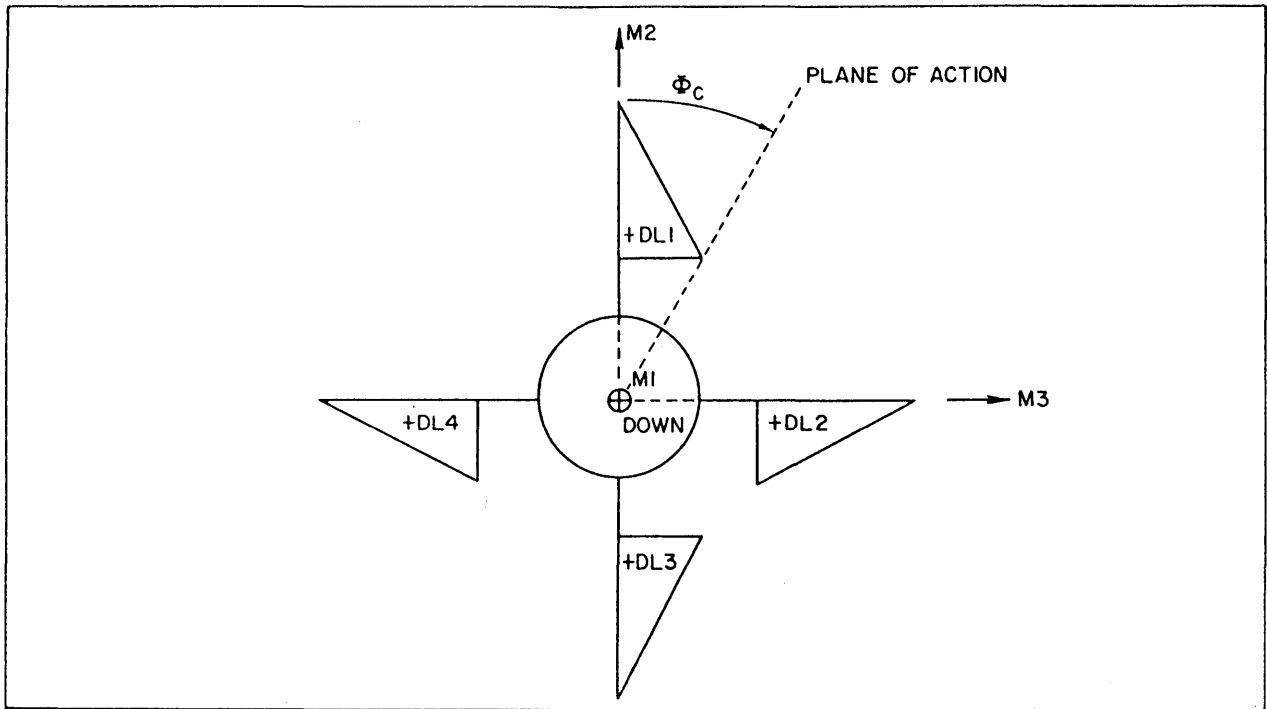


Figure A10-1. Fin Deflection Direction and Missile or M Frame Definition

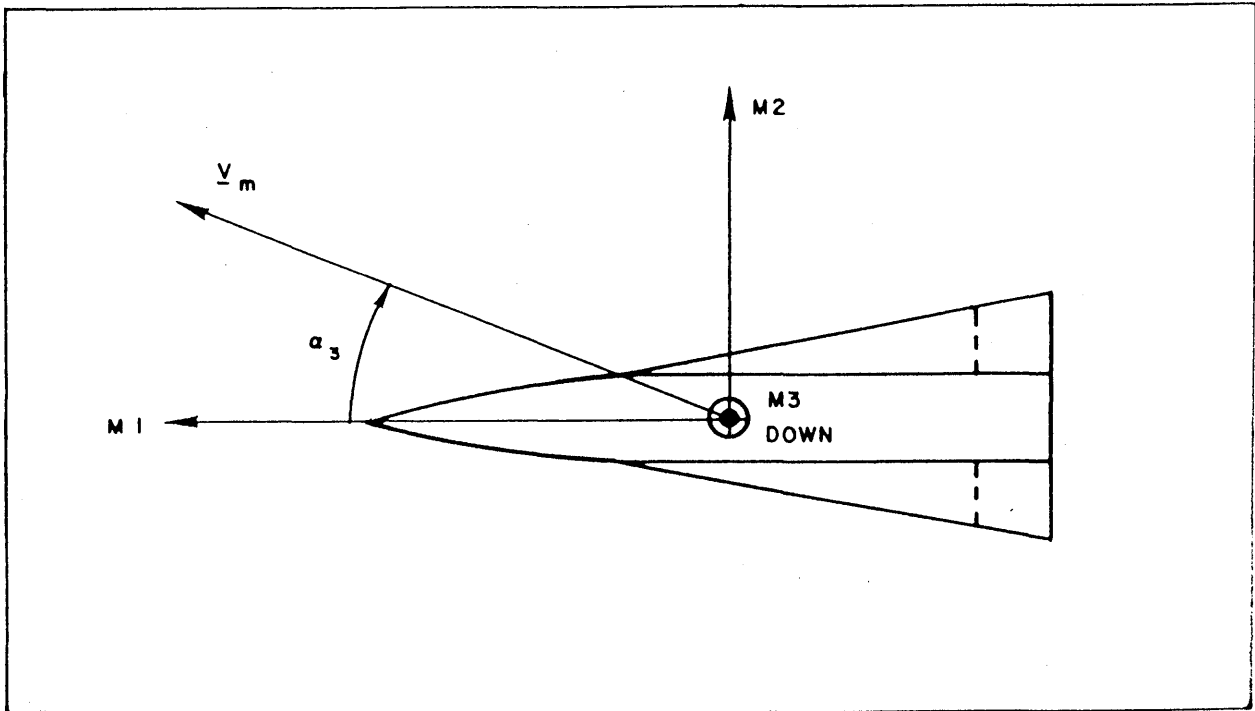


Figure A10-2. M Frame in Missile Showing Position Pitch Angle of Attack (Wind from Above)

PROGRAM - MISSILE AIRFRAME MODEL

```

  *-----A GENERIC MISSILE AIRFRAME MODEL IS *
  * DEVELOPED USING VECTORS FOR ALL THREE DIMENSIONAL QUANTITIES. *
  * THIS MODEL WILL RESPOND TO FIN DEFLECTIONS SO REPRESENTING THE *
  * OPEN LOOP AIRFRAME RESPONSE AND NEEDS A SEEKER, AUTOPILOT, *
  * ACTUATOR, MOTOR AND TARGET MODULE IN ORDER TO EVALUATE GUIDANCE *
  * EFFECTIVENESS *
  
```

INITIAL

```

  ALGORITHM      IALG = 4
  MAXTERVAL      MAXT = 0.010
  NSTEPS         NSTP = 1
  CINTERVAL      CINT = 0.020
  
```

```

  *-----PASS STABILITY DERIVATIVE MATRIX TO THE *
  *          COEFFICIENT GENERATION SUBROUTINE *
  CALL INIT(A)
  
```

END \$ OF INITIAL *

DYNAMIC

DERIVATIVE

```

  *-----ENVIRONMENT MODULE *
  *-----DEFINE ARRAYS AND CONSTANTS FOR MODULE *
  CONSTANT      G = 32.2
  *-----VELOCITY OF SOUND - FUNCTION OF ALTITUDE *
  TABLE        VS, 1, 10
  / 0.0         , 1.0E4   , 2.0E4   , 3.0E4   , 4.0E4   ...
  , 5.0E4       , 6.0E4   , 7.0E4   , 8.0E4   , 9.0E4   ...
  , 1186.5      , 1077.4  , 1036.4  , 994.8   , 968.1   ...
  , 968.1       , 968.1   , 970.9   , 977.6   , 984.3   /
  *-----LOG OF ATMOSPHERIC DENSITY *
  TABLE        LRD, 1, 10
  / 0.0         , 1.0E4   , 2.0E4   , 3.0E4   , 4.0E4   ...
  , 5.0E4       , 6.0E4   , 7.0E4   , 8.0E4   , 9.0E4   ...
  , -6.04191    , -6.34502 , -6.67084 , -7.02346 , -7.43995 ...
  , -7.91851    , -8.39664 , -8.87953 , -9.36448 , -9.87239 /
  *-----CALCULATE ACTUAL ATMOSPHERIC DENSITY *
  RD           = EXP(LRD(RM(2)))
  *-----MISSILE AIRFRAME MODULE *
  *-----DEFINE ARRAYS AND CONSTANTS FOR MODULE *
  REAL          ME(9), VMM(3), NM(3), NME(3), DL(4), CD(3), C(6)
  REAL          VM(3), VMD(3), VMIC(3), RM(3), RMD(3), RMIC(3)
  REAL          WM(3), WMD(3), WMIC(3), A(30)
  *-----MISSILE DIMENSIONAL CONSTANTS *
  CONSTANT      B = 3.95           , CBAR = 5.62
  
```

Figure A10-3. Listing of Model Definition for Missile Airframe Simulation

```

CONSTANT      S = 13.9          , DXREF = 9.60
CONSTANT      DL = 4*0.0
"-----INITIAL CONDITION VALUES "
CONSTANT      SIMIC = 0.0      , THMIC = 0.0
CONSTANT      FIMIC = 0.0      , WMIC = 3*0.0
CONSTANT      VMIC = 2154.8, 2*0.0
CONSTANT      RMIC = 0.0, 10000.0, 0.0
"-----DEFINE ELEMENTS OF STABILITY DERIVATIVE "
" MATRIX. LINEAR AERODATA IS ASSUMED FOR SIMPLICITY IN SUBROUTINE "
" COEFF. NON-LINEAR AERODATA MAY BE INCORPORATED BY REWRITING "
" THIS SUBROUTINE "
CONSTANT      A =
      0.148 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 ...
      , 0.0 , -0.26 , 0.0 , 0.0 , 0.0 , -0.286 ...
      , 0.0 , 0.0 , -0.26 , 0.0 , 0.0 , 0.286 ...
      , 0.0 , 0.528 , 0.0 , 0.0 , 0.0 , 2.0 ...
      , 0.0 , 0.0 , 0.528 , 0.0 , -2.0 , 0.0
"-----ROLL DAMPING - FUNCTION OF MACH NUMBER "
TABLE          CLP, 1, 5
      / 0.0 , 0.8 , 1.0 , 1.2 , 2.0 ...
      , -0.21 , -0.21 , -0.20 , -0.19 , -0.18 /
"-----PITCH DAMPING - FUNCTION OF MACH NUMBER "
TABLE          CMQ, 1, 5
      / 0.0 , 0.8 , 1.0 , 1.2 , 2.0 ...
      , -3.8 , -2.0 , -1.5 , -2.0 , -2.1 /
"-----MAGNITUDE OF MISSILE VELOCITY "
MVM = SQRT(DOT(VM, VM))
"-----MAKE *ME* MATRIX FROM ORIENTATION ANGLES "
CALL MMK(ME = FIM, 1, THM, 3, SIM, 2)
"-----ROTATE VELOCITY TO MISSILE FRAME "
CALL VECROT(VMM = VM, ME)
"-----LATERAL AND VERTICAL ANGLES OF ATTACK "
AL2 = ATAN(-VMM(3)/VMM(1))
AL3 = ATAN( VMM(2)/VMM(1))
"-----MACH NUMBER AND DYNAMIC PRESSURE "
MACH = MVM/VS(RM(2))
Q = 0.5*RO*MVM**2
"-----CALCULATE DAMPING DERIVATIVES "
PROCEDURAL(CD = MVM, MACH, WM)
  CD(1) = 0.5*CLP(MACH)*B*WM(1)/MVM
  CCVV = 0.5*CMQ(MACH)*CBAR/MVM
  CD(2) = CCVV*WM(2)
  CD(3) = CCVV*WM(3)
END $ OF PROCEDURAL
"-----GET MOMENTS AND FORCE AERO COEFFICIENTS "
" AND CORRECT LATERAL MOMENTS FOR SHIFT IN CENTRE OF GRAVITY "
" POSITION "
PROCEDURAL(C = AL2, AL3, DL, MACH, DXCG, DXREF)
  CALL COEFF(C = AL2, AL3, DL, MACH)
  C(2) = C(2) - (DXCG - DXREF)*C(6)/CBAR
  C(3) = C(3) + (DXCG - DXREF)*C(5)/CBAR
END $ OF PROCEDURAL
"-----CALCULATE ACCELERATION DUE TO AERODYNAMIC "
" EFFECTS AND ROTATION RATE DERIVATIVES "

```

Figure A10-4. Listing of Model Definition for Missile Airframe Simulation

```

PROCEDURAL(NM, WMD = Q, C, CD, WM, MASS, IXX, IYY)
  NM(1) = (Q*S*C(4) + THRUST)/MASS
  NM(2) = Q*S*C(5)/MASS
  NM(3) = Q*S*C(6)/MASS
  WMD(1) = Q*S*B*(C(1) + CD(1))/IXX
  WMD(2) = Q*S*CBAR*(C(2) + CD(2))/IYY + WM(1)*WM(3)
  WMD(3) = Q*S*CBAR*(C(3) + CD(3))/IYY - WM(1)*WM(2)
END $ OF PROCEDURAL
  "-----ROTATE ACCELERATION VECTOR TO EARTH FRAME"
  CALL INVROT(NME = NM, ME)
  "-----CALCULATE VELOCITY DERIVATIVES IN THE "
  " EARTH FRAME - NEEDS GRAVITY ADDING IN "
PROCEDURAL(VMD = NME, G)
  VMD(1) = NME(1)
  VMD(2) = NME(2) - G
  VMD(3) = NME(3)
END $ OF PROCEDURAL
  "-----YAW ANGLE DERIVATIVE "
  SIMD = (WM(2)*COS(FIM) - WM(3)*SIN(FIM))/COS(THM)
  "-----INTEGRATE FOR ALL EULER ANGLES - NOTE USE "
  " OF VECTOR INTEGRATOR FOR SINGLE ELEMENT "
  SIM = INTVC(SIMD, SIMIC)
  THM = INTEG(WM(2)*SIN(FIM) + WM(3)*COS(FIM), THMIC)
  FIM = INTEG(WM(1) - SIMD*SIN(THM), FIMIC)
  "-----VECTOR INTEGRATE FOR ROTATIONAL VELOCITY "
  WM = INTVC(WMD, WMIC)
  "-----TRANSLATIONAL VELOCITY "
  VM = INTVC(VMD, VMIC)
  "-----TRANSLATIONAL POSITION - NOTE THE DERIV- "
  " ATIVE VECTOR CANNOT BE A STATE VECTOR (VELOCITY) AS WELL "
  CALL XFERB(RMD = VM, 3)
  RM = INTVC(RMD, RMIC)

  "-----MOTOR MODULE "

  "-----SIMPLE VERSION WITH ZERO THRUST SPECIF- "
  " YING A BURNT OR GLIDE CONDITION "
  CONSTANT THRUST = 0.0 , MASS = 8.77
  CONSTANT IXX = 8.77 , IYY = 361.8
  CONSTANT DXCG = 10.2

END $ OF DERIVATIVE
  "-----STOP ON ELAPSED TIME "
  CONSTANT TSTP = 1.99
  TERMT(T .GE. TSTP)

END $ OF DYNAMIC
END $ OF PROGRAM

```

Figure A10-5. Listing of Model Definition for Missile Airframe Simulation

```

SUBROUTINE INIT(C)
C-----FORTRAN SUBROUTINE WHOSE ONLY JOB IS TO
C  TRANSFER THE STABILITY DERIVATIVE MATRIX TO AN ARRAY IN LABELLED
C  COMMON SO THAT IT MAY BE ACCESSED IN SUBROUTINE COEFF. NOTE NO
C  COMMON BLOCKS MAY BE DEFINED IN THE ACSL MODEL DEFINITION SECTION
C
COMMON/STABD/  A(6,5)
DATA          LENGTH / 30 /

C-----TRANSFER BLOCK
CALL XFERB(C, LENGTH, A)
RETURN

C
END
SUBROUTINE COEFF(AL2, AL3, DL, MACH, C)
C-----COMPUTES SIX AERODYNAMIC COEFFICIENTS -
C  THREE MOMENTS, C(1), C(2) AND C(3), AND THREE FORCES, C(4), C(5)
C  AND C(6). MOMENTS ARE ABOUT AXES CENTRED AT THE REFERENCE POINT
C  AND MUST BE CORRECTED FOR CENTRE OF GRAVITY SHIFT.
C
C  INPUTS
C
C  AL2      ANGLE OF ATTACK ABOUT *M2* - POSITIVE WIND FROM LEFT
C  AL3      ANGLE OF ATTACK ABOUT *M3* - POSITIVE WIND FROM ABOVE
C  DL       ARRAY OF FOUR FIN DEFLECTIONS
C  MACH     MACH NUMBER (REAL)
C
C  OUTPUTS
C
C          C          ARRAY OF SIX AERODYNAMIC COEFFICIENTS
C
C          REAL          DL(4), C(6)
C
COMMON/STABD/  A(6,5)
C-----COMPUTE EQUIVALENT CONTROL SURFACE DEFL-
C          ECTIONS FROM THE FOUR SURFACE ANGLES
DLA      = 0.25*(DL(3) + DL(4) - DL(1) - DL(2))
DLY      = 0.50*(DL(1) + DL(3))
DLZ      = 0.50*(DL(2) + DL(4))
C-----COMPUTE EACH MOMENT ASSUMING IT IS LINEAR
C          IN EACH OF THE ARGUMENTS
DO 110 J = 1, 6
C(J)     = A(J,1)*DLA + A(J,2)*DLY + A(J,3)*DLZ + A(J,4)*AL2
          + A(J,5)*AL3
110 CONTINUE
RETURN

C
END

```

Figure A10-6. Listing of FORTRAN Routines Included in Model Definition of Missile Airframe Simulation

The DERIVATIVE section is split up into logically connected code sequences or modules that help in documenting the simulation. Modules assist in checkout since individual code sequences can be removed and assigned to individuals for verification, so splitting a simulation development task among a team. We have used Control Data's UPDATE utility to maintain modules as common decks for easy access and modification. The simulation model is then a simple DECK containing ACSL structure statements and CALLs to the appropriate module.

In the missile airframe module, vector arrays are specified, then constants followed by the code to compute the derivatives of the state variables. For this simplified case, the aerodynamic characteristics are determined by values in the five by six matrix A that contains the stability derivative coefficients. Function tables are defined for roll (CLP) and pitch/yaw (CMQ) damping as functions of Mach Number. The following discussion now references the code section that starts in the middle of Figure A10-4. First the missile velocity magnitude is obtained from:

$$MVM = \text{SQRT}(\text{DOT}(VM, VM))$$

where VM is a three component vector velocity in the E frame. DOT is an external function that evaluates the dot product of two three component vectors. The next step is to form the direction cosine matrix ME, which is a three by three matrix that transforms vectors expressed in the E frame to components in the M frame. It can be calculated knowing the three angles ψ_M , θ_M and ϕ_M . A subroutine is available* MMK (matrix make) which is called so:

$$\text{CALL MMK}(A, NA, B, NB, C, NC, M)$$

which makes up a direction cosine matrix M that will transform between two axis systems that are connected by a rotation A about the NA axis, B about the NB axis and C about the NC axis. In the ACSL code the subroutine call is expressed

$$\text{CALL MMK}(ME = FIM, 1, THM, 3, SIM, 2)$$

which tells the sorter that ME is an output of the routine. For the FORTRAN program produced the translator will change the order of the arguments so that the outputs are on the right and ME will coincide with M of the above call.

The next step is to use this direction cosine matrix to obtain components of the missile velocity in the M frame from

$$\underline{VM}^{(M)} = [ME] \underline{VM}^{(E)}$$

The subroutine VECROT does this rotation and is called so:

$$\text{CALL VECROT}(VMM = VM, ME)$$

where again the order of the arguments will be inverted by the translator so that the actual FORTRAN call will be

$$\text{CALL VECROT}(VM, ME, VMM).$$

A corresponding subroutine INVROT is used later on that performs the inverse rotation

$$\underline{V1} = [ME]^{-1} \underline{V2}$$

or

$$\text{CALL INVROT}(V1 = V2, ME)$$

* The three dimensional geometry subroutines are provided in an optional library ULIB.

Lateral and vertical angles-of-attack are next calculated. The lateral angle-of-attack α_2 (=AL2) is the angle between M1 and the projection of the velocity vector on to the M1 - M3 plane and is positive for a positive rotation from M1 to the wind vector **about** M2 (wind from left). Vertical angle-of-attack α_3 (=AL3) is the angle between M1 and the projection of the velocity vector on the M1 - M2 plane and is positive for a positive rotation from M1 to the wind vector **about** M3 (wind from above). See Figure A10-2.

$$\text{i.e.,} \quad \alpha_2 = \tan^{-1} \left[\frac{-VM(M3)}{VM(M1)} \right] \text{ rad}$$

$$\alpha_3 = \tan^{-1} \left[\frac{VM(M2)}{VM(M1)} \right] \text{ rad}$$

Atmospheric density ($\rho = RO$) comes from the environment module calculated by

$$\rho = \rho_0 \exp [LROF(h)] \text{ slugs/ft}^3$$

Height h is the component of missile range along E2 or RM(2), and the log density function is used to reduce the dynamic range of the function and so make the straight line interpolation over 10 kft increments more accurate. The airframe module computes dynamic pressure from

$$q = \frac{\rho V^2}{2} \text{ lb/ft}^2$$

Mach number is obtained from the velocity of sound - function of altitude - and missile velocity so

$$\text{MACH} = \frac{MVM}{VS(h)}$$

Next dimensionless damping coefficients are obtained using normalized spin rates. The missile spin rate is expressed as a three component vector WM giving the rates as components resolved along the M1, M2 and M3 axes. The damping coefficient components of the total moment coefficients are now given (P, Q and R are replaced in the model by WM(1), WM(2) and WM(3) respectively):

$$CD(1) = C_{\ell P}(\text{MACH}) \left(\frac{Pb}{2V} \right)$$

$$CD(2) = C_{mQ}(\text{MACH}) \left(\frac{Qc}{2V} \right)$$

$$CD(3) = C_{mq}(\text{MACH}) \left(\frac{Rc}{2V} \right)$$

From symmetry, the pitch and yaw damping derivatives are assumed identical so C_{mQ} is used instead of C_{nR} . Note since the calculation is of array elements the calculations are embedded in a PROCEDURAL block that lets the system know that all values of the array CD are calculated within the block.

Next the aerodynamic coefficients - three moment and three force - are calculated using a subroutine. They can be evaluated as functions of the two angles-of-attack, α_2 and α_3 , four fin deflections $\delta_1 \dots \delta_4$ and Mach Number, seven variables in all. One of the objects of this exercise was to study the effect of different techniques in evaluating aerodynamic coefficients from full nonlinear tables of data to simple linear stability derivatives. For the case shown simple linear stability derivatives were used embedded in the (A) matrix passed to the COEFF subroutine in the INITIAL section (CALL INIT . . .). This (A) matrix is a five by six array having components

$$(A) \equiv \begin{bmatrix} C_{l\delta a} & C_{l\delta y} & C_{l\delta z} & C_{l\alpha 2} & C_{l\alpha 3} \\ C_{m\delta a} & C_{m\delta y} & C_{m\delta z} & C_{m\alpha 2} & C_{m\alpha 3} \\ C_{n\delta a} & C_{n\delta y} & C_{n\delta z} & C_{n\alpha 2} & C_{n\alpha 3} \\ C_{X\delta a} & C_{X\delta y} & C_{X\delta z} & C_{X\alpha 2} & C_{X\alpha 3} \\ C_{Y\delta a} & C_{Y\delta y} & C_{Y\delta z} & C_{Y\alpha 2} & C_{Y\alpha 3} \\ C_{Z\delta a} & C_{Z\delta y} & C_{Z\delta z} & C_{Z\alpha 2} & C_{Z\alpha 3} \end{bmatrix}$$

The effective roll, yaw and pitch fin control deflections δa , δy and δz are obtained from

$$\delta_a = 0.25 (-\delta_1 - \delta_2 + \delta_3 + \delta_4)$$

$$\delta_y = 0.5 (\delta_1 + \delta_3)$$

$$\delta_z = 0.5 (\delta_2 + \delta_4)$$

Figure A10-6 shows the implementation of subroutines COEFF and INIT to return the six component vector C having elements.

