TYMSHARE MANUALS
TYMCOM-IX

CSMP

CONTINUOUS SYSTEM MODELING PROGRAM

MAY 1972

TYMSHARE, INC.
CUPERTINO, CALIFORNIA 95014

TYMSHARE ®

CONTENTS

rice: $3.50

TMB2

SECTION 1
INTRODUCTION

The Tymshare Continuous System Modeling Program, CSMP, simulates continuous processes on a digital computer. Such processes are encountered daily by engineers, scientists, and economists.

Continuous systems are usually described mathematically by a set of nonlinear differential equations which may be represented by a block diagram. CSMP includes a complete set of 39 functional elements and language statements with which the user may describe his system directly from the block diagram or the equations. CSMP handles problems with as many as 300 blocks, including 150 integrators and 10 function generators.

In addition, the CSMP user may easily define as many as 20 blocks to aid in the solution of his particular problem. The user creates these blocks with BATCH FORTRAN IV function subroutines. Such subroutines may access CSMP program variables through the use of global declarations, allowing the user to change parameters, integration algorithms, the integration interval, block specifications, and other values during program execution. The user may debug these subroutines in the BATCH FORTRAN mode of CSMP which permits him access to BATCH FORTRAN IV debugging features.

Tymshare CSMP offers the user numerous options and many outstanding new features. Six integration algorithms are available: an efficient variable step integration routine that automatically controls the solution accuracy, and five fixed step integration routines. Integration routines and intervals may be varied during program execution. Timing specifications may be selected and varied. At the end of a run, the outputs of any block in the configuration may be printed or plotted for any interval without recomputing the solution. In fact, the solution itself may be continued without recomputation. As many as ten block outputs may be plotted simultaneously on the same set of axes; plots may be scaled automatically or by the user. At any point, the user may store all program specifications and output data on a file and continue the analysis later.

CSMP contains convenient debugging commands, including the FIND command to locate all occurrences of specified block numbers or types. The use of command files within CSMP permits repeated execution with iteration through a range of parameter values.

CSMP and Tymshare are a powerful combination. The versatility of CSMP and the conversational nature of the Tymshare system provide an easy, fast, and accurate method of developing and testing continuous system models.

MANUAL ORGANIZATION

Section 2 of this manual illustrates the basic features of CSMP with a sample problem to solve a nonlinear differential equation. Section 3 describes the Tymshare CSMP elements, including user-defined blocks. Section 4 is a summary of the CSMP solution method and includes information about the various integration methods.

Sections 5, 6, and 7 describe the fundamental steps in solving a CSMP problem: entering the problem description, specifying the output data, and executing the problem. Section 8 details the CSMP file commands; Section 9 describes the CSMP scratch file. Section 10 contains a description of the debugging and utility commands.

Section 11 contains four sample problems, demonstrating many features of CSMP. The CSMP commands are summarized in Appendix A. Appendix B contains information about Laplace transform modeling in CSMP. Appendix C lists the decimal representations of ASCII characters.

SYMBOL CONVENTIONS

The symbols used in this manual for user-typed Carriage Returns, Alt Mode/Escapes, and control characters are:

Carriage Return:	↩
Alt Mode/Escape:	↴
Control character:	Superscript c. Thus, D^c denotes Control D.

In the examples in this manual, everything typed by the user is underlined. Lower-case letters used in an example of a command form represent the input which the user types. For example, the characters *file name* in a sample command form indicate that a legal file name should be typed at that point.