C(1) - rolling moment coefficient, about M1

C(2) - moment coefficient about M2

C(3) - moment coefficient about M3

C(4) - force coefficient along M1

C(5) - force coefficient along M2

C(6) - force coefficient along M3

Now the aerodynamic acceleration can be obtained and stored as components of the vector n_M . Aerodynamic acceleration is that produced by aerodynamic forces and excludes gravity. This quantity is that which would be read by any on-board accelerometers.

$$n_{M1} = (q S C_X + THRUST)/MASS$$

$$n_{M2} = q S C_Y/MASS$$

$$n_{M3} = q S C_Z/MASS$$

The rate accelerations have gyroscopic terms in the equations when applied to a normal spinning rigid body.

Equations of motion of a conventional aircraft have the form (neglecting I_{XZ}).

$$I_X \dot{P} = q S b C_\ell + (I_Y - I_Z)QR$$

$$I_Y \dot{Q} = q S \bar{c} C_m + (I_Z - I_X)RP$$

.....

Now for a missile having quadrant symmetry, the lateral moments of inertia are equal i.e.,

$$I_Y = I_Z$$

and in addition the long thin shape assures that the roll moment of inertia is small i.e.,

$$I_X \ll I_Y$$

Using this information the equations become

$$I_X \dot{P} = q S b [C_{\ell} + C_{\ell P} \left(\frac{P b}{2V} \right)]$$

$$I_Y \dot{Q} = q S \bar{c} [C_m + C_{mQ} \left(\frac{Q \bar{c}}{2V} \right)] + I_Y P R$$

$$I_Y \dot{R} = q S \bar{c} [C_n + C_{mQ} \left(\frac{R \bar{c}}{2V} \right)] - I_Y P Q$$

The translational accelerations are rotated from the missile frame (M) to the ground or reference frame (E) using subroutine INVROT which accomplishes

$$\underline{NM}^{(E)} = [ME]^{-1} \underline{NM}^{(M)}$$

The velocity derivatives are obtained by including the acceleration due to gravity

$$\dot{V}_M^{(E1)} = N_M^{(E1)}$$

$$\dot{V}_M^{(E2)} = N_M^{(E2)} - G$$

$$\dot{V}_M^{(E3)} = N_M^{(E3)}$$

The angular rate derivative $\dot{\psi}_M$ is obtained from the standard gimbal equation

$$\dot{\psi}_M = (Q \cos \phi_M - R \sin \phi_M) / \cos \theta_M$$

and now the three angles are obtained by integrating the respective rates.

$$\psi_M = \text{INTVC}(\dot{\psi}_M, \psi_{MIC})$$

$$\theta_M = \text{INTEG}(Q \sin \phi_M + R \cos \phi_M, \theta_{MIC})$$

$$\phi_M = \text{INTEG}(P - \dot{\psi}_M \sin \theta_M, \phi_{MIC})$$

Note since the derivative is an expression, the operator INTEG must be used for θ_M and ϕ_M . Vector integration for ψ_M works since the arguments can be considered as one component vectors. An extra assignment statement is saved since the derivative must be a unique name. The missile angular spin vector and velocity vector are integrated from the respective accelerations by the two lines:

$$WM = \text{INTVC}(WMD, WMIC)$$

$$VM = \text{INTVC}(VMD, VMIC)$$

The three elements of VM are transferred to the range derivative vector RMD so that the derivative has a unique name. For vector integration the derivative cannot be a state since it is allocated to the derivative block and the storage conflict would result. Now the three component range vector is declared to be the integral of the three component derivative or:

$$RM = \text{INTVC}(RMD, RMIC)$$

which completes the specification of the airframe six degree of freedom module.

The last line describes the motor module - see bottom of page A10-5 - that specifies zero thrust and constant mass and inertias. In actual practice a motor model must compute thrust as a function of time from motor ignition and the varying mass, inertia and center of gravity position, all of which significantly effect flight characteristics.

The output stream from exercising the model is shown in Figures A10-7 through A10-14. Starting at Figure A10-7, the TITLE is established and then the PREPAR list defined. Fixed fin deflections of -0.01 radians are specified for fins two and four, so that the motion is in the vertical plane - missile nose tends to go up. After the START, conditions are established for strip chart plots (STRPLT = .T.), with grids (GRDSPL = .T.), a half scale factor (PSFSPL = 0.5) and a longer x-axis (XINSPL = 10.0). This allows us to stack more variables on a page and the following PLOT command produces the response of Figure A10-8. The PRINT "ALL" command lists the numerical values which extends through the end of Figure A10-9. Next, the ANALYZ command is used to find a trim condition, and evaluate the Jacobian and the eigen values. Seven state variables (RM, VM and FIM) are frozen leaving five to be varied. The five roots are listed at the top of Figure A10-10, a real pole at -4.21 and double complex poles at $-1.2 \pm 16.7j$. In order to see the trim condition found, the state variables are transferred back to the initial conditions by REINIT and then one pass through the code with debug output is ensured by setting the stop time (TSTP) to zero and the debug parameter (NDEBUG) to one. The START produces the debug list which extends through Figure A10-11.

The next ANALYZ command evaluates the Jacobian of the full twelve by twelve state matrix with corresponding eigen values at the bottom of Figure A10-12. The last page of output shows moving the center of gravity to the rear (reference point forward) which models a launch or unburnt motor condition. The plot of the response is shown in Figure A10-14 with the matching eigen values listed in Figure A10-13. Note the response is more oscillatory and the dominant roots have become more unstable.

The last figure, Figure A10-15, shows the output obtained from the subroutine LISTD when a dictionary is prepared defining all the variables used in the model. The following statements were placed in the INITIAL section:

```
LOGICAL DICTN $ CONSTANT DICTN = .FALSE.  
IF(DICTN) CALL LISTD(5)  
DICTN = .FALSE.
```

At run time DICTN was SET = .TRUE. and the START card was followed by the dictionary definitions in alphabetical order. Note the indication at the end of the listing that shows definitions were not supplied for SIMIC and S. See Appendix B-3 for more details.

```

SET TITLE="MISSILE LONGITUDINAL DYNAMICS"
PREPAR T, NM, WM, AL2, AL3, VM, SIM, THM, FIM, MACH, Q
SET DL(2)=-0.01, DL(4)=-0.01, TITLE(5)="STEP IN FIN", "NOMINAL CG"
START
SET STRPLT=.T., GRDSPL=.T., PSFSPL=0.5, XINSPL=10.0, XTISPL=2.0
S PRNPLT=.F., TCWPRN=72
PLOT NM(2), WM(3), AL3, VM(2), THM
PRINT "NCIPRN"=5, "ALL"
    
```

LINE	T	NM(1)	NM(2)	NM(3)	WM(1)
0	0.	0.	-18.473875	0.	0.
5	0.1000000	0.	75.759493	0.	0.
10	0.2000000	0.	112.71735	0.	0.
15	0.3000000	0.	68.116555	0.	0.
20	0.4000000	0.	57.548205	0.	0.
25	0.5000000	0.	77.826550	0.	0.
30	0.6000000	0.	79.953864	0.	0.
35	0.7000000	0.	71.025331	0.	0.
40	0.8000000	0.	71.139716	0.	0.
45	0.9000000	0.	74.920565	0.	0.
50	1.0000000	0.	74.409056	0.	0.
55	1.1000000	0.	72.827707	0.	0.
60	1.2000000	0.	73.202458	0.	0.
65	1.3000000	0.	73.805978	0.	0.
70	1.4000000	0.	73.544976	0.	0.
75	1.5000000	0.	73.283640	0.	0.
80	1.6000000	0.	73.389862	0.	0.
85	1.7000000	0.	73.452074	0.	0.
90	1.8000000	0.	73.360048	0.	0.
95	1.9000000	0.	73.302325	0.	0.
100	2.0000000	0.	73.304638	0.	0.

LINE	WM(2)	WM(3)	AL2	AL3	VM(1)
0	0.	0.	0.	0.	2154.8000
5	0.	0.0966540	0.	-0.0072943	2154.7878
10	0.	0.0183467	0.	-0.0101561	2154.6730
15	0.	-0.0137568	0.	-0.0067039	2154.5532
20	0.	0.0234575	0.	-0.0058861	2154.4820
25	0.	0.0328096	0.	-0.0074570	2154.3843
30	0.	0.0158099	0.	-0.0076227	2154.2435
35	0.	0.0137948	0.	-0.0069321	2154.0998
40	0.	0.0212874	0.	-0.0069419	2153.9553
45	0.	0.0212679	0.	-0.0072361	2153.7895
50	0.	0.0180595	0.	-0.0071977	2153.6040
55	0.	0.0184311	0.	-0.0070765	2153.4087
60	0.	0.0197497	0.	-0.0071071	2153.2016
65	0.	0.0194276	0.	-0.0071555	2152.9780
70	0.	0.0188892	0.	-0.0071369	2152.7398
75	0.	0.0190765	0.	-0.0071185	2152.4887
80	0.	0.0192722	0.	-0.0071287	2152.2239
85	0.	0.0191557	0.	-0.0071355	2151.9445
90	0.	0.0190705	0.	-0.0071305	2151.6511
95	0.	0.0191144	0.	-0.0071283	2151.3442
100	0.	0.0191296	0.	-0.0071308	2151.0233

Figure A10-7. Run-time Commands and Output Stream for Missile Dynamics Evaluation.

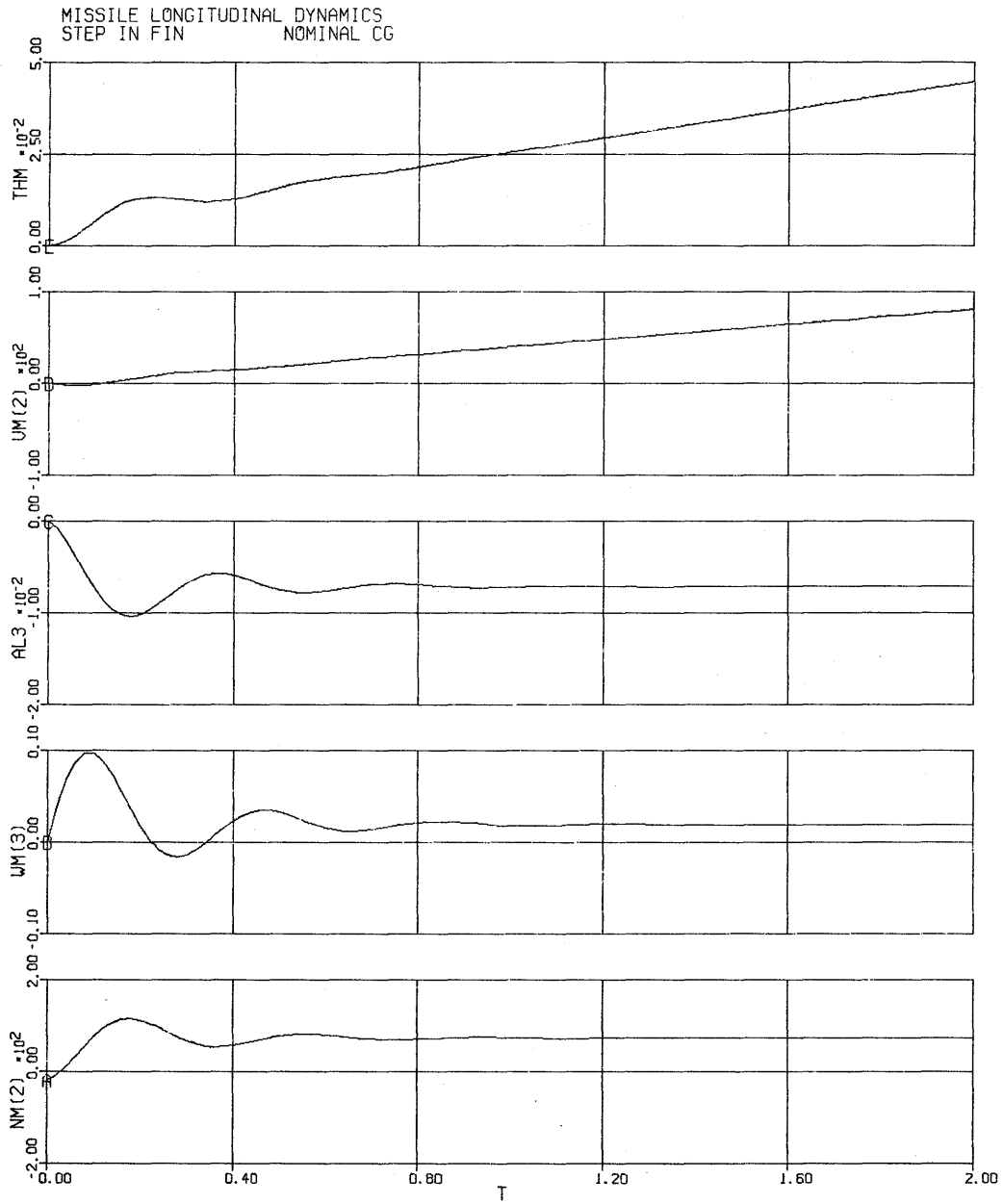


Figure A10-8. Strip Chart Plots of Missile Response to Fixed Fin Deflection - Nominal Centre of Gravity

LINE	VM(2)	VM(3)	SIM	THM	FIM
0	0.	0.	0.	0.	0.
5	-0.9254375	0.	0.	0.0068648	0.
10	6.3848135	0.	0.	0.0131193	0.
15	12.319067	0.	0.	0.0124215	0.
20	14.895778	0.	0.	0.0127998	0.
25	18.452335	0.	0.	0.0160218	0.
30	23.326785	0.	0.	0.0184506	0.
35	27.626594	0.	0.	0.0197565	0.
40	31.428341	0.	0.	0.0215319	0.
45	35.531264	0.	0.	0.0237317	0.
50	39.809254	0.	0.	0.0256806	0.
55	43.935283	0.	0.	0.0274764	0.
60	48.000948	0.	0.	0.0293962	0.
65	52.135127	0.	0.	0.0313661	0.
70	56.283527	0.	0.	0.0332761	0.
75	60.397079	0.	0.	0.0351703	0.
80	64.504384	0.	0.	0.0370907	0.
85	68.622892	0.	0.	0.0390135	0.
90	72.738132	0.	0.	0.0409234	0.
95	76.844041	0.	0.	0.0428322	0.
100	80.947254	0.	0.	0.0447451	0.

LINE	MACH	Q
0	2.0000000	4075.4612
5	1.9999862	4075.4321
10	1.9998917	4075.0020
15	1.9998119	4074.5168
20	1.9997713	4074.1274
25	1.9997187	4073.6431
30	1.9996478	4073.0123
35	1.9995811	4072.3222
40	1.9995178	4071.5812
45	1.9994485	4070.7518
50	1.9993745	4069.8340
55	1.9992996	4068.8442
60	1.9992229	4067.7803
65	1.9991428	4066.6358
70	1.9990598	4065.4125
75	1.9989747	4064.1131
80	1.9988870	4062.7368
85	1.9987966	4061.2825
90	1.9987037	4059.7511
95	1.9986082	4058.1431
100	1.9985101	4056.4583

ANALYZ "FREEZE"=RM, VM, FIM, "TRIM", "JACOB", "EIGEN"

NEW JACOBIAN EVALUATED

ROW VECTOR NAMES

SIM	1	THM	2	WM(1)	3
VM(2)	4	VM(3)	5		

COLUMN VECTOR NAMES

SIMD	1	Z09982	2	WMD(1)	3
WMD(2)	4	WMD(3)	5		

Figure A10-9. Output Stream from Missile Dynamics Evaluation

MATRIX ELEMENTS - ROWS ACROSS, COLUMNS DOWN

1	0.	0.	0.	1.0000266	0.
2	0.	0.	0.	0.	1.0000000
3	0.	0.	-4.2094146	0.	0.
4	-276.73213	0.	0.	-2.4097832	0.
5	0.	-276.72476	0.	0.	-2.4097832

COMPLEX EIGEN VALUES IN ASCENDING ORDER

1	-4.20941462	0.
2	-1.20489158	16.5913531
3	-1.20489158	-16.5913531
4	-1.20489158	-16.5917972
5	-1.20489158	16.5917972

REINIT

S TSTP=0.0,NDEBUG=1

START

....DEBUG DUMP - SYSTEM VARIABLES. NDEBUG IS 1

T	0.	ZZTICG	0.	CINT	0.02000000
ZZIERR	F	ZZNBLK	1	ZZI	1
ZZST	T	ZZFRFL	T	ZZICFL	T
ZZRNFL	T	ZZNS	12	MINT	1.0000E-10
MAXT	0.01000000	NSTP	1	IALG	4

STATE VARIABLES

FIM	0.
RM(1)	0.
RM(2)	10000.0000
RM(3)	0.
SIM	0.
THM	0.00729676
VM(1)	2154.80000
VM(2)	0.
VM(3)	0.
WM(1)	0.
WM(2)	0.
WM(3)	0.

DERIVATIVES

Z09981	0.
RMD(1)	2154.80000
RMD(2)	0.
RMD(3)	0.
SIMD	0.
Z09982	0.
VMD(1)	-0.55302704
VMD(2)	43.5894323
VMD(3)	0.
WMD(1)	0.
WMD(2)	0.
WMD(3)	-1.8318E-14

INITIAL CONDITIONS

FIMIC	0.
RMIC(1)	0.
RMIC(2)	10000.0000
RMIC(3)	0.
SIMIC	0.
THMIC	0.00729676
VMIC(1)	2154.80000
VMIC(2)	0.
VMIC(3)	0.
WMIC(1)	0.
WMIC(2)	0.
WMIC(3)	0.

ALGEBRAIC VARIABLES

A(30)	0.
B	3.95000000
	-2.0817E-17
	0.
CD	0.
CLP	-0.21000000
	-0.19000000
	0.80000000
	2.00000000
	-1.50000000
	0.
	1.20000000
	-0.01000000
DXCG	10.20000000
IXX	8.77000000
MACH	2.00000000

AL2	0.
C	0.
	0.
CBAR	5.62000000
	0.
	-0.21000000
	-0.18000000
	1.00000000
CMQ	-3.80000000
	-2.00000000
	0.80000000
	2.00000000
	0.
DXREF	9.60000000
IYY	361.800000
MASS	8.77000000

AL3	-0.00729676
	0.
	0.01173352
CCVV	-0.00273854
	0.
	-0.20000000
	0.
	1.20000000
	-2.00000000
	-2.10000000
	1.00000000
DL	0.
	-0.01000000
G	32.20000000
LRO(20)	90000.0000
ME	0.99997338

Figure A10-10. Output Stream (Eigen Values and Debug) from Missile Dynamics Evaluation

-0.00729669	0.	0.00729669
0.99997338	0.	0.
0.	1.00000000	MVM 2154.80000
NM 0.	75.7914500	0.
NME-0.55302704	75.7894323	0.
Q 4075.46122	RO 0.00175547	S 13.9000000
THRUST 0.	TSTP 0.	VMM 2154.74264
-15.7229168	0.	VS(20) 90000.0000
Z09983 0.	Z09984-2.10000000	Z09985 0.
Z09986-0.18000000	Z09987 0.	Z09988 1077.40000
Z09993 0.	Z09994-6.34502000	Z09999 0
ZZSEED 555555555	ZZTLXP T	XICITG 0.
Z09989-3.80000000	-2.00000000	-1.50000000
-2.00000000	-2.10000000	Z09990 0.
0.80000000	1.00000000	1.20000000
2.00000000	Z09991-0.21000000	-0.21000000
-0.20000000	-0.19000000	-0.18000000
Z09992 0.	0.80000000	1.00000000
1.20000000	2.00000000	Z09995-6.04191000
-6.34502000	-6.67084000	-7.02346000
-7.43995000	-7.91851000	-8.39664000
-8.87953000	-9.36448000	-9.87239000
Z09996 0.	10000.0000	20000.0000
30000.0000	40000.0000	50000.0000
60000.0000	70000.0000	80000.0000
90000.0000	Z09997 1186.50000	1077.40000
1036.40000	994.800000	968.100000
968.100000	968.100000	970.900000
977.600000	984.300000	Z09998 0.
10000.0000	20000.0000	30000.0000
40000.0000	50000.0000	60000.0000
70000.0000	80000.0000	90000.0000

ANALYZ "JACOB", "EIGEN"
 ROW VECTOR NAMES

FIM	1	RM(1)	2	RM(2)	3
RM(3)	4	SIM	5	THM	6
VM(1)	7	VM(2)	8	VM(3)	9
WM(1)	10	WM(2)	11	WM(3)	12

COLUMN VECTOR NAMES

Z09981	1	RMD(1)	2	RMD(2)	3
RMD(3)	4	SIMD	5	Z09982	6
VMD(1)	7	VMD(2)	8	VMD(3)	9
WMD(1)	10	WMD(2)	11	WMD(3)	12

MATRIX ELEMENTS - ROWS ACROSS, COLUMNS DOWN

1	0.	0.	0.	0.	0.
	0.	0.	0.	0.	1.00000000
	-0.0072969	0.			
2	0.	0.	0.	0.	0.
	0.	1.00000000	0.	0.	0.
	0.	0.			

Figure A10-11. Output Stream from Missile Dynamics Evaluation

3	0.	0.	0.	0.	0.
	0.	0.	1.0000000	0.	0.
	0.	0.			
4	0.	0.	0.	0.	0.
	0.	0.	0.	1.0000000	0.
	0.	0.			
5	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	1.0000266	0.			
6	0.	0.	0.	0.	0.
	0.	0.	0.	0.	0.
	0.	1.0000000			
7	0.	0.	1.739E-05	0.	0.
	-170.05392	-5.133E-04	0.0437463	0.	0.
	0.	0.			
8	0.	0.	-0.0023833	0.	0.
	12917.897	0.0703447	-5.9951967	0.	0.
	0.	0.			
9	-18.475548	0.	0.	0.	-12918.585
	0.	0.	0.	-5.9955159	0.
	0.	0.			
10	0.	0.	0.	0.	0.
	0.	0.	0.	0.	-4.2094146
	0.	0.			
11	-2.0192297	0.	0.	0.	-276.73213
	0.	0.	0.	-0.1284259	0.
	-2.4097832	0.			
12	0.	0.	5.760E-19	0.	0.
	-276.72476	-1.700E-17	0.1284225	0.	0.
	0.	-2.4097832			

COMPLEX EIGEN VALUES IN ASCENDING ORDER

1	0.	0.
2	0.	0.
3	1.2070E-14	0.
4	3.1692E-08	0.
5	-6.0184E-08	0.
6	-3.0912E-04	0.06794567
7	-3.0912E-04	-0.06794567
8	-4.20941462	0.
9	-4.20243747	16.5379930
10	-4.20243747	-16.5379930
11	-4.20264954	-16.5381496
12	-4.20264954	16.5381496

S DXREF=10.7, TITLE(7)='AFT CG

Figure A10-12. Output Stream (Jacobian and Eigen Values) from Missile Dynamics Evaluation

ACSL RUN-TIME EXEC VERSION 1 LEVEL 6M 81/08/11. 17.47.56. PAGE 6
MISSILE LONGITUDINAL DYNAMICS STEP IN FIN AFT CG

S THMIC=0.0, TSTP=1.99
START
PLOT NM(2), WM(3), AL3, VM(2), THM
ANALYZ "EIGEN"
COMPLEX EIGEN VALUES IN ASCENDING ORDER
1 0. 0.
2 0. 0.
3 -1.7439E-08 0.
4 7.3633E-08 0.
5 1.4193E-05 0.
6 -9.1656E-05 0.03939876
7 -9.1656E-05 -0.03939876
8 -4.20851876 0.
9 -4.20152799 -24.8543608
10 -4.20152799 24.8543608
11 -4.20159641 24.8543850
12 -4.20159641 -24.8543850

STOP

END DISSPLA -- 3651 VECTORS GENERATED IN 2 PLOT FRAMES.
1382 WORDS TABLE SPACE USED

Figure A10-13. Output Stream from Missile Dynamics Evaluation Forward Center of Gravity Position.

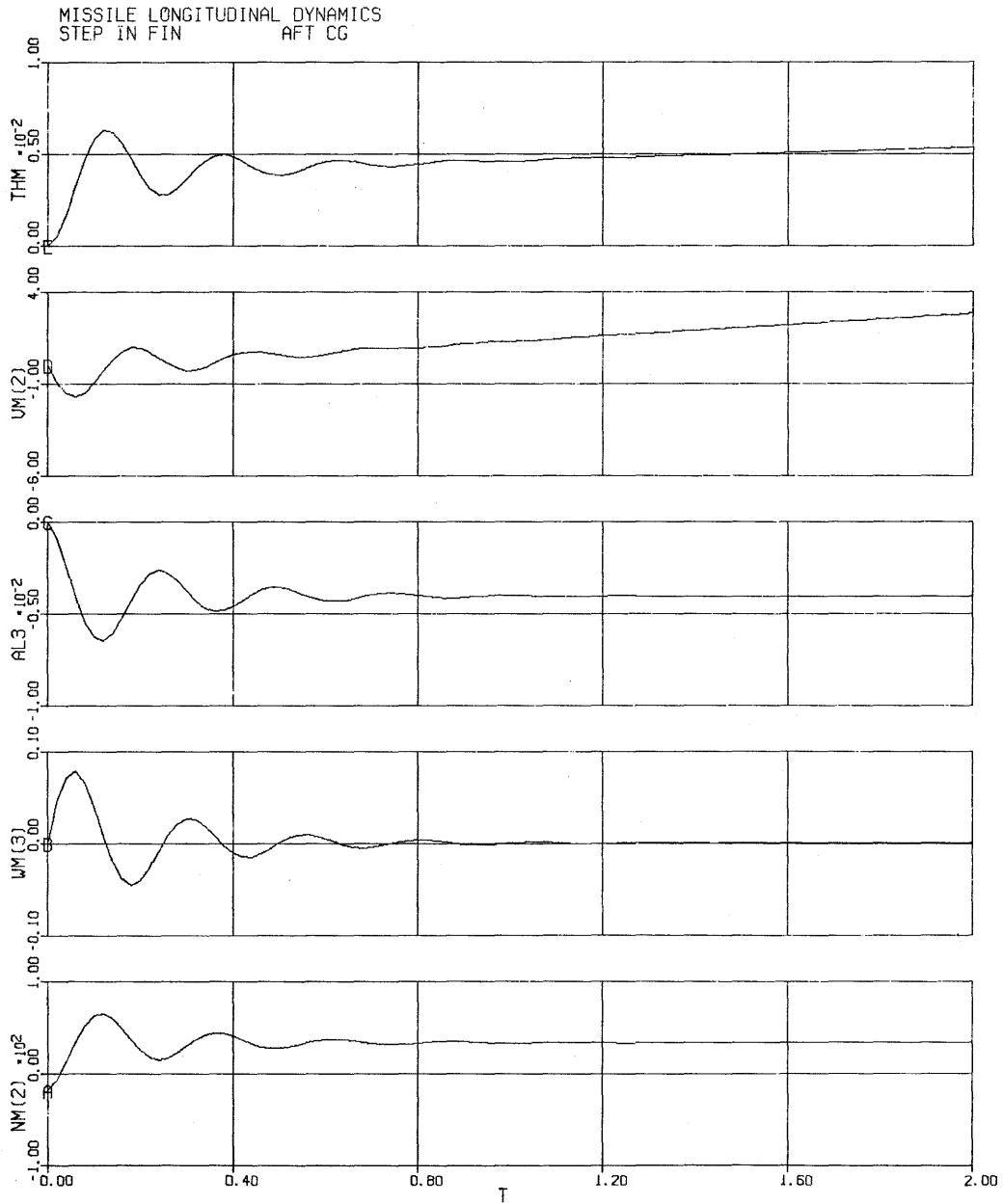


Figure A10-14. Strip Chart Plots of Missile Response to Fixed Fin Deflections - Aft Centre of Gravity

```

A      REAL( 30)  ARRAY OF AERODYNAMIC STABILITY DERIVATIVES
AL2    I          ANGLE OF ATTACK ABOUT *M2*, POSITIVE WIND FROM ABOVE (RAD)
AL3    I          ANGLE OF ATTACK ABOUT *M3*, POSITIVE FROM RIGHT (RAD)
B      3.95000000 CHARACTERISTIC SPAN (FT)
C      REAL( 6)   ARRAY OF AERO COEFFICIENTS
CBAR   5.62000000 CHARACTERISTIC CHORD (FT)
CCVV   I          CHORD OVER VELOCITY (SEC)
CD     REAL( 3)   DAMPING DERIVATIVE - DIMENSIONLESS MOMENT DERIVATIVES
CINT   0.02000000 COMMUNICATION INTERVAL (SEC)
CLP    REAL( 10)  ROLL DAMPING COEFFICIENT
CMQ    REAL( 10)  PITCH AND YAW DAMPING COEFFICIENT
DICTDM T          DICTIONARY DUMP FLAG
DL     REAL( 4)   FIN DEFLECTION (RAD)
DXCG   10.20000000 CENTRE OF GRAVITY POSITION, POSITIVE TO REAR (FT)
DXREF  9.60000000 AERODYNAMIC REFERENCE POINT, POSITIVE TO REAR (FT)
FIM    I          ROLL ANGLE OF MISSILE OR ** FRAME (RAD)
FIMIC  0.         INITIAL CONDITION ON *FIM* (RAD)
G      32.20000000 GRAVITY ACCELERATION (FT/SEC**2)
IALG   4          INTEGRATION ALGORITHM (INTEGER)
IXX    8.77000000 ROLL INERTIA (SLUG-FT**2)
IYY    361.800000 LATERAL INERTIA (SLUG-FT**2)
LRO    REAL( 20)  LOG ATMOSPHERIC DENSITY
MACH   I          MACH NUMBER
MASS   8.77000000 MISSILE MASS (SLUG)
MAXT   0.01000000 MAXIMUM INTEGRATION STEP SIZE (SEC)
MINT   1.0000E-10 MINIMUM INTEGRATION STEP SIZE (SEC)
ME     REAL( 9)   MATRIX TO TRANSFORM FROM *E* FRAME TO ** OF MISSILE FRAME
MVM    I          MAGNITUDE OF *VM* (FT/SEC)
NM     REAL( 3)   AERODYNAMIC ACCELERATION VECTOR OF MISSILE (FT/SEC**2)
NME    REAL( 3)   COMPONENTS OF *NM* EXPRESSED IN *E* FRAME (FT/SEC**2)
NSTP   1          NUMBER OF STEPS (NOT USED) (INTEGER)
Q      I          DYNAMIC PRESSURE (LBS/FT**2)
RM     REAL( 3)   RANGE VECTOR FOR MISSILE (FT)
RMD    REAL( 3)   DERIVATIVE OF *RM* (M/SEC)
RMIC   REAL( 3)   INITIAL CONDITION ON *RM* (FT)
RO     I          ATMOSPHERIC DENSITY (SLUG/FT**3)
SIM    I          YAW ANGLE OF MISSILE (RAD)
SIMD   I          DERIVATIVE OF *SIM* (RAD/SEC)
T      0.         INDEPENDENT VARIABLE TIME (SEC)
THM    I          MISSILE PITCH ANGLE (RAD)
THMD   *****  DERIVATIVE OF *THM* (RAD/SEC)
THMIC  0.         INITIAL CONDITION ON *THM* (RAD)
THRUST 0.         THRUST (LBS)
TSTP   1.99000000 TIME STOP FOR SIMULATION (SEC)
VM     REAL( 3)   VELOCITY OF MISSILE (FT/SEC)
VMD    REAL( 3)   ACCELERATION OF MISSILE (FT/SEC**2)
VMIC   REAL( 3)   INITIAL CONDITION ON *VM* (FT/SEC)
VMM    REAL( 3)   COMPONENTS OF *VM* IN ** FRAME (FT/SEC)
VS     REAL( 20)  VELOCITY OF SOUND (FT/SEC)
WM     REAL( 3)   SPIN RATE VECTOR OF MISSILE (RAD/SEC)
WMD    REAL( 3)   DERIVATIVE OF *WM* (RAD/SEC**2)
WMIC   REAL( 3)   INITIAL CONDITION ON *WM* (RAD/SEC)
XICITG 0.         TEMPORARY INITIAL CONDITION ON *T*
SIMIC  0.         *
S      13.90000000 *

```

Figure A10-15. Dictionary Listing Produced from Subroutine LISTD in the INITIAL section

11. DISCRETE SAMPLED COMPENSATOR

This example was chosen to illustrate the use of a DISCRETE section in modelling a sampled data system in which a control computer interacts with a continuous plant. The system block diagram is shown in Figure A11-1 where the plant is represented by a transfer function:

$$\frac{1}{s(s+1)}$$

This system can be visualized as a water level control problem where the valve controlling the flow into the tank has a one second time constant. An instantaneous level measurement, X , is assumed which is compared with a desired level X_C and the error sampled every tenth of a second. A digital controller is to be designed that takes the samples of level error E and outputs a command U to the valve so closing the loop. This control is taken to be a linear combination of the current error E_n , the previous error E_{n-1} , and the previous control U_{n-1} . Expressed in Z-transform notation this becomes

$$\frac{U}{E} = \frac{a_0 - a_1 z^{-1}}{1 - b_1 z^{-1}}$$

In terms of equivalent lead/lag network, the a and b coefficients above can be written

$$a_0 = \frac{T_{LED}}{T_{LAG}} \exp(-T_S (1/T_{LAG} - 1/T_{LED}))$$

$$a_1 = \frac{T_{LED}}{T_{LAG}} \exp(-T_S/T_{LAG}); \quad b_1 = \exp(-T_S/T_{LAG})$$

which gives unity steady state gain and reduces to a zero order hold when the lead and lag time constants are equal.

The program or model definition code to represent this plant and control system is shown in Figure A11-2. In this model we have made use of the multiple derivative section capability where different blocks of code can be given their own integration algorithm and step size.

In the INITIAL section the coefficients of the digital filter are calculated so that the controller response can be thought of in terms of equivalent lead/lag time constants. At the same time the discrete control (UD) and the previous error (EP) are initialized to zero. Since they are needed before they are calculated, they are effectively system state variables, though not obtained by integration.

In the continuous section, the plant is modelled by the one line:

$$X = \text{INTEG}(\text{REALPL}(TAD, KU*U), 0.0)$$

The control (U) is selected to take either the discrete value (UD) or that produced by a continuous lead-lag compensator (UC) based on a switch DISC i.e.

$$U = \text{RSW}(\text{DISC}, \text{UD}, \text{UC})$$

which will allow test at run time between the effect of the discrete or the continuous controller by changing the logical variable DISC from .FALSE. to .TRUE. Integration step size for this section will be 0.02 seconds, specified both by the communication interval (CINT) and by the global MAXT. The reason for such a comparatively short step is in order to record data during the sampling so that the discrete hold (sample interval 0.1 sec) can be seen on the plots. The communication interval chosen gives five points for every sample, so squaring the corners when the control (U) is plotted in Figures A11-4 through A11-6. If the communication interval had been increased to 0.1 seconds or more, the straight lines drawn between points would have masked the sampling action, though the simulation would still behave the same internally.

In the DISCRETE block, the step size is controlled by the INTERVAL statement which specifies the name of the controlling variable as TSAMP with a value of 0.1 seconds. The code will execute once a T equals zero and then once thereafter, every 0.1 seconds. Since the order of the code is important, it is bracketted by PROCEDURAL . . . END statements which prevent the reordering of any code within the block. Notice that there is no input/output list on the PROCEDURAL statement since the entire DISCRETE section is made procedural. The input/output list is only necessary to ensure correct ordering relative to other statements within the same DISCRETE or DERIVATIVE section. Statements will never be reordered across a section boundary.

Within the PROCEDURAL block, the new control is calculated based on the previous control, previous error and current error or

$$UD = B1*UD + AO*EP$$

and then the current error is transferred to the previous error by EP = E. If we hadn't used the PROCEDURAL block, the EP assignment would have been moved in front of its use in the UD calculation so both E and EP would now contain the current error, when the new UD was calculated, not what was intended in the code.

With the use of the DISCRETE section, the continuous section is guaranteed to be at the sample time when the DISCRETE section code is executed, so the value of current error (E) used is that actually at the sample time: A new value of control is calculated that is immediately available to the continuous section for use over the next sample interval. This action models a control computer with no calculation delay. In actual practice, dedicated controllers are usually compute bound, sampling from the outside world, calculating the new control action through the sample period and transferring this value back to the continuous section at the same time that a new sample is obtained. Modelling this action requires modifying the control calculation, first assigning the next control to the output control and then calculating a new next control i.e.

$$UD = UDN$$

$$UDN = B1*UDN + AO*E - A1*EP$$

Now both UD and UDN are effectively state variables and so must be initialized.

The execution of the model as listed in Figure A11-2 is shown in Figure A11-3. The first run (START) is followed by a column PRINT of the numeric values and a line plot of the base-line response shown in Figure A11-4. Next the OUTPUT list is CLEARED and two more runs are made - first with a equivalent lead/lag ratio of 5:1 (see Figure A11-4) and then with a reduction of the loop gain from 5.0 to 1.0 (See Figure A11-5). Lastly the response of the continuous feedback compensator is obtained, using the same values of lead/lag time constants and gain, by setting the switch variable DISC to .FALSE. The response is shown in Figure A11-7.

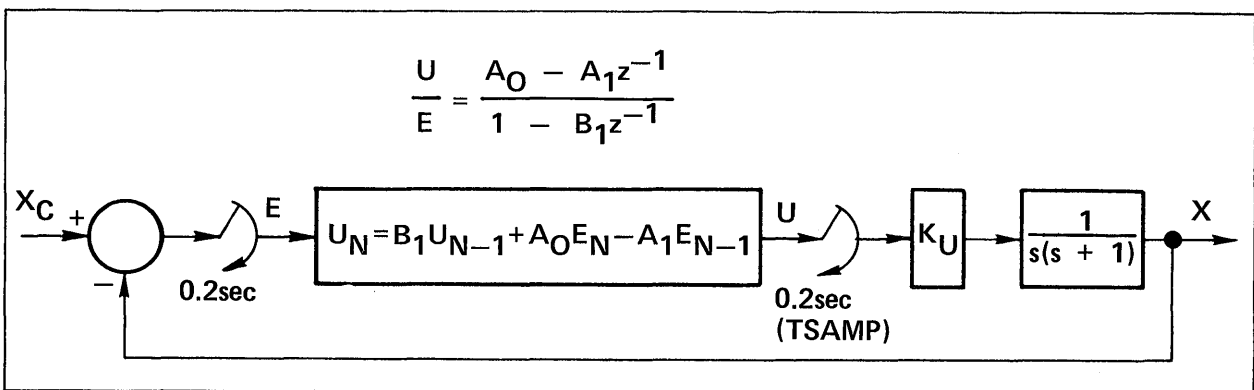


Figure A11-1. Discrete Control System Block Diagram

PROGRAM DISCRETE SAMPLED COMPENSATOR

```

"-----MODELS A CONTINUOUS PLANT WITH EITHER A "
" CONTINUOUS OR DISCRETE FEED BACK CONTROLLER WRAPPED AROUND IT. "
" PLANT CAN BE VISUALISED AS A LEVEL CONTROLLER USING VALVE "
" WITH SINGLE LAG TIME CONSTANT (= 1 SECOND) "

LOGICAL      DISC
CONSTANT     KU = 5.0          , TAD = 1.0
CONSTANT     DISC = .TRUE.    , XC = 1.0
CONSTANT     TLED = 0.5       , TLAG = 0.5
CONSTANT     TSTP = 4.9

INITIAL
CINTERVAL   CINT = 0.02
NSTEPS      NSTP = 1
MAXTERVAL   MAXT = 0.020
"-----CALCULATE Z TRANSFORM CONTROLLER GAINS "
B1          = EXP(-TSAMP/TLAG)
A0          = TLED*EXP(-TSAMP*(1.0/TLAG - 1.0/TLED))/TLAG
A1          = TLED*EXP(-TSAMP/TLAG)/TLAG
"-----INITIALISE PSEUDO STATE VARIABLES "
UD          = 0.0
EP          = 0.0
END $" OF INITIAL "

DERIVATIVE CONTINUOUS
"-----PLANT MODEL "
X          = INTEG(REALPL(TAD, KU*KU), 0.0)
"-----CONTINUOUS CONTROL ACTION "
UC         = LEDLAG(TLED, TLAG, E)
E          = XC - X
"-----CHOOSE DISCRETE OR CONTINUOUS CONTROL "
U          = RSW(DISC, UD, UC)
TERMT(T .GE. TSTP)
END $" OF CONTINUOUS SECTION "

DISCRETE DISCRT
INTERVAL TSAMP = 0.1
PROCEDURAL
"-----DISCRETE CONTROL IS LINEAR COMBINATION OF "
" PREV CONTROL, PREV ERROR AND CURRENT ERR "
UD         = B1*UD + A0*E - A1*EP
"-----CURRENT ERROR BECOMES PREVIOUS ERROR "
EP         = E
END $" OF PROCEDURAL "
END $" OF DISCRETE SECTION "

END $" OF PROGRAM "

```

Figure A11-2. Model Definition Code for Discrete Sampled Compensator

```

S TITLE="DISCRETE SAMPLED CONTROLLER",TITLE(5)="BASE CASE"
SET TCWPRN=72 $" FORCE 3 COLUMN OUTPUT FORMAT "
PREPAR T,U,X,E
OUTPUT T,U,X,"NCIOUT"=50
START
      T 0.                U 0.                X 0.
      T 1.00000000       U-0.17060285       X 1.30834973
      T 2.00000000       U-0.32541587       X 1.22638539
      T 3.00000000       U 0.34323663       X 0.67115330
      T 4.00000000       U-0.12205848       X 1.15748828
      T 4.92000000       U-0.06463221       X 1.05753083
PRINT "ALL", "NCIPRN"=10

```

LINE	T	U	X	E
0	0.	0.	0.	1.0000000
10	0.2000000	0.9758129	0.0930688	0.9069312
20	0.4000000	0.7998402	0.3376226	0.6623774
30	0.6000000	0.5032633	0.6683773	0.3316227
40	0.8000000	0.1565214	1.0134606	-0.0134606
50	1.0000000	-0.1706028	1.3083497	-0.3083497
60	1.2000000	-0.4215391	1.5065476	-0.5065476
70	1.4000000	-0.5613530	1.5855153	-0.5855153
80	1.6000000	-0.5800839	1.5474411	-0.5474411
90	1.8000000	-0.4910931	1.4154223	-0.4154223
100	2.0000000	-0.3254159	1.2263854	-0.2263854
110	2.2000000	-0.1236905	1.0224804	-0.0224804
120	2.4000000	0.0725356	0.8427189	0.1572811
130	2.6000000	0.2284609	0.7163382	0.2836618
140	2.8000000	0.3214041	0.6588552	0.3411448
150	3.0000000	0.3432366	0.6711533	0.3288467
160	3.2000000	0.2998426	0.7413469	0.2586531
170	3.4000000	0.2081127	0.8487028	0.1512972
180	3.6000000	0.0913604	0.9686219	0.0313781
190	3.8000000	-0.0257924	1.0776326	-0.0776326
200	4.0000000	-0.1220585	1.1574883	-0.1574883
210	4.2000000	-0.1828818	1.1977512	-0.1977512
220	4.4000000	-0.2021441	1.1966035	-0.1966035
230	4.6000000	-0.1821059	1.1599839	-0.1599839
240	4.8000000	-0.1318374	1.0994379	-0.0994379

```

S CALPLT=. T., PRNPLT=. F., GRIDCPL=. T.
PLOT U, X
OUTPUT "CLEAR"
S TLED=2.5, TITLE(5)="KU=5.0 TLED=2.5" $ START $ PLOT U, X
S KU =1.0, TITLE(5)="KU=1.0 TLED=2.5" $ START $ PLOT U, X
" NOW COMPARE CONTINUOUS CONTROLLER "
S TITLE="CONTINUOUS ANALOG FEEDBACK CONTROLLER", DISC=. FALSE.
START
PLOT U, X
STOP

```

Figure A11-3. Output Stream from Execution of Discrete Sampled Compensator Model

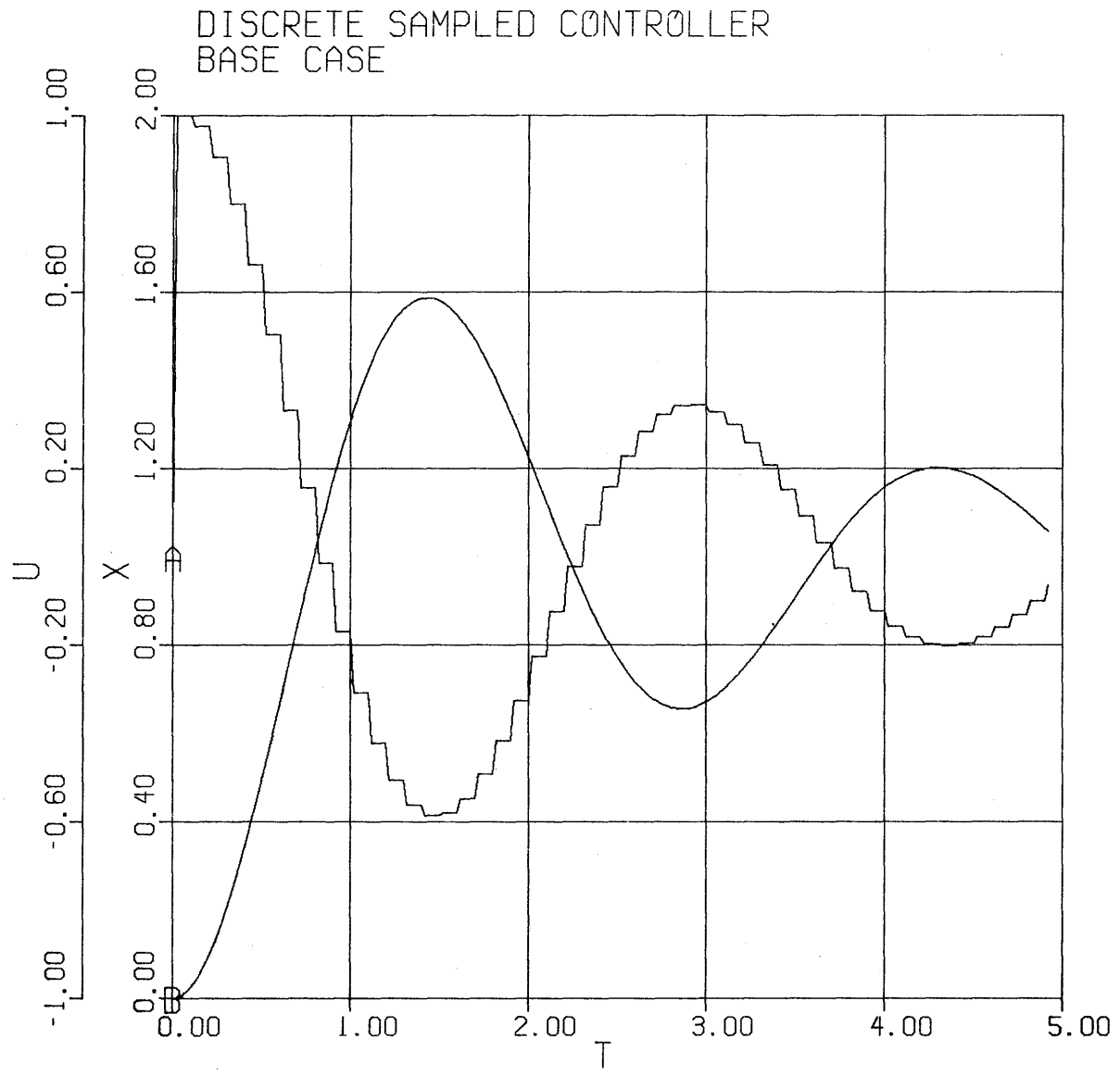


Figure A11-4. Plot of Control and Level against Time for Base Case, Discrete Sampled Compensator