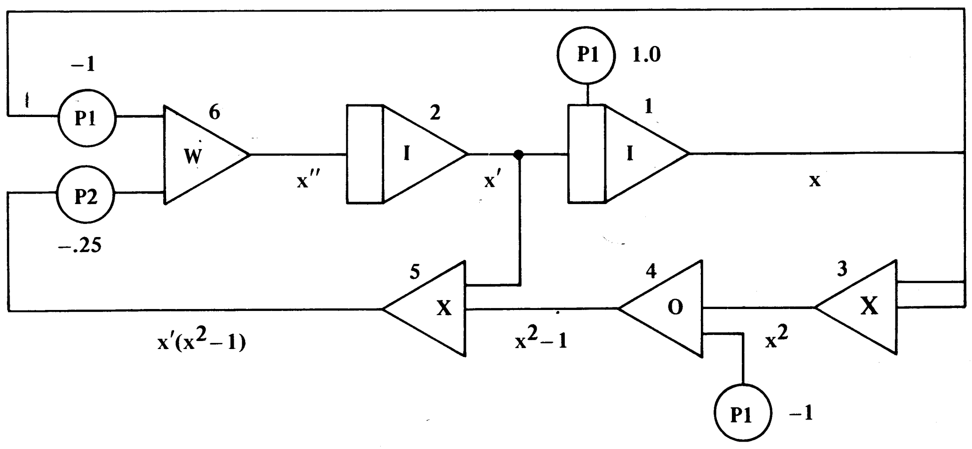
SECTION 2
SAMPLE CSMP PROBLEM

This section illustrates the ease with which CSMP problems may be written and solved by solving Van der Pol's nonlinear differential equation,

$$x'' + A(x^2-1)x' + Bx = 0$$

for $A=.25$, $B=1.0$, $x_0=1$, and $x'_0=0$. Calculations are made from time $t=0$ to 20 in increments of .5.

The block diagram is displayed below.



The problem entry and execution using CSMP are illustrated below.

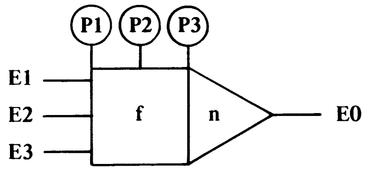
```
-CSMP↵

:BLOCKS↵

BLOCK TYPE E1  E2  E3

1 I 2↵
2 I 6↵
6 W 1  5↵
5 X 2  4↵
4 O 3↵
3 X 1  1↵
↵                  A Carriage Return terminates block input.
BEGIN SORT

:INITIAL↵

BLOCK P1  P2  P3

1 1.0↵
4 -1↵
6 -1  -.25↵ A Carriage Return terminates the entry of initial conditions and parameter values.
↵

:BY  .5  TO  20↵ The timing specifications are entered.

:PLOT 1↵
AUTOMATIC SCALING? NO↵
YMAX = 2.5↵
YMIN = -2.5↵
PLOT BLOCK: 2↵
AUTOMATIC SCALING? YES↵
PLOT BLOCK:↵        A Carriage Return terminates plot specifications.

:VARIABLE↵ The user chooses variable step integration.
```

: GO

```
BLOCK SYMBOL    MIN          MAX         INCREMENT
    1    +   -2.500E 00    2.500E 00    1.000E-01
    2    *   -1.924E 00    2.024E 00    7.895E-02

    TIME      +....+....+....+....+....+....+....+....+....+....+
   .000E 00                           * 0        +1
  5.000E-01                     *               +
  1.000E 00                *                +
  1.500E 00             *              +
  2.000E 00             *        +
  2.500E 00                 +*
  3.000E 00               +          *
  3.500E 00                +                 *
  4.000E 00                    +                   *
  4.500E 00                          +                  *
  5.000E 00                                 +            *
  5.500E 00                                        +*
  6.000E 00                                *          +
  6.500E 00                     *                    +
  7.000E 00             *                        +
  7.500E 00       *                        +
  8.000E 00    *                   +
  8.500E 00        *      +
  9.000E 00          +        *
  9.500E 00          +                    *
  1.000E 01             +                         *
  1.050E 01                   +                          *
  1.100E 01                            +                     *
  1.150E 01                                     +          *
  1.200E 01                                     *     +
  1.250E 01                          *                    +
  1.300E 01                *                           +
  1.350E 01        *                            +
  1.400E 01   *                         +
  1.450E 01  *                +
  1.500E 01           +
  1.550E 01        +                *
  1.600E 01         +                          *
  1.650E 01              +                            *
  1.700E 01                      +                          *
  1.750E 01                                +                   *
  1.800E 01                                         +    *
  1.850E 01                                *             +
  1.900E 01                    *                         +
  1.950E 01           *                              +
  2.000E 01     *                            +
              +....+....+....+....+....+....+....+....+....+....+
:DISPLAY ALL

BLOCK TYPE   E1     E2     E3       P1            P2            P3
  1    I     2      0      0    1.000E 00
  3    X     1      1      0
  4    O     3      0      0   -1.000E 00
  5    X     2      4      0
  6    W     1      5      0   -1.000E 00   -2.500E-01
  2    I     6      0      0

DT:   5.000E-02

OUTPUT
FROM:     .000E 00
BY:   5.000E-01
TO:   2.000E 01

RELERR:   1.000E-03

DTMIN:   5.000E-08

ALGORITHM: VARIABLE

DATA FILE
FROM:     .000E 00
BY:   5.000E-01
TO:   2.000E 01

:QUIT