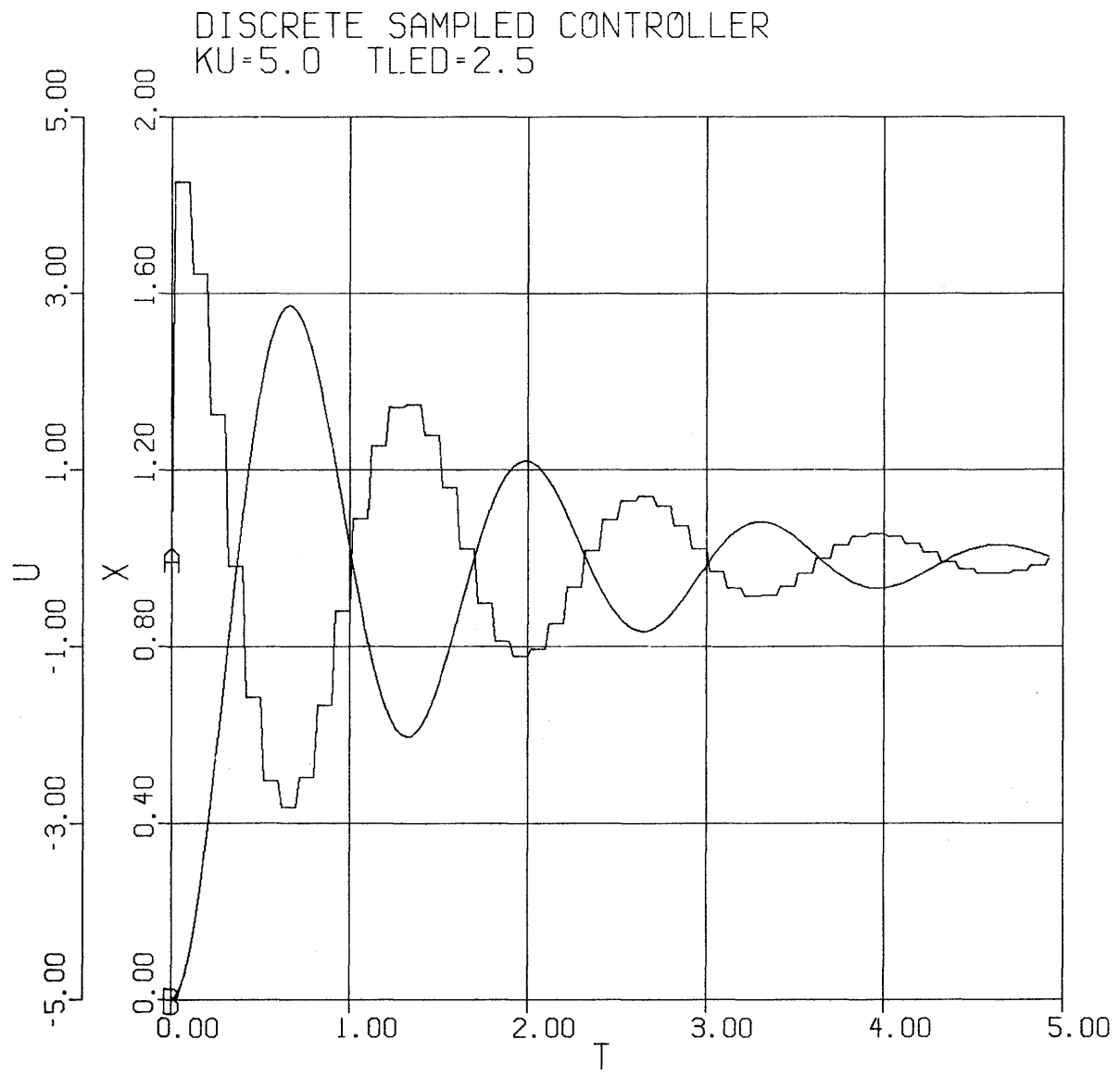


Figure A11-5. Plot of Control and Level against Time for Lead/Lag Ratio of 5

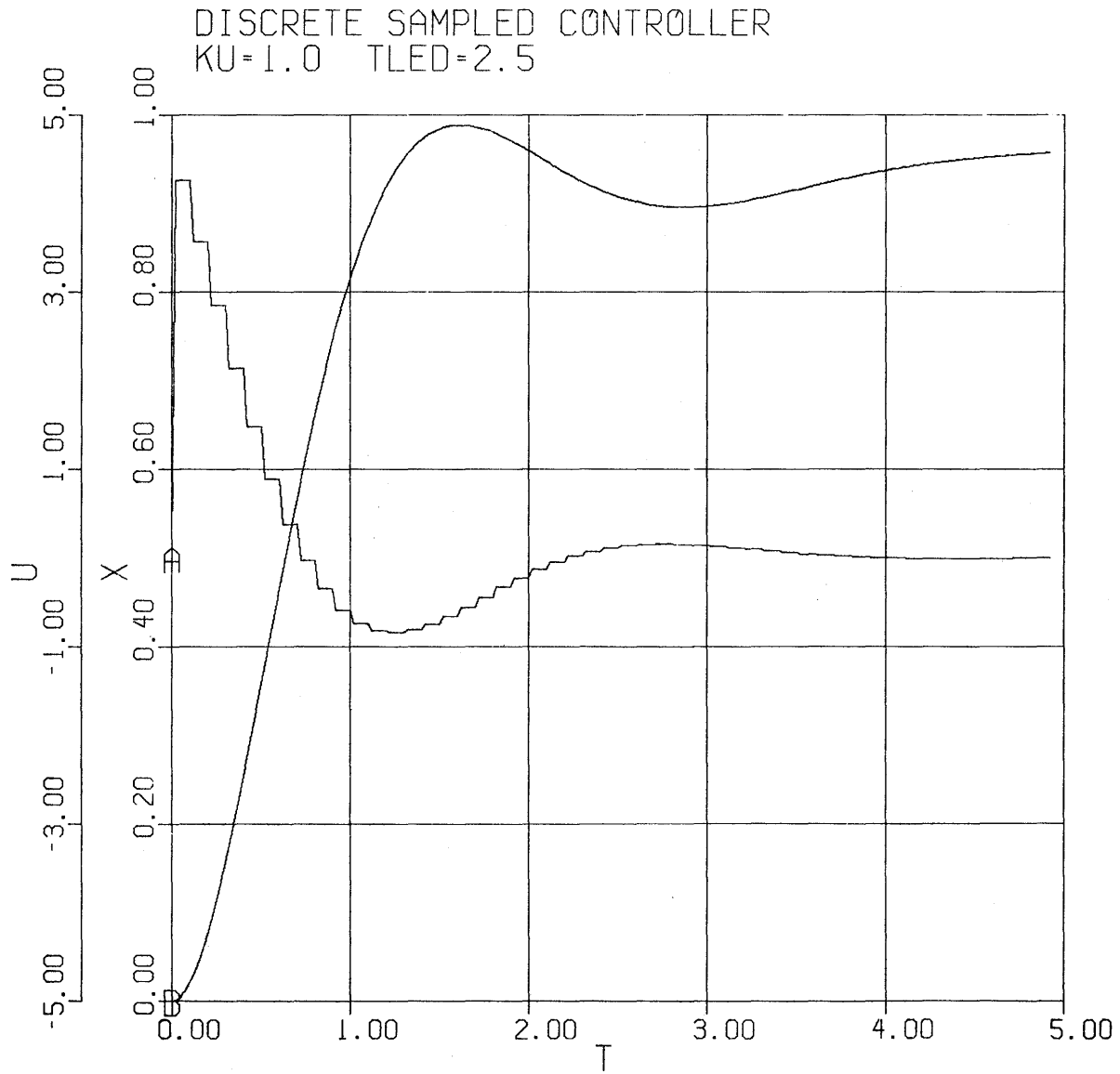


Figure A11-6. Plot of Control and Level against Time for Reduced Loop Gain, Final Design

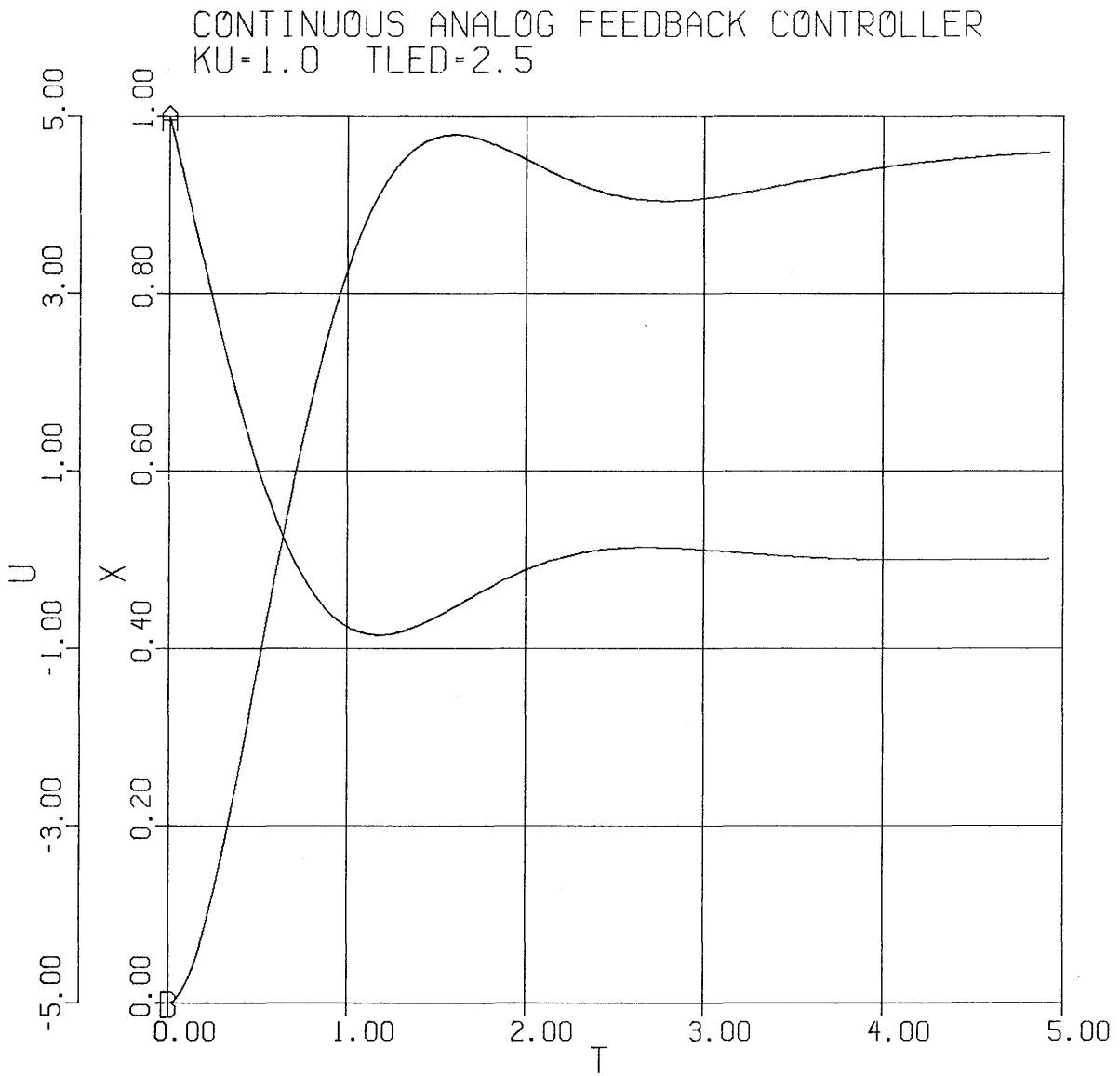


Figure A11-7. Comparison using Continuous Lead/Lag Feedback Control

12. ASPIRIN DOSAGE EVALUATION

This model follows mathematically the aspirin concentration level in the blood stream of a person taking variable doses at various preset times. A similar program would be used to determine the concentration of any drug which had an exponential decay. The example was chosen to show how discontinuities can be introduced into the simulation state variables without having to recompute the values of the output of integrators, a matter that has to be left to the integration routine. In actual fact there are no true discontinuities in the physical world and one should really consider the dynamics of the ingestion process - the passage through the stomach wall or, if injected, the velocity of the hypodermic piston. These effects, however, would normally be approximated in most simulations as discontinuities if their time of action is short compared with the overall period of interest.

The basic equation in the model is the exponential decay given by

$$\frac{dC}{dt} = -R C$$

which says that the rate of change (decay) of concentration of (C) is proportional to the concentration. The constant of proportionality is a rate R.

Since the concentration can change suddenly as doses are taken, we need to add a term indicating the total dose. As an integral equation, the concentration can now be expressed as

$$C = \int -R C dt + \sum D_i$$

In the simulation model code this is written

$$\text{BLOOD} = \text{INTEG}(-\text{RATE}*\text{BLOOD}, 0.0) + \text{TOTAL}$$

where BLOOD is the concentration in grains and TOTAL contains the sum of the doses up until the current time.

The listing of the model definition section is shown in Figure A12-1 which defines two arrays, TDOS for the time of the dose and DOSE for the actual dose at the corresponding time. An integer index INDX is used to advance through the arrays. The arrays are preset so that a larger dose of the five tablets (25 grains) is given initially and only two hours are between the first and second doses. Eight hours lie between the fourth and fifth doses and the last effective dose is at thirty hours.

In the INITIAL section INDX is started at one to access the initial dose time and TOTAL, effectively a state variable, is set to zero. In the DERIVATIVE section the BLOOD and URINE levels are calculated, the URINE being the total amount excreted. The URINE rate is the opposite of the rate for BLOOD which means that what is removed from the bloodstream appears directly in the urine.

The dose time and dose accumulation is performed within a PROCEDURAL . . . END block which is treated as a whole. Within the PROCEDURAL block, the current dose time TDOS (INDX) is tested against the independent variable time (T). If it's not time for a dose the rest of the block is bypassed. If the dose time is equalled or exceeded, the dose is added to total by

$$\text{TOTAL} = \text{TOTAL} + \text{DOSE}(\text{INDX})$$

and then the index is incremented by one, ready for the next dose.

In writing this type of simulation model there is an inclination to change the state variable itself i.e.,

```
BLOOD = INTEG(-RATE*BLOOD, 0.0)
```

```
BLOOD = BLOOD + DOSE(INDX)
```

This operation is illegal since the value in the state variable BLOOD is only a temporary copy of the actual state variable which is saved internally by the integration routine. Note if we convert the BLOOD integration to the expression form:

```
BLOOD = -INTEG (RATE*BLOOD, 0.0)
```

BLOOD is no longer a state variable, it is now just the negative of the actual state variable (the output of the INTEG operator) which will be given a generated name Z0nnnn.

When we add TOTAL to the integration a similar transformation occurs so that BLOOD is no longer a state variable but now we can manipulate TOTAL in such a way that the answer comes out right.

The run-time output stream is shown in Figure A12-2. After the START a column PRINT is obtained to list numeric values and then a PLOT, the output of which is shown in Figures A12-3 and A12-4. The first figure shows the format for the strip plots (STRPLT = .T.) and the second for the standard line plots (CALPLT = .T.) where all curves are superimposed. The actual plot line reads

```
PLOT TOTAL, 'TAG' = '+URINE', URINE, 'SAME', 'OVER', BLOOD
```

The tag string is used to add the extra label on the TOTAL axis, 'SAME' ensures identical scales and 'OVER' suppresses the now redundant axis for URINE.

The RATE parameter is changed to 0.28 and a second run made, followed by a PLOT. Only the strip plot is used in Figure A12-5 which shows the extra strip generated when 'OVER' was eliminated from the plot line. From the plot it can be seen that the higher decay rate has reduced the average blood level concentration.

The discontinuities introduced into TOTAL violate the restrictions on the state variables placed by most integration algorithms, i.e., the derivatives shall be continuous and differentiable. In practice, fixed step size algorithms step over these discontinuities very well with only minor differences in the calculated solutions when the step size changes. In the code we make no attempt to synchronize the discontinuity with the integration step so it can occur at any of the derivative evaluations that make up the step. This means that this particular step (which contains the discontinuity) will be in error but the answers will only be slipped a fraction of a step length. It is this requirement that dictates the integration step size. From the decay rate of the aspirin (0.14 or 0.28) step sizes of two or three hours would do quite well in integrating the differential equations. However, uncertainty in dose times of this much is too large an error. In the model, the step size is set (via MAXT) to 0.05 or 3 minutes to reduce the uncertainty which now seems tolerable in light of what we are trying to simulate.

PROGRAM ASPIRIN DOSAGE TEST

```

  "-----FOLLOWS CONCENTRATION OF DRUG IN BODY "
  " GIVEN A DOSAGE HISTORY. USES EXPONENTIAL ELIMINATION RATE "
  " WITH FIXED TIME CONSTANT "

  "-----DEFINE TYPES AND ARRAY DIMENSIONS"
  INTEGER      INDX      , INDXMX
  ARRAY        DOSE(9) , TDOS(9)
  "-----DEFINE PRESET CONSTANT VALUES"
  CONSTANT     RATE = 0.14 , INDXMX = 9      ...
              TSTOP = 49.0
  CONSTANT     DOSE = 25.0 , 15.0 , 15.0    ...
              , 15.0 , 15.0 , 15.0      ...
              , 15.0 , 15.0 , 0.0
  CONSTANT     TDOS = 0.0 , 2.0 , 6.0      ...
              , 10.0 , 18.0 , 22.0      ...
              , 26.0 , 30.0 , 99.0

  "-----DEFINE COMMUNICATION INTERVAL AND INTEGRATION STEP "
  CINTERVAL    CINT = 0.5
  MAXTERVAL    MAXT = 0.05
  NSTEPS       NSTP = 1

INITIAL
  "-----START WITH FIRST DOSE,NONE PRESENT"
  "          AT BEGINNING"
  INDX      = 1
  TOTAL     = 0.0
END $* OF INITIAL*
DYNAMIC
DERIVATIVE
  "-----AMOUNT LEFT IN BLOOD STREAM"
  BLOOD     = INTEG(-RATE*BLOOD, 0.0) + TOTAL
  "-----TOTAL AMOUNT EXCRETED AS URINE"
  URINE     = INTEG( RATE*BLOOD, 0.0)
  "-----TEST FOR DOSE,BUMP TOTAL"
  PROCEDURAL(TOTAL,INDX = DOSE, TDOS)
  "-----IF NOT TIME FOR DOSE"
  IF(T.LT.TDOS(INDX)) GO TO L1
  "-----ADD NEW DOSE TO TOTAL"
  TOTAL     = TOTAL + DOSE(INDX)
  "-----GET SET FOR NEXT DOSE"
  INDX      = INDX + 1
L1..CONTINUE
END $* OF PROCEDURAL*

END $* OF DERIVATIVE*
  "-----STOP WHEN GIVEN LAST DOSE"
  TERMT(INDX.GT.INDXMX .OR. T.GE.TSTOP)
END $* OF DYNAMIC*
END $* OF PROGRAM*
```

Figure A12-1. Listing of Model Definition for Aspirin Dosage Evaluation

```

SET TITLE = "ASPIRIN DOSAGE TEST"
S TCWPRN=72 $* FORCE 3 COLUMN OUTPUT WIDTH *
S PRNPLT=.F.,CALPLT=.T.,STRPLT=.T.,GRDSPL=.T.
PREPAR T, BLOOD, URINE, TOTAL, INDX
START
PRINT "NCIPRN"=5, "ALL"
    
```

LINE	T	BLOOD	URINE	TOTAL	INDX
0	0.	25.000000	0.	25.000000	2
5	2.5000000	31.619426	8.3805736	40.000000	3
10	5.0000000	22.281833	17.718167	40.000000	3
15	7.5000000	27.874691	27.125309	55.000000	4
20	10.000000	19.642963	35.357037	55.000000	4
25	12.500000	24.424815	45.575185	70.000000	5
30	15.000000	17.211877	52.788123	70.000000	5
35	17.500000	12.129004	57.870996	70.000000	5
40	20.000000	19.897147	65.102853	85.000000	6
45	22.500000	28.023507	71.976493	100.00000	7
50	25.000000	19.747832	80.252168	100.00000	7
55	27.500000	26.089011	88.910989	115.00000	8
60	30.000000	18.384615	96.615385	115.00000	8
65	32.500000	23.538073	106.46193	130.00000	9
70	35.000000	16.586999	113.41300	130.00000	9
75	37.500000	11.688661	118.31134	130.00000	9
80	40.000000	8.2368602	121.76314	130.00000	9
85	42.500000	5.8044173	124.19558	130.00000	9
90	45.000000	4.0903037	125.90970	130.00000	9
95	47.500000	2.8823883	127.11761	130.00000	9

```

PLOT TOTAL, "TAG"="+ URINE", URINE, "SAME", "OVER", BLOOD
DISPLY RATE, TDOS, DOSE
    
```

RATE	TDOS	DOSE
0.14000000	0.	2.00000000
6.00000000	10.0000000	18.0000000
22.0000000	26.0000000	30.0000000
99.0000000	DOSE 25.0000000	15.0000000
15.0000000	15.0000000	15.0000000
15.0000000	15.0000000	15.0000000
0.		

```

SET RATE=0.28, TITLE(5)="RATE = 0.28"
    
```

```

START
PLOT TOTAL, URINE, "SAME", BLOOD
DISPLY RATE, TDOS, DOSE
    
```

RATE	TDOS	DOSE
0.28000000	0.	2.00000000
6.00000000	10.0000000	18.0000000
22.0000000	26.0000000	30.0000000
99.0000000	DOSE 25.0000000	15.0000000
15.0000000	15.0000000	15.0000000
15.0000000	15.0000000	15.0000000
0.		

```

STOP
    
```

Figure A12-2. Output Stream from Aspirin Dosage Evaluation Study

ASPIRIN DOSAGE TEST

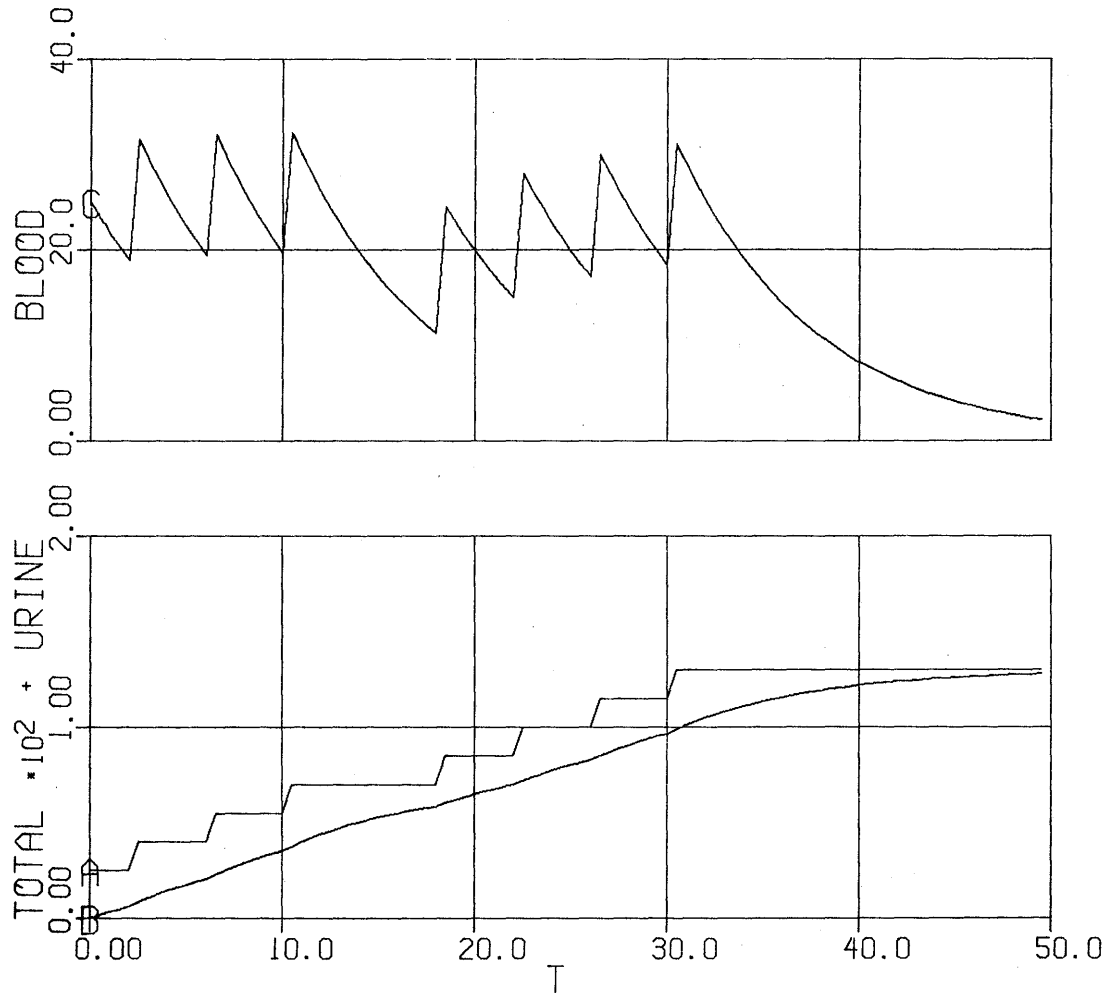


Figure A12-3. Strip Plot of Blood Concentration, Total Dosage and Urine Elimination, Rate in 0.14

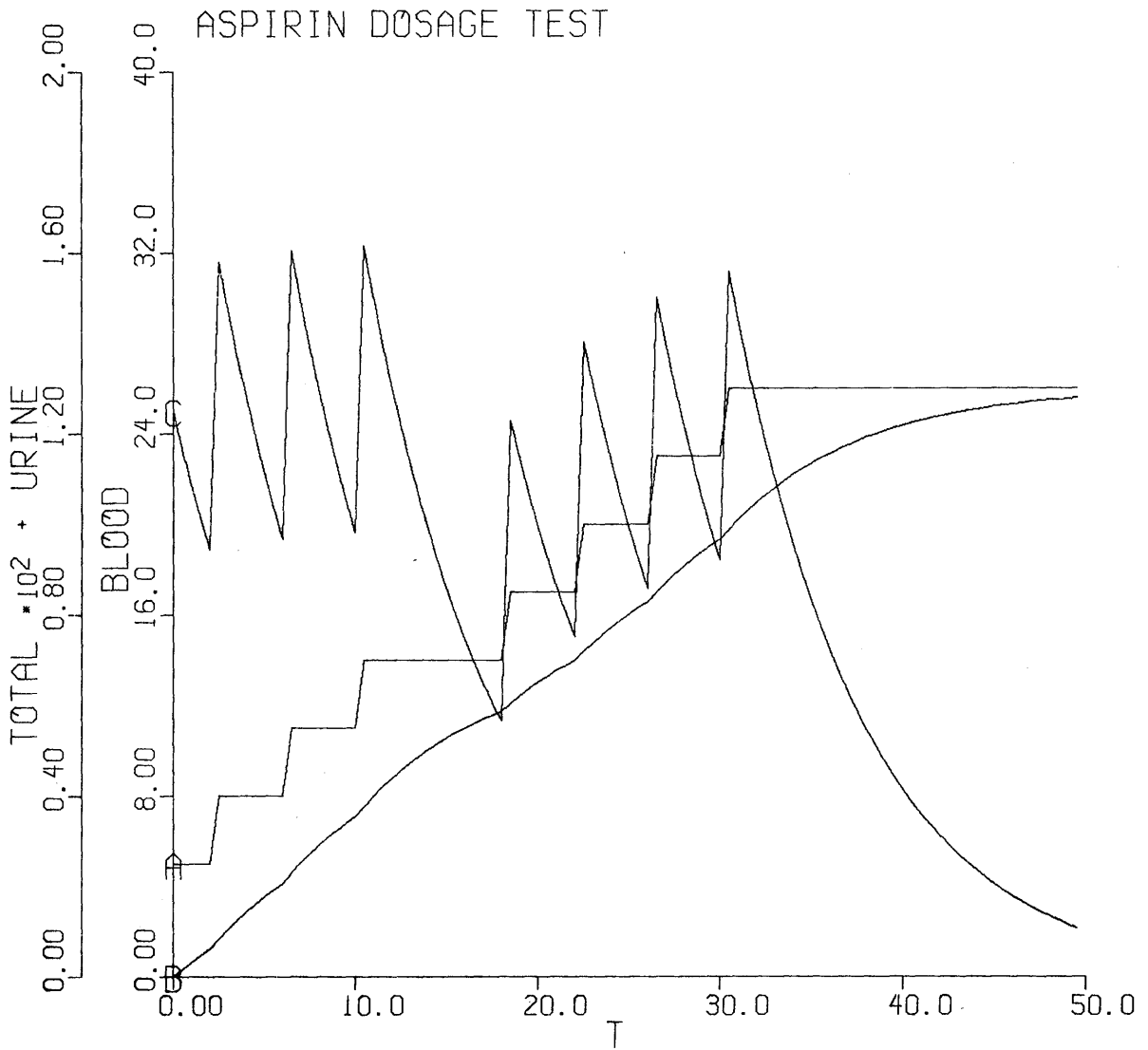


Figure A12-4. Line Plot of Blood Concentration, Total Dosage and Urine Elimination

ASPIRIN DOSAGE TEST
RATE = 0.28

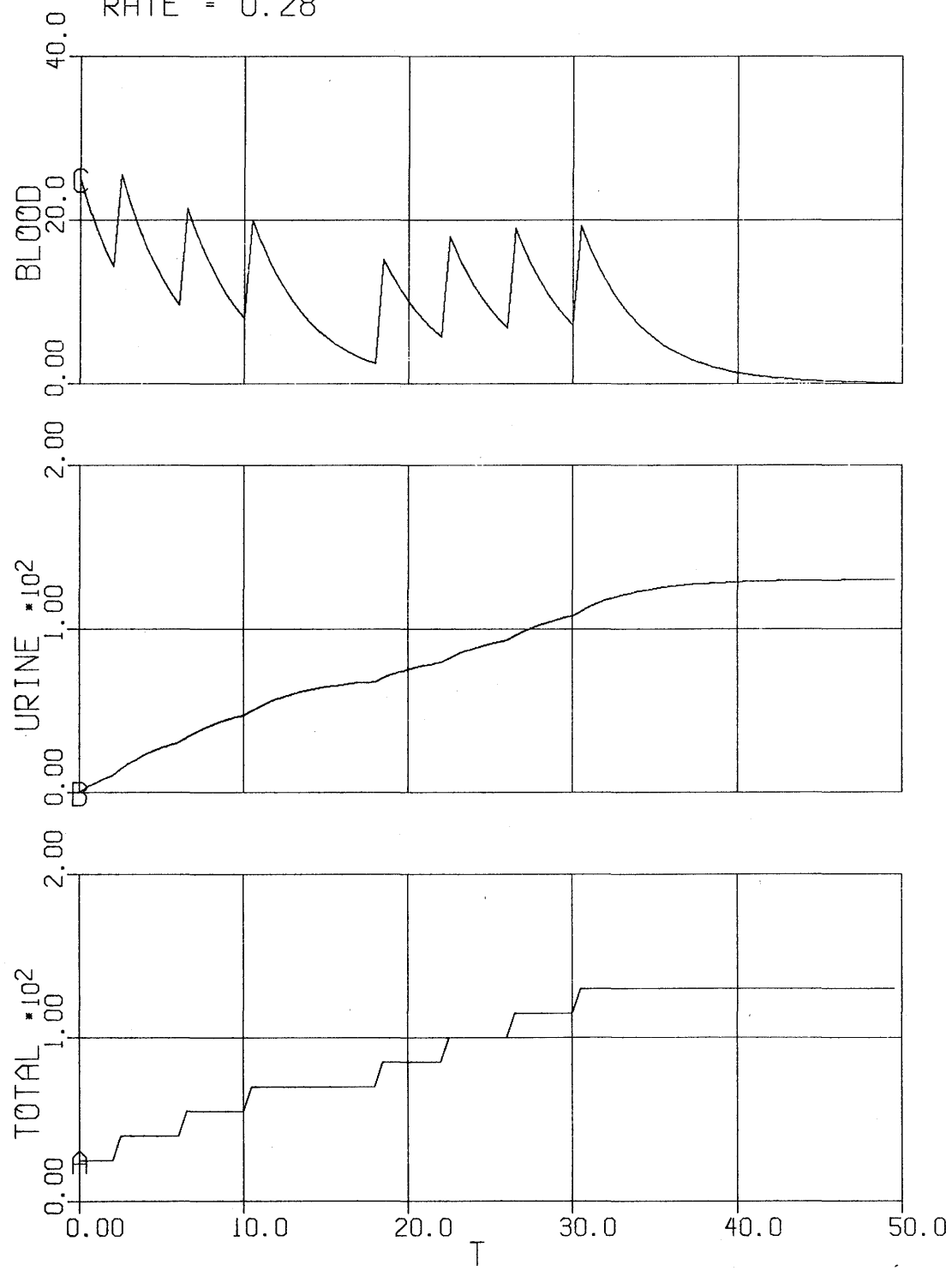


Figure A12-5. Strip Plot of Blood Concentration, Urine Elimination and Total Dosage, Rate in 0.28

APPENDIX B

GENERAL PURPOSE UTILITY SUBROUTINES

A number of general purpose subroutines have been developed and included in the system library.

1. AGET (name, a), APUT (name, a)

Obtain the values from or put values into a variable array in either the user or system dictionary. Useful in a separate FORTRAN subroutine to obtain or return isolated values not passed through the calling sequence. Necessary in order to access values from the system dictionary. Arguments are:

name - Hollerith representation of dictionary name to be accessed, i.e. Direct Hollerith constant (3HDIS) or symbol preset to Hollerith constant.

a - array of size equal to or greater than that of 'name' - may be a scalar of dimension 1.

For AGET all elements of 'name' are returned in successive locations of 'a'. APUT works in the opposite direction and fills all positions of the array 'name' with the contents of successive locations of 'a'. Standard form of use would be

```
CALL AGET (6HNPXPPL, NPX)
CALL APUT(5HTITLE, NEWTTL)
```

where NEWTTL is dimensioned to contain at least 120 characters (12 words on CDC 6000/7000, 20 on UNIVAC 1100 and 30 on 32 bit hex machines IBM 360/370, SEL etc.). See RGET, IGET and VPUT for changing individual elements of an array.

NOTE: These functions require a full dictionary search and if placed in a loop executed every calculation interval will use an excessive amount of computer time.

2. BLDDCT (nHname, name, type, size)

NITBLD (length)

Build dictionary allows variables in other FORTRAN subroutines and COMMON blocks to be added to the ACSL dictionary. Since labelled COMMON block locations are defined at load or link-edit time, this operation must be performed once at the start of each simulation study. The typical requirement is to add variables (XVEL and IPNT are used in the example) that are in external common blocks (/USER/is used) that communicate between FORTRAN subroutines external to the ACSL simulation model definition. The actual number of variables and common blocks depends on the particular situation and can be large.

In order to add the example names to the dictionary, write a subroutine so

```
SUBROUTINE ADDNMS
COMMON/USER/IPNT, XVEL
COMMON/ZZDCT/DUMMY(1000)
CALL NIT BLD(1000)
CALL BLD DCT(4HIPNT, IPNT, 2, 1)
CALL BLD DCT(4HXVEL, XVEL, 1, 1)
RETURN
```

END

and call this from the pre-initial section of the ACSL model definition code i.e.

```
PROGRAM TO EXTEND DICTIONARY
```

```
CALL ADDNMS $ 'ADD NAMES TO DICTIONARY'
```

```
INITIAL
```

```
...
```

```
END $ 'OF PROGRAM'
```

Alternatively all the code can be placed in the main program and then use the P option on the translator to indicate it is user supplied.

In the ADDNMS subroutine, the external COMMON blocks can be referred to (no COMMON blocks can be included in the ACSL model definition). The dictionary COMMON block /ZZDCT/ must be included and extended beyond the length established by the ACSL translator, which normally will size the dictionary to just fit all the names in the model definition (2 words per name in machines with six or more characters per word, 3 words per name for machines with 4 or 5 characters/word). In the example, the dictionary COMMON block is extended to a thousand words, which length is then communicated to the extension program via the NITBLD call. This must be present since the BLDDCT subroutine has no idea how much space is available at the end of the standard ACSL dictionary.

The actual calls to BLDDCT pass a Hollerith version of the new name, the name itself which really corresponds to the address in the COMMON block, and an integer indicating type and an integer indicating size. The types are one for REAL, two for INTEGER and three for LOGICAL. Size must be the array size if an array (product of dimensions if more than one) or one if 'name' is a scalar.

The action of BLDDCT is to search through the dictionary to the end and then add the entry corresponding to 'name', moving the dictionary terminator one block. An error is reported if 'name' is already in the dictionary or if the end of the dictionary would have to be extended beyond the length established by the call to NITBLD.

Once the name has been added to the dictionary, all run-time references can be made just like any other ACSL variable with the data being transferred to and from the external COMMON blocks

```
OUTPUT XVEL
```

```
DISPLY IPNT
```

```
...
```

All the ACSL run-time commands can be used to PLOT, PRINT, SET these variables rather than having to move everything into the main ACSL common block /ZZCOM/.

3. DEBUG

A call to this routine will produce a debug list of all variables, excluding arrays greater than MALPRN (maximum array limit for print), on both PRN and DIS units. Described in Section 7 is the technique of setting NDBUG to a positive integer whereby a debug list is produced at the end of every derivative evaluation. While useful as a checkout tool, with large programs this action can produce an over-whelming amount of output. Selective output can now be obtained by

```
IF(logical condition) CALL DEBUG
```

included in the DERIVATIVE section. Including the statement

```
CALL DEBUG
```

in the DYNAMIC section produces the entire list at each communication interval and is synonymous with asking for the OUTPUT of all variables. Including

IF(DUMP)CALL DEBUG

in the TERMINAL section is a useful artifice since all final values are displayed as well as the initial conditions for that run.

4. IGET(nHname,i) RGET(nHname,i)

Obtain the value of a variable (integer = IGET, real = RGET) in the user or system dictionary. For use primarily for selecting elements of arrays since AGET and APUT should be used for scalars. Arguments are:

name - Hollerith representation of dictionary name to be accessed i.e. symbol preset to Hollerith constant or direct Hollerith constant (3HPRN)

i - integer constant or variable denoting element of array

Standard form of use would be

WORD = IGET(5HTITLE, 4)

NOTE: These functions require a full dictionary search and if placed in a loop executed every calculation interval will use an excessive amount of computer time.

5. INTEG

In order to provide flexibility in trying new and improved integration algorithms, it is possible to incorporate a user written routine via this subroutine (INTEG). Setting IALG=7 will transfer control to this routine at the beginning of every integration step. In order to write an effective INTEG routine, familiarity with the ACSL run-time routines ZZINTG, ZZRKIN, ZMZVM, ZZNITS and ZZNITA is essential.

6. LISTD (file)

Provides a listing of the user dictionary and current variable values along with any explanation of variables supplied on the named file. Used mainly for reports, it requires preparing a dictionary with variable name and definition. The argument 'file' is an integer constant or variable defining a file containing the definitions or card images.

col 1-9 variable name, left justified

col 10 continuation indicator in sequence 0,1,2,3 etc.

col 11-80 definition

Standard form of invocation would be in the INITIAL or TERMINAL section on a switch

IF(LIST)CALL LISTD(5)

LIST = .FALSE.

...

Logical unit five is normally the input file so the dictionary would follow immediately behind the START card. The dictionary is terminated with a blank name field. When the definition must be continued beyond column eighty, continuations can be used which consist of a non-blank character in column ten. The name field in this case is ignored and just the extra definition is listed out. For convenience in ordering the initial dictionary repeat the name on each continuation card, numbering the cards in column ten 0, 1, 2, 3 etc. Now a standard sort on columns one through ten will produce an alphabetical order with continuation cards in their correct place. Note that zero is used in column ten to start a continuation sequence and acts just like a blank.

The reason for using a zero is that in some computer systems blanks will collate after numbers rather than before.

When used with logical unit five, the dictionary definitions appear on the run-time drive file immediately after the START and the following card images will be read until a blank or end-of-file.

7. LOG

This routine can force a data recording and output list from within the ACSL model definition or from any external FORTRAN subroutine.

CALL LOG

will reset the NCIOUT count of the OUTPUT statement, forcing a value listing. All the variables on the PREPAR list are recorded for later PRINTing or PLOTting.

8. RGET (nHname, i)

Real get - - see IGET

9. SET (value, name, times)

A useful subroutine that is provided primarily to initialize arrays and set all elements to a given value. Standard form of use is

CALL SET (v, x, n)

where the value of the expression v is placed into n locations of array x. i.e.

CALL SET(0.0, ARR(5), 3)

zeros the fifth, sixth and seventh elements of array ARR. No check is made to see whether these elements actually exist.

10. TIMER

Program execution time can be estimated by use of this subroutine placed in the INITIAL, DYNAMIC or TERMINAL sections. The derivative evaluation routine is called one hundred times and the average central processing time used per evaluation reported. It must *not* be called from the DERIVATIVE section since it will then be activated recursively and the program will abort with an error message (TIMER CALLED FROM DERIVATIVE SECTION)

11. VPUT (name, i, value)

Place a value in a named variable in either the user or system dictionary. Arguments are:

name - a Hollerith constant or variable defining a name in either the user or system dictionary

i - the element number in the array

value - a constant, variable or expression that corresponds in type to the named variable

Standard form of invocation would be

CALL VPUT(5HARRAY, 4, DATE)

NOTE: This subroutine requires a full dictionary search and if placed inadvertently in a loop will use an excessive amount of computer time.

12. WEM (nH message, nchar)

Error messages may be written using the ACSL standard output interface by this subroutine. Standard form of use is

CALL WEM(nH message of length n characters, n)

which will write an error message on both PRN and DIS logical units if different.

13. WRITG

Write integration intermediate data. Primary use is in debugging the variable step integration routines (IALG = 1 or 2). The state history stacks and error tables are listed. Should normally be placed at the end of the DERIVATIVE section. Data is written on the logical unit number contained in system variable PRN. Standard form of invocation is:

CALL WRITG

14. XFERB

A transfer block routine is provided for moving arrays from one place to another. Useful for initialization instead of forming DO-loops. Standard forms of call are

CALL XFERB(x, n, y)

CALL XFERB(y = x, n)

which takes n elements from array x and moves them to array y. No check is made to see whether these elements are contained in the arrays.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

ACSL SYSTEM SYMBOLS

Certain system constants can be changed if not used in the simulation model to allow greater flexibility. These names are generated by the first three characters to describe the action and the last three characters to describe the processor.

- PPL refers to printer plots
- CPL refers to line or Calcomp plots
- SPL refers to strip plots
- PLT refers to plotting in general
- ITG refers to integration
- PRN refers to printed outputs

These variables can all be changed by SET statements. The value set must agree in type - i.e., integer to integer, logical to logical, real or integer into real. The nominal value preset is given below the symbol.

1. Refers to Plots in General

CALPLT Logical: Plot on the line plot device selected at load or link-edit time.
(.FALSE.)

DEFPLT Logical: Defer plots. The current plot is not printed so that subsequent plots can be built up on the same picture. This feature has been used to plot trajectories of both missile and target on the same grid, i.e.

```
SET DEFPLT = .TRUE.  
PLOT 'XAXIS' = RM1, RM3  
SET DEFPLT = .FALSE.  
PLOT 'XAXIS' = RT1, RT3
```

The first plots the missile trajectories with the x-axis, RM1. Then the deferred plot restriction is lifted and the second plot plots target trajectory RT1 versus RT2 and produces the output. Scale factors should be set so that the same scale factors are used for both target and missile trajectories. A useful feature here is the use of a symbolic name to imply the contents of the location denoted by the symbol.

Line or Calcomp plots work similarly with the exception that the axes are not drawn while DEFPLT is .TRUE.. Thus it is important to be sure that the scales are the same since no indication will be given if they are not: The reason is the limited size of the plot image area so that scales cannot continue to be drawn without running out of room.

FTSPLT Logical: Flyback trace suppressed on plots. If a number of parametric runs have been made - by cycling between the INITIAL and TERMINAL sections - they can be plotted and if this variable is TRUE, plotting is suppressed and the symbol is incremented on flyback. The flyback is

Cont.

determined from the first variable on the PREPARE list. When the difference between successive values becomes negative it is assumed that a new run has started. Normally, the independent variable is made the first variable on the PREPARE list.

- NPPPLT
(3) Integer: Number of plots per page used when invoking the PLOT 'ALL' command and controls the number of variables plotted per drawing.
- PRNPLT
(.TRUE.) Logical: Plot on printer
- STRPLT
(.FALSE.) Logical: Plot on line device in strip chart form. Normally one variable per axis set stacked in reverse order to that on the command line.

2. Refers to Printer Plots

- CGDPPL
(47) Integer: Character for the grid in the printer plot. Set to be a period. Note, it can only be changed by knowing the character value as an integer, not by quoting.
- NGXPPL
(20) Integer: Number of points between grid lines in the x-direction for printer plots. Nominal setting sets a grid of periods every twenty characters.
- NGYPPL
(10) Integer: Same as NGXPPL in y direction.
- NPCPPL
(1) Integer: Number of points per character plotted on the printer: This feature can be used for placing time ticks on a phase plane plot, i.e.
- ```
SET DEFPLT = .TRUE.
PLOT 'XAXIS' = X, Y
SET DEFPLT = .FALSE., NPCPPL = 10
PLOT 'XAXIS' = X, Y, 'CHAR' = '*'
```
- The first plots Y against X using values recorded every communication interval but the output is deferred. The plot frequency is changed and the second plot is plotted over the first, with a different character so flagging every tenth point.
- NPXPPL  
(100) Integer: Gives number of points in the x direction for printer plots. For square plots make this six tenths of the y direction points.
- NPYPPL  
(100) Integer: Gives number of points in the y direction for printer plots. For narrow terminals this will be reduced to fifty.

## 3. Refers to Line or Calcomp Plots

- GRDCPL  
(.FALSE.) Logical: When set TRUE, draw grid lines from each tick mark on the axes.
- LINCPL  
(.TRUE.) Logical: Draw lines between points for the line plots. The lines between points can be suppressed by making this variable .FALSE.. Note if both SYMCPL and LINCPL are false, nothing will be plotted.

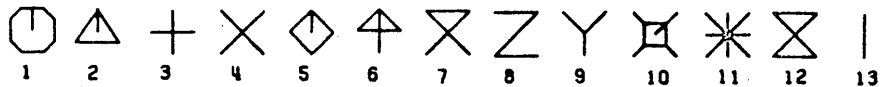
Cont.

NPCCPL (10) Integer: Number of points per character on the line plots. In this case a symbol is written every ten points plotted. Used for time ticks on phase plane plots.

PSFCPL (1.0) Real: Plot scale factor for line plots. The overall size of the plot, including axes and lettering can be made smaller (or larger) by changing this factor.

SATCPL (.FALSE.) Logical: Suppress axis text. Allows the axes and tick marks to be drawn but suppresses all numbers and labels for speed in repetitive plots.

SYMCPL (.FALSE.) Logical: Plot symbols on the curve. Characters will be centered over the point and will correspond to the normal FORTRAN character set. Special symbols can be obtained as follows:



which correspond to asking for special characters such as 'CHAR'='%'. The actual characters available are installation dependent so see local addendum. If symbols are plotted for every point, they are usually over-written and confused (so see item NPCCPL).

TBRCPL (9600) Integer: Baud rate of channel to line plot device. Normally needed for Tektronix plotters. Note must be set before the first PLOT command when the plot device is initialized.