-
```

SECTION 3
TYMSHARE CSMP ELEMENTS

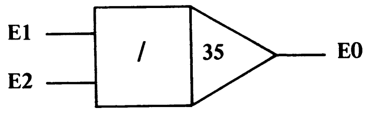
Tymshare CSMP provides a complement of 39 standard functional elements. In addition to the common simulation elements such as integrators, multipliers, and summers, a number of more specialized functional elements such as dead space, clipper, limiter, and zero-order hold devices are available to the user. Each of the CSMP element types specifies a functional relationship involving as many as three input variables and three parameters. (See the illustration below.)



E0 = f(E1,E2,E3,P1,P2,P3)

where E1, E2, and E3	are input variables.
P1, P2, and P3	are associated parameters.
n	is the block number.
f	indicates the functional relationship.
E0	is the output variable.

The output of each element type is a single-valued quantity defined by the particular relationship. For example, consider the divider block below.

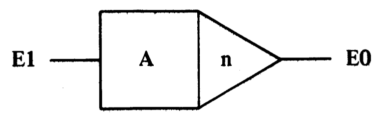
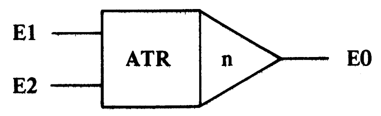
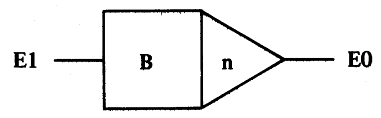
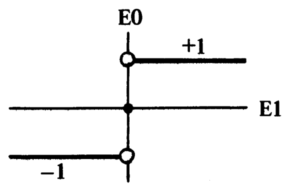
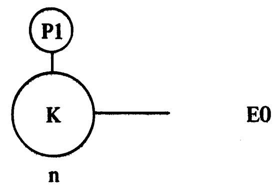


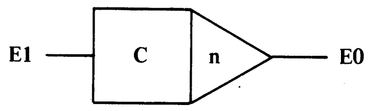
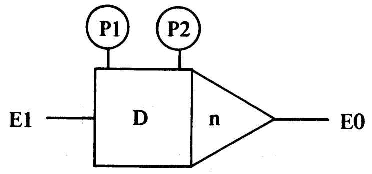
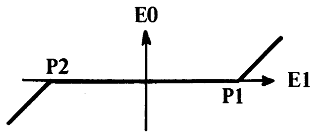
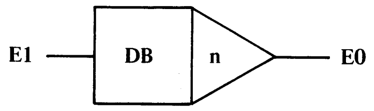
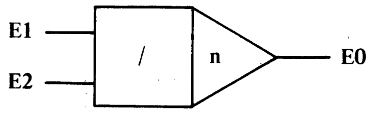
This block requires two input variables, E1 and E2. The functional relationship is E0=E1/E2. The input variable E3 and the parameters P1, P2, and P3 are not required for the divider block and should not be specified. In this example, the CSMP symbol is a slash, /, and the block number is arbitrarily assigned as 35.

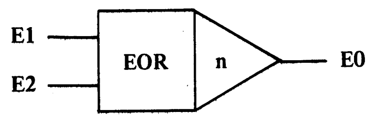
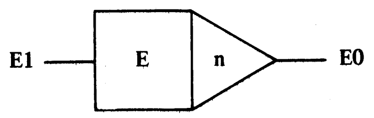
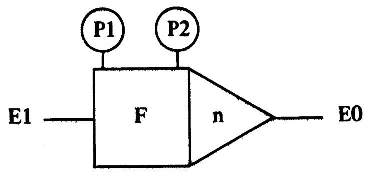
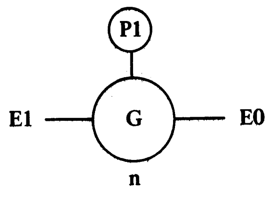
The table on the following pages is arranged in alphabetical order by function name. The table contains the CSMP elements, their CSMP and diagrammatic symbols, and the definitions of their functional operations. Notice that for those blocks that require only one input, for example, the cosine block, the input must be E1. Similarly, for those blocks requiring only two inputs, for example, the divider block, the inputs must be E1 and E2.