TTLCP (TRUE.) Logical: Title on line plots. The full 120 character title array is written at the top of each plot in three lines of forty characters each. The last line overlaps into the plot area. Since most of TITLE is not often used, trailing blanks in TITLE may result in fewer than three lines being written.

XCICPL (0.0) Real: X-axis cross position. Position on x-axis (in inches) where last y-axis is positioned.

XINCPL (5.0) Real: X-axis length in inches for line plots.

XTICPL (1.0) Real: X-axis tick increment in inches for line plots.

YCICPL (0.0) Real: Y cross inches for line plots. This dimension is where the x-axis is drawn on the page. Normally at the bottom, it can be raised up to ten inches. Negative values will send the plotter into limit.

YINCPL (5.0) Real: Y-axis length in inches for line plots.

YTICPL (1.0) Real: Y-axis tick increment in inches for line plots.

#### 4. Refers to Strip Plots

|                     |                                                                                                                                     |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| GRDSPL<br>(.FALSE.) | Logical: Draw grid lines from each axis tick on strip chart plots.                                                                  |
| LINSPL<br>(.TRUE.)  | Logical: Draw lines between points on strip chart plots.                                                                            |
| NPCSPL<br>(10)      | Integer: Number of points per character or symbol on strip chart plots.                                                             |
| PSFSPL<br>(1.0)     | Real: Plot scale factor for the strip chart plots.                                                                                  |
| SATSPL<br>(.FALSE.) | Logical: Suppress axis text. Allows the axes and tick marks to be drawn but suppresses all numbers and labels for repetitive plots. |
| SYMSPL<br>(.FALSE.) | Logical: Plot symbols on the curve. The symbols will correspond to those given for SYMCPL.                                          |
| TTLSPL<br>(.TRUE.)  | Logical: Draw a title over the strip plots. Three lines of forty characters each.                                                   |
| XCISPL<br>(0.0)     | Real: X-axis cross position. Position on x-axis (in inches) where y-axis is placed.                                                 |
| XINSPL<br>(5.0)     | Real: X-axis length in inches for strip chart plots.                                                                                |
| XTISPL<br>(1.0)     | Real: X-axis tick increment in inches for strip chart plots.                                                                        |
| YASSPL<br>(0.5)     | Real: Y-axis separation between successive axes stacked vertically.                                                                 |
| YCISPL<br>(1.0)     | Real: Y-axis cross position. Position on y-axis (in inches) where the x-axis is placed.                                             |
| YINSPL<br>(2.0)     | Real: Y-axis length in inches for strip chart plots.                                                                                |
| YTISPL<br>(1.0)     | Real: Y-axis tick increment in inches for strip chart plots.                                                                        |

#### 5. Refers to Print Data

|                     |                                                                                                                                                                                                                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HVDPRN<br>(.FALSE.) | Logical: High volume data to display unit. Ensures all data written on PRN unit is also copied on the DIS unit if different.                                                                                                                                                                                                      |
| MALPRN<br>(10)      | Integer: Maximum array length. For debug output all arrays were normally listed. In order to control the amount of output, this variable was added that suppresses listing of the contents of any array longer than this value. The last element only will be listed to show that it is present and to indicate the array length. |
| TCWPRN<br>(132)     | Integer: Terminal character width. Controls the line width of any data written on the display (DIS) logical unit. For basically interactive machines such as DEC VAX/11 the default is changed to 72.                                                                                                                             |

## 6. Integration Control

- CIOITG**  
(. . .) Integer: Current integration order. Calculated by the program it may be OUTPUT or PREPARED so that the integration order may be followed when using one of the variable order, variable step integration routines (IALG = 1 or 2).
- CSSITG**  
(. . .) Real: Current step size. Calculated by the program it may be OUTPUT or PREPARED so that the actual integration step size may be followed as explained previously (CIOITG).
- MXOITG**  
(6) Integer: Maximum order. The maximum order of the integration algorithm may be specified between 1 and 6 when using the variable order, variable step integration routines.
- NRWITG**  
(.FALSE.) Logical: No rewind. When this flag is true the data file containing the value of all the variables on the PREPARE list is not rewound immediately after a START. Data from sequential runs is then accumulated rather than being written on top of the previous data, thereby erasing it.
- TSMITG**  
(.FALSE.) Logical: Two sided matrix evaluation. The stiff integration algorithm (IALG = 2) needs to evaluate the linearized state transition matrix. If the state equation is  $dx/dt = F(x)$ , then two sided is obtained from  $F(x+dx)$  and  $F(x-dx)$ ; single sided from  $F(x)$  and  $F(x-dx)$ . While two sided is more accurate than single sided, it requires twice as long to evaluate the complete matrix.
- WESITG**  
(.TRUE.) Logical: Write error summary. At the end of a run using the variable order, variable step integration routines (IALG = 1 or 2) the option exists to list all the states along with the count of the number of times each state controlled the integration stepsize. Normally true, this data may be suppressed by setting WESITG false.

## 7. General

- CMD**  
(5) Integer: Logical unit from which run-time commands are read. May only be five, six or nine on Control Data computers. See local installation guide.
- DIS**  
(6) Integer: Logical unit on which display data is written out on. Output from DISPLY, RANGE and OUTPUT commands. Allowed values depend on machine and installation.
- NDEBUG**  
(0) Number of derivative evaluations that will have DEBUG output tied to them.
- PLT**  
(9) Integer: Logical unit for line plot output (when CALPLT is .TRUE.). Device and installation dependent - see local installation guide.
- PRN**  
(6) Integer: Logical unit on which high volume data is written out. All data written on unit DIS is echoed on unit PRN if different. In addition printer plots and PRINT command output are only written on this file. Allowed values depend on machine and installation.

Cont.

RRR  
(8)

Integer: Logical unit on which intermediate data is written out. Allowed values depend on machine and installation.

TITLE  
(blank)

Hollerith: Up to 120 characters may be set into this array which is listed at the top of each page. Hollerith data must be quoted.

# APPENDIX D

## QUICK REFERENCE GUIDE FOR ACSL OPERATORS

Tables D-1, D-2 and D-3 list the operators available in the model definition section, the run-time executive commands and the system constants.

Throughout the model definition operators, X is a real expression, J is an integer expression. An expression can be a signed or unsigned constant, a variable or array name, or any combination formed into a legal expression.

**TABLE D-1. Summary of ACSL Model Definition Statements**

| Statement                 | Explanation                                                       |
|---------------------------|-------------------------------------------------------------------|
| ABS(x)                    | Absolute value                                                    |
| ACOS(x)                   | Arc-cosine; result in radians                                     |
| AINT(x)                   | Integer part of real expression x                                 |
| ALGORITHM IALG = 5        | Define integration algorithm                                      |
| ALOG(x)                   | Natural logarithm                                                 |
| ALOG10(x)                 | Logarithm to base ten                                             |
| AMAX0(j1, j2 . . .)       | Real maximum of integer expressions, j <sub>i</sub>               |
| AMAX1(x1, x2 . . .)       | Maximum of given string of expressions, x <sub>i</sub>            |
| AMIN0(j1, j2 . . .)       | Real minimum of integer expressions, j <sub>i</sub>               |
| AMIN1(x1, x2 . . .)       | Minimum of given string of expressions, x <sub>i</sub>            |
| AMOD(x1, x2)              | Remainder when x1 is divided by x2                                |
| ARRAY v(1, 2, 3), . . .   | Specifies up to three dimensions                                  |
| ASIN(x)                   | Arc-sin; result in radians                                        |
| ASSIGN k TO m             | Used before a GO TO m branch                                      |
| ATAN(x)                   | Arc-tangent-result in radians                                     |
| ATAN2(y, x)               | Angle in radians between x-axis and point (x, y)                  |
| BCKLSH(ic, dl, x)         | Backlash or hysteresis                                            |
| BOUND (ll, ul, x)         | Limit expression x to be between lower and upper limits           |
| CALL name                 | Invoke subroutine                                                 |
| CINTERVAL CINT = 0.1      | Define name and value for communication interval                  |
| CMPXPL(p, q, x, ic1, ic2) | $y = \frac{1}{ps^2 + qs + 1}$ ; $\dot{y}(0) = ic1$ ; $y(0) = ic2$ |

Cont.



TABLE D-1. Summary of ACSL Model Definition Statements—Continued

| Statement                            | Explanation                                                                               |
|--------------------------------------|-------------------------------------------------------------------------------------------|
| COMMENT                              | Enclose statement in quote for comment                                                    |
| CONSTANT d = a                       | Set constant d to value a                                                                 |
| CONTINUE                             | Do nothing - usually labelled                                                             |
| COS(x)                               | Cosine of expression x in radians                                                         |
| DBG''                                | Translator debug feature                                                                  |
| DBLINT(x, xd = ic, dd, dic, ll, ul ) | Double integrator with limit                                                              |
| DEAD(ll, ul , x)                     |          |
| DELAY(x, ic, tdl, nmX)               | Delay expression x through fixed time tdl                                                 |
| DERIVATIVE                           | Begins block evaluating state variable derivatives                                        |
| DERIVT(ic, x, T)                     | Differentiate expression x: <b>WARNING</b> should not be used unless absolutely necessary |
| DIM(x1, x2)                          | Difference (x1 - x2)if positive, else zero                                                |
| DISCRETE                             | Begins block representing a DISCRETE event                                                |
| DO 1 j = 1, n                        | Start of DO-loop                                                                          |
| DYNAMIC                              | Begins section entered every communication interval                                       |
| END                                  | Must complete each section, block or PROCEDURAL                                           |
| EQUIVALENCE(mv, v)                   | Equivalence names to same location                                                        |
| ERRTAG ERR                           | Defines name for variable to indicate integration error                                   |
| EXP(x)                               | Natural exponent of expression x                                                          |
| EXPF(ic, τ, on)                      | Rise and fall between 0.0 and 1.0 on exponential with time constant                       |
| FCNSW(p, x1,x2, x3)                  | Function switch                                                                           |
| FORMAT(. . .)                        | Format description for READ, WRITE or PRINT statements                                    |
| GAUSS(m, s)                          | Normally distributed random variable, given mean, m, and standard deviations.             |
| GAUSI(j)                             | Initialize random number seed                                                             |
| GO TO 1                              | Transfer control to statement labelled 1                                                  |
| HARM(tz, w, p)                       | Output is sin(w*(T - tz) + p)); T ≥tz else 0.0                                            |
| IABS(j)                              | Absolute value of integer expression j                                                    |

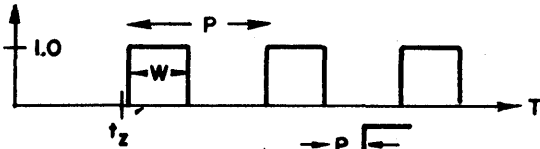
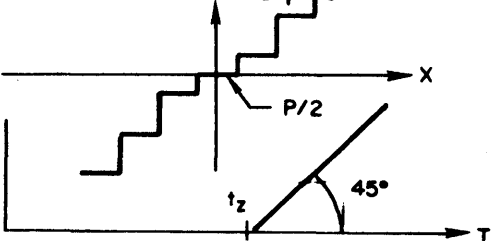
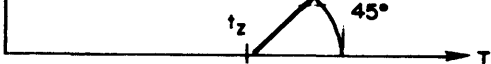
Cont.

**TABLE D-1. Summary of ACSL Model Definition Statements—Continued**

| Statement                     | Explanation                                                                                                                                          |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| IDIM(j1, j2)                  | Integer positive difference of integer expression j1 and j2                                                                                          |
| IF(lexpr) statement           | If logical expression, lexpr is .TRUE., execute statement                                                                                            |
| IMPL(yz, e, m, efl, f(y), yd) | Solves implicit equation $f(y) = y$                                                                                                                  |
| INITIAL                       | Begins section executed at beginning of every run - after each START command                                                                         |
| INT(x)                        | Integer part of real expression x                                                                                                                    |
| INTEG(xd, xic)                | Integrates derivative xd given initial value xic                                                                                                     |
| INTEGER                       | Defines type (and size) for variables, functions (and arrays)                                                                                        |
| INTVC(xd, xic)                | Integrates vector derivative xd given initial value vector xic. State, derivative and initial condition may be names only - arrays must be same size |
| ISIGN(j1, j2)                 | Absolute value of integer expression j1 times a sign of integer expression j2                                                                        |
| LEDLAG(p, q, x, ic)           | $\frac{y}{x} = \frac{ps + 1}{qs + 1}$ ; $y(0) = ic + px/q$                                                                                           |
| LIMINT(yd, ic, ll, ul)        | Limited integrator                                                                                                                                   |
| LINES(i, 'TOF')               | Inform top-of-page manager i lines are about to be written. Optional 'TOF'                                                                           |
| LOG                           | Forces data recording action when called as subroutine                                                                                               |
| LOGICAL                       | Defines type (and size) of variables, functions (and arrays)                                                                                         |
| LSW(p, tv, fv)                | Logical or integer switch                                                                                                                            |
| MACRO                         | Begins macro definition                                                                                                                              |
| MAX0(j1, j2 . . .)            | Maximum of integer expression j1, j2 . . .                                                                                                           |
| MAX1(x1, x2 . . .)            | Integerized maximum of real expression x1, x2 . . .                                                                                                  |
| MAXTERVAL MAXT = 1.0 E+10     | Defines name and value of maximum calculation interval                                                                                               |
| MERROR X=0.001, . . .         | Defines allowed relative error for state variables                                                                                                   |
| MIN0(j1, j2 . . .)            | Minimum of integer expressions j1, j2 . . .                                                                                                          |
| MIN1(x1, x2 . . .)            | Integerized minimum of real expressions x1, x2 . . .                                                                                                 |
| MINTERVAL MINT = 1.0E-10      | Defines name and value of minimum calculation interval                                                                                               |
| MOD(j1, j2)                   | Remainder when integer expression j1 is divided by integer expression j2                                                                             |


Cont.

TABLE D-1. Summary of ACSL Model Definition Statements—Continued

| Statement                      | Explanation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |      |    |  |  |   |   |    |   |      |  |   |      |
|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|----|--|--|---|---|----|---|------|--|---|------|
| MODINT(yd, ic, l1, l2)         | Moded integrator <table border="1" data-bbox="813 380 1081 596" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td colspan="2" style="text-align: center;">L1</td> </tr> <tr> <td></td> <td style="text-align: center;">T</td> <td style="text-align: center;">F</td> </tr> <tr> <td style="text-align: center;">L2</td> <td style="text-align: center;">T</td> <td style="text-align: center;">HOLD</td> </tr> <tr> <td></td> <td style="text-align: center;">F</td> <td style="text-align: center;">OPER</td> </tr> </table> |      | L1 |  |  | T | F | L2 | T | HOLD |  | F | OPER |
|                                | L1                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |      |    |  |  |   |   |    |   |      |  |   |      |
|                                | T                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | F    |    |  |  |   |   |    |   |      |  |   |      |
| L2                             | T                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | HOLD |    |  |  |   |   |    |   |      |  |   |      |
|                                | F                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | OPER |    |  |  |   |   |    |   |      |  |   |      |
| NSTEPS NSTP = 10               | Define name and value of number of steps (calculation intervals) in a communication interval - overridden by MAXT and MINT                                                                                                                                                                                                                                                                                                                                                                                                                     |      |    |  |  |   |   |    |   |      |  |   |      |
| OU( $\tau$ , m, s)             | Band limited white noise, mean m, standard deviation, s: break frequency $2\pi/\tau$ Hz                                                                                                                                                                                                                                                                                                                                                                                                                                                        |      |    |  |  |   |   |    |   |      |  |   |      |
| OUTPUT(v1, v2 . . .)           | Record values of $v_i$ each communication interval                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |      |    |  |  |   |   |    |   |      |  |   |      |
| PREPAR(v1, v2 . . .)           | Save values of $v_i$ on prepare file, each communication interval                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |      |    |  |  |   |   |    |   |      |  |   |      |
| PRINT n, L                     | Print list L according to FORMAT n. Precede with LINES(i)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |      |    |  |  |   |   |    |   |      |  |   |      |
| PROCEDURAL(y1, y2 = x1, . . .) | Begins block whose order is to be maintained. The block will be placed before statements using the $y_i$ and after those calculating the $x_i$                                                                                                                                                                                                                                                                                                                                                                                                 |      |    |  |  |   |   |    |   |      |  |   |      |
| PROGRAM text string            | First card of model definition deck. No dollar sign in text.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |      |    |  |  |   |   |    |   |      |  |   |      |
| PTR(x, y = r, $\theta$ )       | $x = r \cos \theta$ ; $y = r \sin \theta$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |      |    |  |  |   |   |    |   |      |  |   |      |
| PULSE(tz, p, w)                |                                                                                                                                                                                                                                                                                                                                                                                                                                                            |      |    |  |  |   |   |    |   |      |  |   |      |
| QNTZR(p, x)                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                            |      |    |  |  |   |   |    |   |      |  |   |      |
| RAMP(tz)                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                            |      |    |  |  |   |   |    |   |      |  |   |      |
| READ n, L                      | Read list L according to FORMAT n                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |      |    |  |  |   |   |    |   |      |  |   |      |
| REAL                           | Define type (and size) of variables, functions (and arrays). Assumed default for all names in program.                                                                                                                                                                                                                                                                                                                                                                                                                                         |      |    |  |  |   |   |    |   |      |  |   |      |
| REALPL(p, x, ic)               | $\frac{y}{x} = \frac{1}{ps + 1} \quad y(0) = ic$                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |      |    |  |  |   |   |    |   |      |  |   |      |

Cont.

TABLE D-1. Summary of ACSL Model Definition Statements—Continued

| Statement                      | Explanation                                                                                                                                                                                                         |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RESET(a)                       | Used in INITIAL section to transfer initial condition array to state array. Argument must be 'EVAL' or 'NOEVAL'                                                                                                     |
| RSW(p, tv, fv)                 | Real switch                                                                                                                                                                                                         |
| RTP(r, $\theta = x, y$ )       | $r = \sqrt{x^2 + y^2}$ ; $\theta = \text{ATAN2}(y, x)$                                                                                                                                                              |
| SAVE                           | Save current macro tables on the macro file                                                                                                                                                                         |
| SCALE(dmn, dmx = ymn, ymx)     | Convert ymx and ymn to scales on plots                                                                                                                                                                              |
| SIGN(x1, x2)                   | Absolute value of expression x1, times sign of x2                                                                                                                                                                   |
| SIN(x)                         | Sine of expression x - x in radians                                                                                                                                                                                 |
| SQRT(x)                        | Square root of expression x: Error if negative                                                                                                                                                                      |
| STEP(tz)                       |                                                                                                                                   |
| TABLE name,n,d/list/           | Define arbitrary function of n variables, dimensions d - breakpoints and function values given in list                                                                                                              |
| TAN(x)                         | Tangent of expression x - x in radians                                                                                                                                                                              |
| TERMINAL                       | Begins section entered at termination of a run                                                                                                                                                                      |
| TERMT(lexpr)                   | Identifies run termination condition; forces transfer to TERMINAL section when logical expression, lexpr, is .TRUE.                                                                                                 |
| TRAN(nn, nd, qn, qd, x)        | Transfer function: nn is ORDER of numerator; nd is ORDER of denominator, nn and nd must be integer constants, qn and qd are arrays of coefficients of s for numerator and demoninator, high order coefficient first |
| UNIF( l ,u)                    | Uniform random number distributed between lower, l, and upper, u.                                                                                                                                                   |
| UNIFI(j)                       | Initialize seed for random number generator - changes same seed as GAUSI                                                                                                                                            |
| VARIABLE T = 0.0,<br>TSC = 0.0 | Defines name and initial condition on independent variable                                                                                                                                                          |
| XERROR X = 1.0E-4              | Define allowed absolute error for state variable                                                                                                                                                                    |
| ZHOLD(ic, p, x)                | Output is x if p is .TRUE., last value output if .FALSE.                                                                                                                                                            |
| ZOH(x, ic, tz, dt, i)          | Periodic hold every dt seconds starting at tz                                                                                                                                                                       |

**TABLE D-2. Summary of ACSL Run-Time Executive Commands**

| Command  | Subcommands                                                                                                                                              | Explanation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ACTION   | 'VAR' =<br>'VAL' =<br>'LOC' =<br>'CLEAR'                                                                                                                 | Schedule action<br>Independent Variable (when)<br>Value (what)<br>Data location (where)<br>Clear action list                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| ANALYZ   | 'FREEZE' =<br>'EIGVEC' =<br>'EIGPER' =<br>'DISPLY' =<br>'LIST' =<br>'RMSEMX' =<br>'NITRMX' =<br>'FRACMX' =<br>'FRACDL' =<br>'TRIM'<br>'JACOB'<br>'EIGEN' | Hold value of state variables listed<br>Calculate eigen vectors along with eigen values<br>Calculate eigen finder performance<br>List all output on display (DIS) unit<br>List details of trim iteration<br>Specify allowable error for trim convergence<br>Specify maximum number of iterations during trim<br>Specify maximum fractional change during trim<br>Specify fraction of Newton-Raphson step taken during trim iteration<br>Initiate the trim iteration<br>Calculate and list the system Jacobian<br>Calculate system eigen values |
| CONTIN   |                                                                                                                                                          | Continue to integrate                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| DISPLY D |                                                                                                                                                          | Display values of named variables                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| END      |                                                                                                                                                          | Completes PROCEDURE definitions                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| MERROR   |                                                                                                                                                          | Establish relative errors for given state variables                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| OUTPUT   | 'NCIOUT' =<br>'CLEAR'                                                                                                                                    | Schedule names on list to have values listed during run<br>Number of communication intervals between output<br>Clear output list                                                                                                                                                                                                                                                                                                                                                                                                               |
| PLOT     | 'ALL'<br>'CHAR' =<br>'HI' =<br>'LO' =                                                                                                                    | Printer and/or line plots and/or strip plots<br>Plot all variables on prepare file<br>Use given character for plot<br>Specify upper y-axis value<br>Specify lower y-axis value                                                                                                                                                                                                                                                                                                                                                                 |

Cont.

**TABLE D-2. Summary of ACSL Run-Time Executive Commands—Continued**

| Command | Subcommands                                                                                   | Explanation                                                                                                                                                                                                                                                                                                         |
|---------|-----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|         | 'TAG' =<br>'LOG'<br>'OVER'<br>'SAME'<br>'XAXIS' =<br>'XHF' =<br>'XLO' =<br>'XTAG' =<br>'XLOG' | Specify y-axis character tag string<br>Use logarithmic scale for y-axis<br>Overplot - no axis drawn<br>Same scales for variables to left<br>Specify X-axis variable<br>Specify right-most x-axis value<br>Specify left-most x-axis value<br>Specify x-axis character tag string<br>Use logarithmic scale for x-axis |
| PREPAR  | 'CLEAR'                                                                                       | Schedule names on list to have values saved on prepare file during run<br>Clear prepared list                                                                                                                                                                                                                       |
| PRINT   | 'NCIPRN' =<br>'ALL'                                                                           | Select names on PREPARE list to have values listed in column form<br>Number of communication intervals between print lines<br>Print all variables on PREPARE file                                                                                                                                                   |
| PROCED  |                                                                                               | Define beginning of procedure block. Terminated by END.                                                                                                                                                                                                                                                             |
| RANGE   | 'ALL'<br>'IHI' =<br>'ILO' =<br>'IVAR' =                                                       | Print maximum and minimum values of selected variables on PREPARE list<br>Determine range for all variables on PREPARE list<br>High value for independent variable<br>Low value for independent variable<br>Define independent variable                                                                             |
| REINIT  |                                                                                               | Reinitialize initial conditions to current state                                                                                                                                                                                                                                                                    |
| RESTOR  |                                                                                               | Restore data area from any named file 'fn'                                                                                                                                                                                                                                                                          |
| SAVE    |                                                                                               | Save data area on a named file 'fn'                                                                                                                                                                                                                                                                                 |
| SET(S)  |                                                                                               | Set values of constants                                                                                                                                                                                                                                                                                             |
| SPARE   |                                                                                               | Links to FORTRAN subroutine of that name: Library routine gives central processor time used and that elapsed since last call                                                                                                                                                                                        |
| START   |                                                                                               | Start simulation run                                                                                                                                                                                                                                                                                                |
| STOP    |                                                                                               | Terminate simulation study                                                                                                                                                                                                                                                                                          |
| XERROR  |                                                                                               | Establish absolute errors for given state variables                                                                                                                                                                                                                                                                 |

**TABLE D-3. Executive System Variables**

| Name   | Default | Explanation                                         |
|--------|---------|-----------------------------------------------------|
| CALPLT | .FALSE. | Draw line plots                                     |
| CGDPPL | 0       | Character for grid - printer plots                  |
| CIOITG | ...     | Current integration order - integration control     |
| CMD    | 5       | Logical unit executive commands are read from       |
| CSSITG | ...     | Current step size - integration control             |
| DEFPLT | .FALSE. | Defer output of plot                                |
| DIS    | 6       | Logical unit for display output                     |
| FTSPLT | .FALSE. | Flyback trace suppression (channel 1) for all plots |
| GRDCPL | .FALSE. | Grids on line plots                                 |
| GRDSPL | .FALSE. | Grids on strip plots                                |
| HVDPRN | .FALSE. | High volume display                                 |
| LINCPL | .TRUE.  | Draw lines between points - line plots              |
| LINSPL | .TRUE.  | Draw lines between points - strip plots             |
| MALPRN | 10      | Maximum array length printed by debug               |
| MXOITG | 6       | Maximum integration order                           |
| NDEBUG | 0       | Debug listing forced if greater than zero           |
| NGXPPL | 20      | Number of points between grid lines in x direction  |
| NGYPPL | 10      | Number of points between grid lines in y direction  |
| NPCCPL | 10      | Number of points between characters - line plots    |
| NPCSPL | 10      | Number of points between characters - strip plots   |
| NPPPLT | 3       | Number of plots per page - 'ALL' plots              |
| NPXPPL | 100     | Number of points in X direction - printer plots     |
| NPYPPL | 100     | Number of points in Y direction - printer plots     |
| NRWITG | .FALSE. | No rewind of RRR file before run STARTs             |
| PLT    | 9       | Logical unit number for plot output                 |
| PRN    | 6       | Logical unit number for high volume output          |
| PRNPLT | .TRUE.  | Draw printer plots                                  |
| PSFCPL | 1.0     | Plot scale factor - line plots                      |
| PSFSPL | 1.0     | Plot scale factor - strip plots                     |
| RRR    | 8       | Logical unit for prepare file - raw run record file |
| SATCPL | .FALSE. | Suppress axis text - line plots                     |

Cont.

**TABLE D-3. Executive System Variables—Continued**

| Name   | Default | Explanation                                       |
|--------|---------|---------------------------------------------------|
| SATSPL | .FALSE. | Suppress axis text - strip plots                  |
| STRPLT | .FALSE. | Draw strip plots                                  |
| SYMCPL | .FALSE. | Plot symbols on line plots - every NPCCPL points  |
| SYMSPL | .FALSE. | Plot symbols on strip plots - every NPCSPS points |
| TBRCPL | 9600    | Terminal baud rate for graphics                   |
| TCWPRN | 132     | Terminal character width                          |
| TITLE  | BLANK   | Array of 120 characters printed at page top       |
| TSMITG | .FALSE. | Two sided matrix evaluation of Jacobian           |
| TTLCP  | .TRUE.  | Draw title on line plots                          |
| TTLSP  | .TRUE.  | Draw title on strip plots                         |
| WESITG | .TRUE.  | Write error summary - integration control         |
| XCICPL | 0.0     | X-axis crosses in inches - line plots             |
| XCISPL | 0.0     | X-axis crosses in inches - strip plots            |
| XINCPL | 5.0     | Length of X-axis in inches - line plots           |
| XINSPL | 5.0     | Length of X-axis in inches - strip plots          |
| XTICPL | 1.0     | X-axis tick increment - line plots                |
| XTISPL | 1.0     | X-axis tick increment - strip plots               |
| YASSPL | 0.5     | Y-axis separation - strip plots                   |
| YCICPL | 0.0     | Y-axis crosses in inches - line plots             |
| YCISPL | 1.0     | Y-axis crosses in inches - strip plots            |
| YINCPL | 5.0     | Length of Y-axis in inches - line plots           |
| YINSPL | 2.0     | Length of Y-axis in inches - strip plots          |
| YTICPL | 1.0     | Y-axis tick increment - line plots                |
| YTISPL | 1.0     | Y-axis tick increment - strip plots               |



THIS PAGE INTENTIONALLY LEFT BLANK

# APPENDIX E

## EXAMPLE CONTROL CARDS

This manual is accompanied by a separate booklet or addenda that gives a detailed description of how to use ACSL within the particular operating environment of your computing system. Since operating system commands differ from manufacturer to manufacturer, and even within competing groups from the same manufacturer, it is impossible to describe the details for all machines. As far as the ACSL program is concerned, it is machine transportable and models defined on one machine should produce the same results when run on another. Of course, numerical precision will vary somewhat. The main difference the user must be aware of is the characters per word when SET-ting individual elements of the TITLE array.

In order to provide some idea of how the program is actually run, this section shows the command sequence or deck set up when running under the NOS operating system with a CDC 6000 or CYBER 70 machine.

### 1. Batch Operation

Your job card

ACCOUNT (your account number, password)

CHARGE (charge number)

GET (ACSL/UN = ACLSYS)

ACSL (I = INPUT, PLT = UNI)

SAVE (PLFILE)

7/8/9 - end of record

} ACSL model definition program

7/8/9 - end of record

} ACSL run-time control cards

6/7/8/9 - end of information

This sequence will exercise the model under the command of the run-time drive cards, producing output on the line printer. Any line or strip plots will be written on an intermediate plot file (PLFILE) for later processing by UNIPOST

## 2. Terminal Operation

```
Log on - enter account number, then password
charge(your charge number)
get(acsl/un=acslsys)
acsl(i=model)
?set prn=9
 etc } ACSL run-time commands
?end
route(print,dc = pr)
bye
```

This sequence will exercise the model, the definition statements for which have been prestored on file 'MODEL'. Low volume data will be presented on the terminal, high volume being disposed of to a local line printer as the last operation.

# APPENDIX F

## ACSL ERROR MESSAGES

The following lists the error messages that can be produced during the translation phase of the program together with amplification and possible causes.

### 1. ACSL Translator Error Messages

#### BAD BREAKPOINT SPEC

The number of arguments specified in the TABLE statement is not 1, 2 or 3 or the number of breakpoint integers given does not correspond with the number of arguments.

#### BAD FUNCTION DATA COUNT

The number of data items in the TABLE statement (between slashes) does not add up to the total length expected - product of dimensions plus sum of dimensions.

#### CONFLICTING DATA TYPE

The identifier indicated in the error message is being used in a way that conflicts with an earlier usage. Some examples are:

- 1) Attempt to use a label as a variable
- 2) Using the variable in two different type declarations
- 3) Using "States" or "ICs" in logical or integer typing statements
- 4) Using duplicate "ICs" in separate INTEG statements

#### CONFLICTING LABEL NAME

The identifier indicated in the error message is either a variable being used as a statement label, or a previously defined label.

#### DIMENSIONS ALREADY SET

Attempt to define the dimensions of a previous dimensioned array.

#### EQUIVALENCE ERROR

Usually caused by not realizing the special place occupied by the first variable in the EQUIVALENCE list which is used, in the ACSL sense, as a primary variable. This variable must always appear first on the list for any subsequent EQUIVALENCE statements into the same array. Another cause is equivalencing the same secondary variable to two different primary variables. System variables such as CINT, MAXT, state, derivative and initial condition variables must be given as primary variables

#### ILLEGAL BLOCK COMBINATION

An ACSL block delimiter (PROGRAM, INITIAL, END, etc) was encountered, out of the proper sequence.

#### ILLEGAL DERIVATIVE DEFINITION

Derivative arrays are not allowed except in INTVC statements.

#### INSUFFICIENT REGION FOR TRANSLATION

Insufficient field length for ACSL to attempt translation. Always fatal. Increase memory region parameter and resubmit job.

## **INTEG NOT IN SORT BLOCK**

Integration statements must be in a DERIVATIVE block (an implicit PROGRAM . . . END structure is synonymous with a DERIVATIVE block) rather than the INITIAL, DYNAMIC or TERMINAL section. Usually caused by an END mismatch which terminates a block inadvertently. May also be caused by an integration statement inside a PROCEDURAL . . . END block nested to level two or higher.

## **MACRO ARGUMENT ERROR**

Error in the macro definition and attempting to invoke the macro, or the computed argument number for the main argument is less than "1" or greater than the original number of arguments.

## **MACRO NOT DEFINED**

Attempt to use a macro which has not been defined. The syntax of the statement indicates a macro is present but the macro is not found in the macro file.

## **MACRO NOT INVOKED**

Attempt to use an "ASSIGNED" variable when variable was not defined in an ASSIGN statement.

## **MACRO STATEMENT ORDER ERROR**

During macro expansion: No standard value for an unspecified macro argument, or an attempt to use the dimensions of an undimensioned argument.

## **MISSING DERIVATIVE STATEMENT**

An INTEG statement is missing for some element of a state array.

## **MULTIPLY DEFINED SYMBOL**

The named variable has previously been assigned a value in this sort block or derivative section.

## **NO RIGHT SIBLING**

Internal ACSL system error during syntax analysis. Report to system staff with example.

## **NO TABLE SPACE LEFT**

Insufficient field length for ACSL to continue running. Usually caused by the use of a large number of variables in a large program. Can also be caused by arithmetic loops encountered in statement sorting. An increase in field length of 10K should be more than sufficient in most cases. Always fatal.

## **OUTSIDE TABLE LIMIT**

A system error in the ACSL translator. Report to system staff with example.

## **PARAMETER NOT FOUND**

Attempt to assign a standard value to a variable other than one of the main arguments in a MACRO definition.

## **PROC. ENDED INCORRECTLY**

An ACSL block delimiter, other than "END", was found inside a "PROCEDURAL" block.

## **PREMATURE END OF FILE**

An end of file was encountered on the translator input file before the logical end of the ACSL source program was found. Implies an incomplete program or a missing END statement.

## **TABLE ALREADY DEFINED**

The table name has already been used in another context prior to the current table definition. Either as a simple variable, another TABLE definition or simple use. TABLE functions must be defined before

their first use since otherwise the translator cannot distinguish the use from normal FORTRAN functions.

#### **TOO MANY CHARACTERS**

The truncated identifier, indicated by the message, contains more than 6 characters. Translation continues with the truncated version of the identifier.

#### **TOO MANY ENDS**

Statements remain on the input file after the final END of the ACSL program. The remaining statements are listed but not processed. Implies that too many END statements were included in the program, or that statements were misplaced, or that FORTRAN routines are included with the source deck but don't start with SUBROUTINE, FUNCTION, INTEGER FUNCTION, etc.

#### **UNSORTABLE STATEMENT BLOCK**

An arithmetic loop was encountered during sorting, and is listed below the message. The statements will not appear on the compile file, but the statements which depend on this block will.

#### **VARIABLE NOT DIMENSIONED**

The variable indicated in the error message is being used as an array without being declared as such by an ARRAY, DIMENSION or other such typing statement.

#### **WRONG DIMENSIONS**

Attempt to use an array with more than 3 subscripts.

### **2. RUN-TIME ERROR MESSAGES**

Run-time error messages produced by the ACSL executive processing the model drive cards, are as follows:

#### **CAN-T FIND ARRAY ELEMENT IN name**

A reference has been made to a particular array element that doesn't exist or has already been deleted once before by a preceding FREEZE.

#### **CAN-T FIND name**

The 'name' does not appear in the dictionary or in the case of PLOT, PRINT or RANGE commands, might not have been included on the PREPAR list.

#### **CAN-T SATISFY ERROR CRITERION**

The integration algorithm has decided it needs to take a step smaller than MINT to keep the largest error within bound. Must be acknowledged with ERRTAG (q.v.) or the simulation run will stop.

#### **CANNOT TRIM WITH DELAY FUNCTION**

An attempt has been made to use the TRIM subcommand of ANALYZ. The Jacobian is in error if DELAY functions are used in the model.

#### **CONSTANT COLUMN - NUMBER IS n**

In trying to invert the Jacobian, a constant column has been identified and indicated by numeric value into the list of unfrozen state variables. Usually the result of keeping a state variable in the iteration that has no influence on any other part of the model.

#### **CONSTANT ROW - NUMBER IS n**

In trying to invert the Jacobian, a constant row has been identified. Usually the result of keeping a state variable in the iteration with a constant (may be zero) derivative.

## END OF COMMAND FILE ENCOUNTERED

An end of file has been sensed on the file identified by logical unit CMD. The simulation study is terminated and control reverts to the operating system.

## ERROR AFTER name

Some sort of syntax error has been determined after the given 'name' appears on the command card image. Trying to specify NCI by a real number (2.0 for example): 'CHAR' value as a logical for instance.

## ERROR IN EIGEN ANALYSIS ROUTINE

### ERROR NUMBER IS n

An error was made in trying to determine the eigen values of the Jacobian matrix. See error listing for eigen analysis routine actually used.

## FILE LENGTH TOO SHORT

The RESTORE command found the data block did not match the user block length. Usually caused by attempting to RESTORE a file SAVED from a previous translation run. Changes to the ACSL program will invalidate previous assignments.

## FILE NAME SHOULD BE QUOTED

The argument for the SAVE/RESTORE commands should be a valid file name in quotes, i.e.

SAVE 'JOE'

File names should start with a letter and contain no more than six characters (letters or digits)

## ILLEGAL COMMAND WORD

The command word starting the statement is not in the standard list nor is it the name of any PROCEDURE established.

## ILLEGAL DATA TYPE FOLLOWING name

Something is wrong with the command following the symbol 'name' and before any following symbol.

## INDEPENDENT VARIABLE CHANGED IN JACOBIAN EVALUATION name

A state variable has been modified inside the DERIVATIVE section used to calculate the Jacobian. Since the Jacobian evaluator uses numerical perturbation of the unfrozen state variables, any other modification will invalidate the calculation.

## INSUFFICIENT AREA FOR DELAY FUNCTION

The array length specified in one of the DELAY functions is too small to accommodate all the data points needed. Usually produced when the model goes through a region requiring a very small step size.

## INSUFFICIENT DATA

The statement is terminated when data was expected.

## JACOBIAN DETERMINANT ZERO, CAN-T TRIM

The TRIM subcommand of ANALYZ has found that the Jacobian has a zero determinant so the Newton-Raphson iteration can't proceed. Using FREEZE prior to the TRIM can usually eliminate the offending state.

## LINE PLOT LIBRARY NOT LOADED

An attempt has been made to make line plots (PLOT with CALPLT true) without instructing the loader or link-edit program to substitute the appropriate device driver. See local addendum for devices available and JCL or system control cards required.

## LINEAR ANALYSIS ROUTINE NOT LOADED

An attempt has been made to use the run time command ANALYZ without ensuring that the routines are present to handle the command. Due to infrequent usage, the normal default is to omit the routines that handle the ANALYZ command in order to save memory and load time. See local addendum for JCL required to instruct the loader or link-editor to link in the appropriate modules.

## LIST DOESN'T CONTAIN name

One of the ANALYZ subcommands is looking for a variable in a list. For instance FREEZE can only apply to state variables.

## NAME ALREADY DEFINED

In using LISTD to process a set of dictionary definitions, the same variable name has appeared twice. Usually caused by omitting continuation digits from column ten (10) for long definitions.

## NAME ALREADY IN DICTIONARY

In using BLDDCT to extend to the ACSL dictionary an attempt was made to add a name already present in the dictionary.

## NAME FOR PROCEDURE NOT GIVEN OR ILLEGAL

The procedure name is not of the correct form following the PROCED command.

## NAME MUST PRECEDE DATA

A data item is given before a name has been established to store the value in, i.e.

SET A = 2.0 is alright

but SET 2.0 = A is wrong

## NAME OR ELEMENT NOT ON PREPAR LIST - name

One of the commands that refers to the PREPAR list (PLOT, PRINT, RANGE, etc) has a variable name or array element in the list that was not included in the original PREPAR command i.e. PREPAR Y(2), Y(3) followed later by PLOT Y(1).

## NEED A NAME FIRST

Modifiers to one of the plot variables refer to a preceding symbol. If no symbol is given it is an error, i.e..

PLOT 'LO' = 0.0 'HI' = 5.0, Y1 is incorrect

## NO MORE TABLE SPACE, MAX LENGTH USED IS i

The run time table space manager has run out of space and the simulation study must be aborted. Frequently associated with using the stiff integration algorithm which needs 2N squared words, where N is the number of the state variables. See local addendum for mechanism to increase table space at run-time.

## NO SPACE LEFT IN DICTIONARY FOR name

In attempting to extend the user dictionary with BLDDCT, the common block space designated for the dictionary has been filled and no more names can be added. Length may not have been established with NITBLD. Extend the length of the dictionary common block/ZZDCT/ in the user supplied main program.

## NO USER SUPPLIED INTEGRATION ROUTINE

An attempt has been made to use integration algorithm seven (IALG=7) without supplying a subroutine INTEG to handle the integration.



#### REFERENCE OUT OF LIMIT OF ARRAY name

An array element is referred to that is outside the declared size of the array. In a SET command sequential data items go into succeeding slots of an array. Each is checked to make sure the array bound is not exceeded, i.e., if A is an array of size five, the following

```
SET A(3) = 1.5, 2.5, 5.0, 6.0
```

would produce this message since the 6.0 is to be stored in A(6).

#### REFERENCE TO NON-STATE VARIABLE name

An attempt has been made to specify error tolerances with XERROR or MERROR for a variable that is not a state variable. Check names in state list from debug output.

#### RESCALE NOT IMPLEMENTED

Reference has been made to the old ACSL system symbols RSCCPL, RSCPPL or RSCPLT that were supposed to cause dynamic rescaling within a plot. This feature was eliminated at level 6M.

#### STEP SIZE TOO LARGE. STATE - n

A zero determinant has been found in trying to invert the matrix  $(I + hA)$  in the stiff integration algorithm. Should never happen but may be fixed by reducing the allowable step size.

#### SYNTAX ERROR X = Y++Y

```

```

The statement is repeated and the line of asterisks stops where the first syntax error occurred.

#### TAG TOO LONG AFTER name

A TAG string on a PLOT command has too many characters. Actual number allowed will depend on machine type but all machines will accept twenty (20) character messages.

#### TOO MANY ITERATIONS, CAN-T CONVERGE

The TRIM sub-command of ANALYZ has failed to converge within the specified number of iterations. Either increase the numbers of iterations (NITRMX), reduce the convergence criteria (RMSEMX) or decrease the step (FRACDL). Use REINIT to hold on to any gains obtained with the current iteration.

#### TYPE CONFLICT ON STORE INTO name

The data does not agree with the predetermined type of 'name'. Logical data can only be .TRUE. (.T.) or .FALSE. (.F.). Integers are allowed into reals, but all other combinations are illegal.

#### X-AXIS SCALES INCORRECT FOR LOG PLOTS

#### Y-AXIS SCALES INCORRECT FOR LOG PLOTS on name

The scale values are either negative or zero when making logarithmic plots. Usually the LO value must be specified since the normal rounding will change the LO axis marker to zero.

#### ZERO PIVOT ELEMENT FOUND AT STATE INDEX n

In using TRIM within ANALYZ, a zero determinant has been found when trying to invert the Jacobian. This message identifies the row number at which the zero pivot element first showed up and can sometimes be correlated to a state variable having a degenerate relationship with the rest of the model.

# INDEX

|                                   |              |                                  |         |
|-----------------------------------|--------------|----------------------------------|---------|
| ABS .....                         | 4-2          | Arc-Tangent .....                | 4-6,4-7 |
| Absolute Error at Runtime .....   | 5-6          | Arresting Gear Example .....     | A-41    |
| Absolute Errors .....             | 4-34         | Arithmetic Operators .....       | 2-4     |
| Absolute Per Step Error .....     | 4-34         | ARRAY .....                      | 4-5     |
| Absolute Value .....              | 4-2,4-24     | Array GET .....                  | B-1     |
| Accumulated CP Time .....         | 5-12         | Array Indices at Run Time .....  | 5-1     |
| ACOS .....                        | 4-3          | Array Initialization (SET) ..... | B-4     |
| ACSL Examples .....               | A-1          | Array Put .....                  | B-1     |
| ACSL PROGRAM Card .....           | 4-41         | ASIN .....                       | 4-6     |
| ACSL Program Flow Diagram .....   | 3-2          | Aspirin Dosage Example .....     | A-112   |
| ACSL Statements .....             | 4-1          | ASSIGN .....                     | 4-6     |
| ACTION .....                      | 5-2          | ASSIGN (MACRO) .....             | 6-4     |
| Adams-Moulton .....               | 4-3          | Assignment Statement .....       | 4-6     |
| Addition .....                    | 2-4          | ATAN .....                       | 4-6     |
| AGET .....                        | B-1          | ATAN2 .....                      | 4-7     |
| AINT .....                        | 4-3          | Axis Crossing Point .....        | C-3,C-4 |
| Aircraft Longtdnl Motion .....    | A-51         | Axis Lengths .....               | C-3,C-4 |
| Airframe Stability .....          | A-51         | Axis Tick Mark Increment .....   | C-3,C-4 |
| Algebraic Loop .....              | 4-25         | Backlash .....                   | 4-7     |
| ALGORITHM .....                   | 4-3          | Band Limited White Noise .....   | 4-37    |
| ALL Subcommand .....              | 5-8,5-9,5-10 | Batch Operation .....            | E-1     |
| ALOG .....                        | 4-4          | Baud Rate .....                  | C-3     |
| ALOG10 .....                      | 4-5          | BCKLSH .....                     | 4-7     |
| Alphanumeric Labels .....         | 2-3          | BLDDCT .....                     | B-1     |
| AMAX0 .....                       | 4-5          | Block Descriptor Names .....     | 4-15    |
| AMAX1 .....                       | 4-5          | Block Transfer (XFERB) .....     | B-5     |
| AMIN0 .....                       | 4-5          | Boolean Operators .....          | 2-5     |
| AMIN1 .....                       | 4-5          | BOUND .....                      | 4-8     |
| AMOD .....                        | 4-5          | Branch Between Sections .....    | 3-3     |
| ANALYZ .....                      | 5-2          | Branch Statement (GO TO) .....   | 4-23    |
| Append a Sign (SIGN, ISIGN) ..... | 4-45,4-31    | Calculated Flags .....           | 3-3     |
| APUT .....                        | B-1          | CALL .....                       | 4-8     |
| Arbitrary Functions .....         | 4-46         | CALPLT .....                     | C-1     |
| Arc-Cosine .....                  | 4-3          | Central Processor Usage .....    | 5-12    |
| Arc-Sine .....                    | 4-6          | Centralized Integration .....    | 4-27    |