The variables and parameters used in the table are:

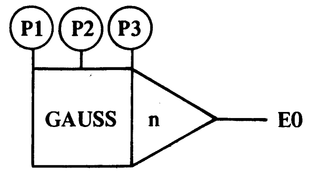
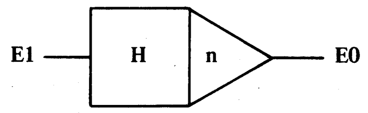
E1	=	input from first input block
E2	=	input from second input block
E3	=	input from third input block
E0	=	output
P1	=	initial condition or parameter 1
P2	=	parameter 2
P3	=	parameter 3
t	=	time
n	=	block number

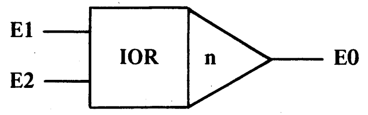
Name	CSMP Symbol	Diagrammatic Symbol	Description	Graph
Arctangent	A		$E0 = \arctan(E1)$ (E0 in radians)	
Arctangent2	ATR		$E0 = \arctan(E1/E2)$ $-\pi \leqslant E0 \leqslant \pi$ (E0 in radians)	
Bang bang	B		$E0 = +1$ *if* $E1 > 0$ $E0 = 0$ *if* $E1 = 0$ $E0 = -1$ *if* $E1 < 0$	
Constant	K		$E0 = P1$	

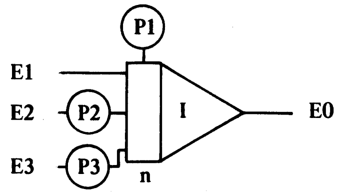
Name	CSMP Symbol	Diagrammatic Symbol	Description	Graph
Cosine	C		$E0 = \cos(E1)$ (E1 in radians)	
Dead space	D		$E0 = 0$ *if* $E1 = 0$ $E0 = \mathrm{MAX}(0, E1-P1)$ *if* $E1 > 0$ $E0 = \mathrm{MIN}(0, E1-P2)$ *if* $E1 < 0$	
Decibel	DB		$E0 = 10[\mathrm{LOG10}(E1^2)]$ ($E1 \neq 0$)	
Divider	/		$E0 = E1/E2$ ($E2 \neq 0$)	

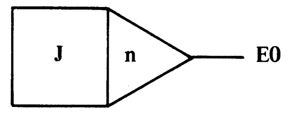
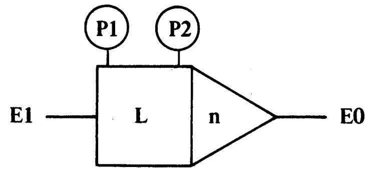
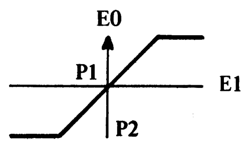
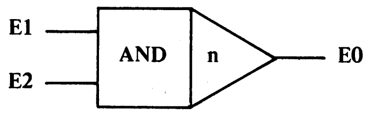
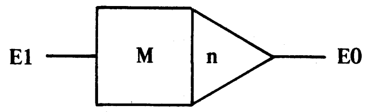
Name	CSMP Symbol	Diagrammatic Symbol	Description
Exclusive OR	EOR		$E0 = 1.0 \quad (E1 > 0$ *and* $E2 \leqslant 0$ *or* $E2 > 0$ *and* $E1 \leqslant 0)$ $E0 = 0.0$ *otherwise*
Exponential	E		$E0 = \exp(E1)$
Function generator	F		$E0 = f(E1)$ *Linear interpolation between 11 output points evenly spaced between P2 and P1.*[1]
Gain	G		$E0 = P1 \cdot E1$

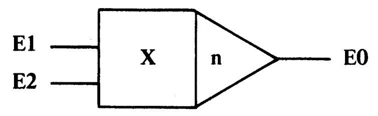
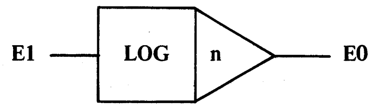
1 - The function generator block is detailed on page 19.