|                                          |             |                                    |          |
|------------------------------------------|-------------|------------------------------------|----------|
| CGDPPL .....                             | C-2         | Data Statements .....              | 4-11     |
| CHAR Subcommand .....                    | 5-7,5-8     | DBG .....                          | 4-11     |
| Character for Grid (CGDPPL) .....        | C-2         | DBLINT .....                       | 4-12     |
| Character for Printer Plots .....        | 5-7         | DEAD .....                         | 4-12     |
| Characters on Line Plots .....           | C-3         | Dead Space .....                   | 4-12     |
| Checkout .....                           | 7-1         | DEBUG .....                        | B-2      |
| CINTERVAL .....                          | 4-8         | Debug Print Out .....              | 7-2      |
| CIOITG .....                             | C-5         | Debug Subroutine Call .....        | B-2      |
| Circulatory System Model .....           | A-62        | Debugging .....                    | 7-1      |
| CLEAR Subcommand .....                   | 5-6,5-9,5-2 | Defer Plots (DEFPLT) .....         | C-1      |
| CMD .....                                | C-5         | DEFPLT .....                       | C-1      |
| CMPXPL .....                             | 4-10        | DELAY .....                        | 4-13     |
| Coding Line, Model Definition .....      | 1-4         | Denominator Polynomial .....       | 4-49     |
| Coding Procedure .....                   | 1-4         | DERIVATIVE .....                   | 4-14     |
| Column Print .....                       | 5-10        | Derivative Operator .....          | 4-16     |
| Command Sequences (PROCD) .....          | 5-10        | DERIVT .....                       | 4-16     |
| Comments .....                           | 4-10,5-5    | Dictionary Extension .....         | B-1      |
| Common Block in FORTRAN Routine .....    | 1-5         | Dictionary for a Model .....       | B-3,A-94 |
| Communication Interval Divisor .....     | 4-37        | Dictionary Listing .....           | B-3      |
| Communication Interval .....             | 4-8,1-5     | Differences, ACSL to FORTRAN ..... | 2-1      |
| Complex Pole .....                       | 4-10        | Differentiation, Approximate ..... | 4-16     |
| Concatenation .....                      | 6-1,6-8     | DIM .....                          | 4-17     |
| Constants .....                          | 2-1,4-11    | Dimension .....                    | 4-5      |
| CONTIN .....                             | 5-5         | DIS .....                          | C-5      |
| CONTINUE .....                           | 4-11        | DISCRETE .....                     | 4-17     |
| Continue Previous Run .....              | 5-5         | Discrete Compensator Example ..... | A-104    |
| Control Card Example (NOS) .....         | E-1         | Discrete Interval .....            | 4-29     |
| Control Loop Example .....               | A-7         | Display Data Value .....           | 5-5      |
| COS .....                                | 4-11        | DISPLY .....                       | 5-5      |
| Cosine Function .....                    | 4-11        | DISPLY Subcommand .....            | 5-3      |
| CSSITG .....                             | C-5         | DIVIDE (MACRO) .....               | 6-4      |
| Current Integration Order .....          | C-5         | Division .....                     | 2-4      |
| Current Step Size (Monitor) .....        | C-5         | DO Statement .....                 | 4-19     |
| Damped Oscillation .....                 | A-7         | Dot Product MACRO .....            | 6-8      |
| Data Logging .....                       | 4-33        | DYNAMIC .....                      | 4-19     |
| Data Logging Subroutine Call (LOG) ..... | B-3         | Eigen Performance .....            | 5-3      |
| Data Move (XFERB) .....                  | B-5         | EIGEN Subcommand .....             | 5-3      |
| Data Recording Interval .....            | 4-8         | Eigen Value/Vectors .....          | 5-3      |

|                                         |           |                                      |           |
|-----------------------------------------|-----------|--------------------------------------|-----------|
| EIGPER Subcommand .....                 | 5-3       | Gear Stiff Integration .....         | 4-3       |
| EIGVEC Subcommand .....                 | 5-3       | GO TO STATEMENT .....                | 4-23      |
| END .....                               | 5-5,4-20  | GRDCPL .....                         | C-2       |
| EQUIVALENCE .....                       | 4-20      | GRDSPL .....                         | C-4       |
| Equivalenced Variable .....             | 4-20      | Grid on Line Plots .....             | C-2       |
| Error Messages .....                    | F-1       | Grid Separation, X-axis .....        | C-2       |
| ERRTAG .....                            | 4-21      | Grid Separation, Y-axis .....        | C-2       |
| Estimated Error .....                   | 5-6,4-34  | HARM .....                           | 4-24      |
| Euler Integration .....                 | 4-3       | Harmonic Forcing Functn (HARM) ..... | 4-24      |
| EVAL .....                              | 4-43      | Heart-Lung Model .....               | A-62      |
| Event List .....                        | 4-15      | Heat Flow .....                      | A-28      |
| EXIT (MACRO) .....                      | 6-5       | HI Subcommand .....                  | 5-7       |
| EXP .....                               | 4-21      | High Volume Data to Terminal .....   | C-4       |
| EXPF .....                              | 4-21      | Hold Mode (MODINT) .....             | 4-36      |
| Explicit Structure .....                | 3-1       | Hollerith Constants .....            | 2-2       |
| Exponential (EXP) .....                 | 4-21      | HVDPRN .....                         | C-4       |
| Exponential Forcing Functn (EXPF) ..... | 4-21      | Hysteresis .....                     | 4-24      |
| Exponentiation .....                    | 2-4       | I/O Statements .....                 | 4-30      |
| Expressions .....                       | 2-4       | IABS .....                           | 4-24      |
| FCNSW .....                             | 4-22      | IDIM .....                           | 4-25      |
| First Order Lag .....                   | 4-43      | IF Statements .....                  | 4-25      |
| Fly Back Trace Suppression .....        | C-1       | IGET .....                           | B-3       |
| FORMAT .....                            | 4-22      | IHI Subcommand .....                 | 5-10      |
| FORTRAN PRINT .....                     | 4-31      | ILO Subcommand .....                 | 5-10      |
| FORTRAN READ .....                      | 4-31      | IMPL .....                           | 4-25      |
| FORTRAN Subroutines .....               | 1-5       | Implicit or Algebraic Loops .....    | 4-25      |
| FORTRAN WRITE .....                     | 4-31      | Impulse Response .....               | 8-3       |
| FRACDL Subcommand .....                 | 5-4       | INCREMENT (MACRO) .....              | 6-4       |
| FRACMX Subcommand .....                 | 5-4       | Independent Variable .....           | 4-51      |
| Fractional Change During Trim .....     | 5-4       | Independent Variable for Range ..... | 5-10      |
| Freeze State Variables .....            | 5-3       | INITIAL .....                        | 4-27      |
| FREEZE Subcommand .....                 | 5-3       | Initial Condition Expression .....   | 4-28      |
| FTSPLT .....                            | C-1       | Initial Conditions .....             | 4-27      |
| Function Switch (RSW or FCNSW) .....    | 4-33,4-22 | Initialize Noise Generators .....    | 4-23      |
| Functions of 1, 2 or 3 Variables .....  | 4-46      | Input Statements .....               | 4-30,5-11 |
| GAUSI .....                             | 4-23      | INT .....                            | 4-27      |
| GAUSS .....                             | 4-22      | INTEG (Subroutine) .....             | B-3       |
| Gaussian Noise (OU or GAUSS) .....      | 4-37,4-22 | INTEG (ACSL Operator) .....          | 4-27      |

|                                          |               |                                               |               |
|------------------------------------------|---------------|-----------------------------------------------|---------------|
| INTEGER .....                            | 4-29          | List Dictionary (LISTD) .....                 | B-3           |
| Integer Absolute Value .....             | 4-24          | LIST Subcommand .....                         | 5-3           |
| Integer Positive Difference .....        | 4-25          | LISTD .....                                   | B-3           |
| Integerization of Real Argument .....    | 4-27          | LO Subcommand .....                           | 5-7           |
| Integration .....                        | 4-27          | LOC, Subcommand of Action .....               | 5-2           |
| Integration Algorithms .....             | 4-3           | LOG .....                                     | 4-33          |
| Integration Debug .....                  | B-5           | LOG Subcommand .....                          | 5-7           |
| Integration Step Size .....              | 4-10          | Logarithm Base 10 (A <sub>LOG</sub> 10) ..... | 4-5           |
| Integration, Rules on Intvc .....        | 4-30          | Logarithmic Scaling .....                     | 5-7           |
| Integration, Rules on Integ .....        | 4-28          | LOGICAL .....                                 | 4-33          |
| Interactive Operation .....              | E-2           | Logical Expressions .....                     | 2-5           |
| INTERVAL .....                           | 4-29          | Logical Switch .....                          | 4-33          |
| INTVC .....                              | 4-30          | Logical Unit for Command Stream .....         | C-5           |
| Inverse Cosine .....                     | 4-3           | Logical Unit for Display .....                | C-5           |
| Inverse Sine .....                       | 4-6           | Logical Unit for High Volume Output .....     | C-5           |
| ISIGN .....                              | 4-31          | Lower Limit .....                             | 4-8,4-32,4-12 |
| Iterations for Trim .....                | 5-4           | LSW .....                                     | 4-33          |
| IVAR Subcommand .....                    | 5-10          | MACRO .....                                   | 6-1,4-34      |
| JACOB Subcommand .....                   | 5-2           | Macro Arguments .....                         | 6-1           |
| Jacobian .....                           | 5-2           | MACRO ASSIGN .....                            | 6-4           |
| Job Processing .....                     | 1-1           | MACRO Calls .....                             | 6-10          |
| Labels .....                             | 2-3           | MACRO CONTINUE .....                          | 6-5           |
| Language Elements .....                  | 2-1           | MACRO DECREMENT .....                         | 6-4           |
| Language Features .....                  | 1-1           | MACRO Definitions .....                       | 6-1           |
| Laplace Operator, S .....                | 4-49          | MACRO Descriptions .....                      | 6-1           |
| Lead-Lag Compensator .....               | 4-31          | MACRO Directives .....                        | 6-6,6-3       |
| LEDLAG .....                             | 4-31          | MACRO DIVIDE .....                            | 6-4           |
| LIMINT .....                             | 4-32          | MACRO Examples .....                          | 6-7           |
| Limit .....                              | 4-8,4-12,4-32 | MACRO EXIT .....                              | 6-5           |
| Limit Cycle .....                        | A-1,1-2       | MACRO GO TO .....                             | 6-5           |
| Limited Integrator .....                 | 4-32          | MACRO Header .....                            | 6-2           |
| LINCPL .....                             | C-2           | MACRO IF .....                                | 6-6           |
| Line Count With FORTRAN Write .....      | 4-32          | MACRO INCREMENT .....                         | 6-4           |
| Line Plots (CALPLT) .....                | C-1           | MACRO Language .....                          | 6-1           |
| Linear Ramp .....                        | 4-42          | MACRO MULTIPLY .....                          | 6-4           |
| LINES .....                              | 4-32          | MACRO Primary Arguments .....                 | 6-2           |
| Lines Between Points on Line Plots ..... | C-2           | MACRO PRINT .....                             | 6-6           |
| LINSPL .....                             | C-4           | MACRO REDEFINE .....                          | 6-6           |

|                                            |      |                                       |           |
|--------------------------------------------|------|---------------------------------------|-----------|
| MACRO RELABEL .....                        | 6-7  | NGXPPL .....                          | C-2       |
| MACRO Secondary Arguments .....            | 6-2  | NGYPPL .....                          | C-2       |
| MACRO STANDVAL .....                       | 6-7  | NITBLD .....                          | B-1       |
| MAIN VARIABLE .....                        | 4-20 | NITRMX Subcommand .....               | 5-4       |
| MALPRN .....                               | C-4  | No Rewind Flag .....                  | C-5       |
| Mass-Spring-Damper Model .....             | A-7  | No Sort Action (PROCEDURAL) .....     | 4-40      |
| Matrix Integration .....                   | 4-30 | NOEVAL .....                          | 4-43      |
| MAX0 .....                                 | 4-34 | NOISE .....                           | 4-37,4-22 |
| MAX1 .....                                 | 4-34 | NPCCPL .....                          | C-3       |
| Maximum Array Length for Debug .....       | C-4  | NPCPPL .....                          | C-2       |
| Maximum Integration Step Size .....        | 4-35 | NPCSPL .....                          | C-4       |
| Maximum Integration Order .....            | C-5  | NPPPLT .....                          | C-2       |
| Maximum of Integer Arguments (AMAX0) ..... | 4-5  | NPXPPL .....                          | C-2       |
| Maximum of Real Arguments (AMAX1) .....    | 4-5  | NPYPPL .....                          | C-2       |
| MAXTERVAL .....                            | 4-34 | NRWITG .....                          | C-5       |
| MERROR .....                               | 4-34 | NSTEPS .....                          | 4-37      |
| MIN0 .....                                 | 4-35 | Number of Plots Per Page .....        | C-2       |
| MIN1 .....                                 | 4-35 | Number of Points For Grid .....       | C-2       |
| Minimum Integration Step Size .....        | 4-35 | Number of Points in Y Direction ..... | C-2       |
| Minimum of Integer Arguments (AMIN0) ..... | 4-5  | Number of Points in X Direction ..... | C-2       |
| Minimum of Real Arguments (AMIN1) .....    | 4-5  | Number of Points Per Character .....  | C-2,C-3   |
| MINTERVAL .....                            | 4-35 | Numerator Polynomial .....            | 4-49      |
| Missile Airframe Example .....             | A-84 | Operate Mode (MODINT) .....           | 4-36      |
| MOD .....                                  | 4-36 | Operators .....                       | 4-1       |
| Moded Integrator (MODINT) .....            | 4-36 | Ornstein-Uhlenbeck Process (OU) ..... | 4-37      |
| Model Coding .....                         | 1-4  | OU .....                              | 4-37      |
| Model Execution (START) .....              | 5-12 | OUTPUT .....                          | 5-6,4-39  |
| Model Split, Derivative Sections .....     | 4-14 | OUTPUT at End of Run .....            | 8-3       |
| MODINT .....                               | 4-36 | Output Communication Rate .....       | 5-6       |
| Modulus of Real Arguments (AMOD) .....     | 4-5  | Output Lines Written (LINES) .....    | 4-32      |
| Multiple Derivatives .....                 | 4-14 | Output List Defined in Model .....    | 4-39      |
| Multiplication .....                       | 2-4  | Over Plot .....                       | 5-8       |
| MULTIPLY (MACRO) .....                     | 6-4  | OVER Subcommand .....                 | 5-8       |
| MXOITG .....                               | C-5  | PAGE .....                            | 4-39      |
| Natural Logarithm (ALOG) .....             | 4-4  | Page Control (PAGE and LINES) .....   | 4-39      |
| NCIOUT Subcommand .....                    | 5-6  | Parameter Sweep .....                 | 8-2       |
| NCIPRN Subcommand .....                    | 5-9  | Phase and Gain Example .....          | A-76      |
| NDEBUG .....                               | C-5  | Phase Plane Plot .....                | A-5       |

|                                           |          |                                            |           |
|-------------------------------------------|----------|--------------------------------------------|-----------|
| PHYSBE Example .....                      | A-62     | Random Number Seed .....                   | 4-22      |
| Physiological Simulation .....            | A-62     | Random Numbers .....                       | 4-22,4-51 |
| Pilot Ejection Example .....              | A-19     | RANGE .....                                | 5-10      |
| PLOT .....                                | 5-7      | Raw Run Record File .....                  | C-6       |
| Plot Scale Factor .....                   | C-3,C-4  | READ .....                                 | 4-43      |
| PLT .....                                 | C-5      | REAL .....                                 | 4-43      |
| Points per Character on Line Plots .....  | C-3      | Real Integerization (AINT) .....           | 4-3       |
| Polar to Rectangular .....                | 4-41     | Real Pole (REALPL) .....                   | 4-43      |
| Position Limit (DBLINT) .....             | 4-12     | Real Switch (RSW) .....                    | 4-33      |
| Positive Difference (DIM) .....           | 4-17     | REALPL .....                               | 4-43      |
| Precision .....                           | 2-1      | Rectangular to Polar .....                 | 4-44      |
| Predictor Corrector .....                 | 4-3      | REDEFINE (MACRO) .....                     | 6-6       |
| PREPAR .....                              | 5-9,4-39 | REINIT .....                               | 5-11      |
| PREPAR List Change Rule .....             | 5-7      | Reinitialize State Variables .....         | 5-11      |
| PRINT .....                               | 5-9,4-40 | RELABEL (MACRO) .....                      | 6-7       |
| Printer Plot, Points on X-Axis .....      | C-2      | Relational Expressions .....               | 2-5       |
| Printer Plot, Points on Y-Axis .....      | C-2      | Relational Operators .....                 | 2-4       |
| Printer Plots, Points per Character ..... | C-2      | Relative Errors .....                      | 4-34      |
| Printer Plots (PRNPLT) .....              | C-2      | Remainder With Real Arguments (AMOD) ..... | 4-5       |
| PRN .....                                 | C-5      | Reserved Names .....                       | 1-4       |
| PRNPLT .....                              | C-2      | RESET .....                                | 4-43      |
| PROCED .....                              | 5-10     | Reset Mode (MODINT) .....                  | 4-36      |
| PROCEDURAL .....                          | 4-40     | Residual for Trim .....                    | 5-3       |
| PROGRAM .....                             | 4-41     | RESTOR .....                               | 5-11      |
| Program Execution Time (TIMER) .....      | B-4      | Restore User Data Area .....               | 5-11      |
| Program Flow .....                        | 3-1      | RGET .....                                 | B-3       |
| Program Output .....                      | 3-3      | RMSEMX Subcommand .....                    | 5-3       |
| Program Sorting .....                     | 3-3      | Round Scale Factors .....                  | 4-45      |
| PSFCPL .....                              | C-3      | RRR .....                                  | C-6       |
| PSFSPL .....                              | C-4      | RSW .....                                  | 4-33      |
| PTR .....                                 | 4-41     | RTP .....                                  | 4-44      |
| PULSE .....                               | 4-41     | Runtime Output .....                       | 5-6       |
| Pulse Train (PULSE) .....                 | 4-41     | Runge-Kutta Integration .....              | 4-27,4-3  |
| QNTZR .....                               | 4-42     | Runtime Command Format .....               | 5-1       |
| Quantization (QNTZR) .....                | 4-42     | Runtime Commands .....                     | 5-1       |
| Quick Reference Guide .....               | D-1      | Runtime Procedures .....                   | 5-10      |
| Radiating Fin Example .....               | A-28     | Runtime Value Assignment .....             | 5-11      |
| RAMP .....                                | 4-42     | Same Scale Factors on Plots .....          | 5-8       |

|                                        |           |                                        |           |
|----------------------------------------|-----------|----------------------------------------|-----------|
| SAME Subcommand .....                  | 5-8       | Subroutine Call .....                  | 4-8       |
| Sampled Data .....                     | A-104     | Subscripted Variables .....            | 2-2       |
| Sampler Macro Example .....            | 6-7       | Subtraction .....                      | 2-4       |
| SATCPL .....                           | C-3       | Summary ACSL Operators .....           | D-1       |
| SATSPL .....                           | C-4       | Summary Runtime Commands .....         | D-6       |
| SAVE .....                             | 5-11,4-45 | Summary System Symbols .....           | D-8       |
| Save User Data Area .....              | 5-11      | Suppress Text on Plot Axes .....       | C-3,C-4   |
| SCALE .....                            | 4-45      | Switch .....                           | 4-33,4-22 |
| Scale Factors for Plots .....          | 5-8,4-43  | Symbol Table .....                     | 3-3       |
| Second Order Transfer Function .....   | 4-10      | Symbols on Plots .....                 | C-3,C-4   |
| SET (Runtime Command) .....            | 5-11      | SYMCPL .....                           | C-3       |
| SET (Subroutine) .....                 | B-4       | SYMSPL .....                           | C-4       |
| SIGN .....                             | 4-45      | Syntax Error Indicator .....           | 7-1       |
| SIN .....                              | 4-46      | System Symbol Access .....             | B-1       |
| Sine Function .....                    | 4-46      | System Symbols .....                   | C-1       |
| Sinusoidal Forcing Functn (HARM) ..... | 4-24      | TABLE .....                            | 4-46      |
| Sorting .....                          | 3-3       | Tag Strings on Plots .....             | 5-7,5-8   |
| SPARE .....                            | 5-12      | TAN .....                              | 4-48      |
| Spring Example .....                   | A-7       | Tangent of an Angle .....              | 4-48      |
| SQRT .....                             | 4-46      | TBRCPL .....                           | C-3       |
| Square Root Function .....             | 4-46      | TCWPRN .....                           | C-4       |
| Standard Value for MACRO Arogs .....   | 6-7       | Temperature Distribution .....         | A-28      |
| STANDVAL (MACRO) .....                 | 6-7       | TERMINAL .....                         | 4-48      |
| START .....                            | 5-12      | Terminal Baud Rate .....               | C-3       |
| State List .....                       | 4-29      | Terminal Operation .....               | E-2       |
| State Transfer .....                   | 4-43      | Terminal Width .....                   | C-4       |
| State Transition Matrix .....          | 4-4       | TERMINAL-INITIAL Iteration .....       | A-28      |
| State Variables .....                  | 4-27      | Terminate Statement (TERMT) .....      | 4-48      |
| Statement Order .....                  | 4-2       | TERMT .....                            | 4-48      |
| Statement Separator .....              | 1-4,4-11  | Time Constant .....                    | 4-4,4-43  |
| Steady State Deermination .....        | 5-2       | TIMER .....                            | B-4       |
| STEP .....                             | 4-46      | TITLE .....                            | C-6       |
| Step Responses .....                   | 8-3       | Title on Line Plots .....              | C-6       |
| Stiff Integration .....                | 4-3       | Top-of-Form .....                      | 4-32      |
| STOP .....                             | 5-12      | TRAN .....                             | 4-49      |
| Strip Chart Plot (STRPLT) .....        | C-2       | Transfer Block (XFERB) .....           | B-5       |
| STRPLT .....                           | C-2       | Transfer Function Macro Listing .....  | 4-50      |
| Subroutine Argument Order .....        | 4-8       | Transfer Function Representation ..... | 4-49      |



|                                        |               |                                       |      |
|----------------------------------------|---------------|---------------------------------------|------|
| Translator .....                       | 1-1           | Write Error Summary .....             | C-5  |
| Transport Delay .....                  | 4-13          | Write Integration Data (WRITG) .....  | B-5  |
| TRIM Subcommand .....                  | 5-2           | Write Integration Error Summary ..... | C-5  |
| TSMITG .....                           | C-5           | WRITG .....                           | B-5  |
| TTLCPL .....                           | C-3           | X-Axis Logarithmic Scaling .....      | 5-7  |
| TTLSPL .....                           | C-4           | X-Axis Low Value .....                | 5-7  |
| Two Sided Jacobean Evaluation .....    | C-5           | X-Axis Specification .....            | 5-7  |
| Type Statements .....                  | 4-49          | X-Axis Subcommand .....               | 5-7  |
| UNIF .....                             | 4-51          | XCICPL .....                          | C-3  |
| UNIFI .....                            | 4-23          | XCISPL .....                          | C-4  |
| Uniform Random Number (UNIF) .....     | 4-51          | XERROR .....                          | 4-34 |
| Upper Limit .....                      | 4-8,4-43,4-12 | XFERB .....                           | B-5  |
| User Defined Subroutine (SPARE) .....  | 5-12          | XHI Subcommand .....                  | 5-7  |
| User Integration Routine (INTEG) ..... | B-3           | XINCPL .....                          | C-3  |
| VAL Subcommand .....                   | 5-2           | XINSPL .....                          | C-4  |
| Value Get .....                        | B-1,B-3       | XLO Subcommand .....                  | 5-7  |
| Value Put .....                        | B-1,B-4       | XLOG Subcommand .....                 | 5-7  |
| VAR Subcommand .....                   | 5-2           | XTAG Subcommand .....                 | 5-7  |
| VARIABLE .....                         | 4-51          | XTICPL .....                          | C-3  |
| Variable Delay .....                   | 4-13          | XTISPL .....                          | C-4  |
| Variable Step Size Algorithm .....     | 4-3           | Y-Axis High Value .....               | 5-7  |
| Variables .....                        | 2-2           | Y-Axis Logarithmic Scaling .....      | 5-7  |
| Variables Undefined .....              | 7-1           | Y-Axis Low Value .....                | 5-7  |
| Vector Form .....                      | 4-5           | YASSPL .....                          | C-4  |
| Vector Integration .....               | 4-30          | YCICPL .....                          | C-3  |
| Vector Integrator Restrictions .....   | 4-30          | YCISPL .....                          | C-4  |
| VPUT .....                             | B-4           | YINCPL .....                          | C-3  |
| WEM .....                              | B-4           | YINSPL .....                          | C-4  |
| WESITG .....                           | C-5           | YTICPI .....                          | C-3  |
| Whole Part (AINT) .....                | 4-3           | YTISPL .....                          | C-4  |
| WRITE .....                            | 4-51          | Zero Order Hold .....                 | 4-51 |
| Write Error Messages (WEM) .....       | B-4           | ZHOLD .....                           | 4-51 |
|                                        |               | ZOH .....                             | 4-52 |