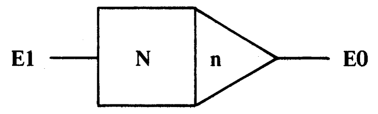
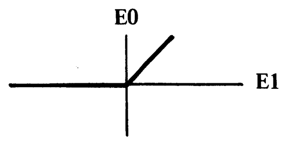
Name	CSMP Symbol	Diagrammatic Symbol	Description
Gaussian noise generator	GAUSS		E0 = 0.0 *for* $t < P3$ E0 is a random variable from a normal distribution *for* $t \geqslant P3$ *where* P1 is the mean. P2 is the standard deviation. P3 is a time delay.
Half power	H		$E0 = \sqrt{E1}$ $(E1 \geqslant 0)$

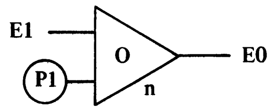
Inclusive OR	IOR		E0 = 1.0 (E1 > 0 *or* E2 > 0)
			E0 = 0.0 *otherwise*

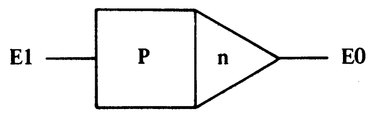
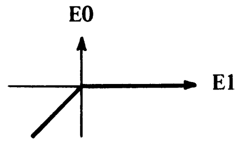
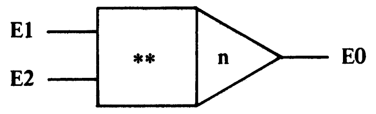
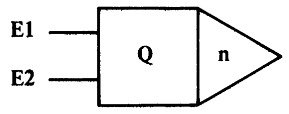
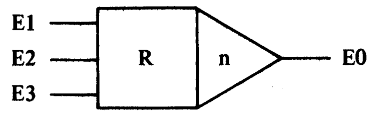
Integrator	I		$E0 = P1 + \int (E1 + P2 \cdot E2 + P3 \cdot E3)\, dt$

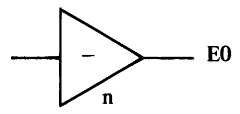
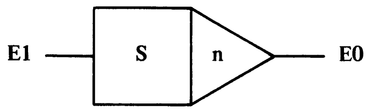
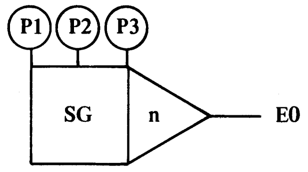
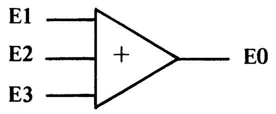
Name	CSMP Symbol	Diagrammatic Symbol	Description	Graph
Jitter	J		Random number generator between ±1	
Limiter	L		$E0 = E1$ *if* $P2 \leqslant E1 \leqslant P1$ $E0 = P1$ *if* $E1 > P1$ $E0 = P2$ *if* $E1 < P2$	
Logical AND	AND		$E0 = 1.0 \quad (E1 > 0$ *and* $E2 > 0)$ $E0 = 0.0$ *otherwise*	
Magnitude	M		$E0 = ABS(E1)$	

Name	CSMP Symbol	Diagrammatic Symbol	Description	Graph
Multiplier	X		E0 = E1 · E2	
Natural logarithm	LOG		E0 = ALOG(E1) (E1 > 0)	

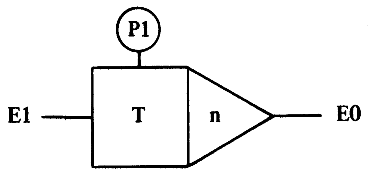
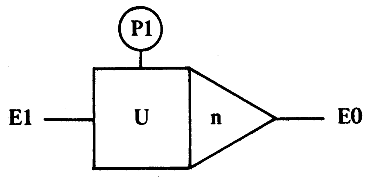
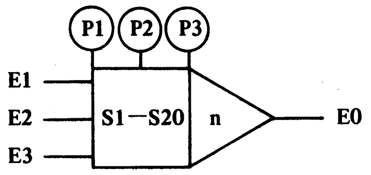
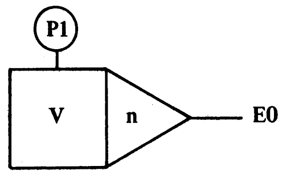
Negative clipper	N		E0 = 0 *if* E1 ⩽ 0
			E0 = E1 *if* E1 > 0
				

Offset	O		E0 = E1 + P1	

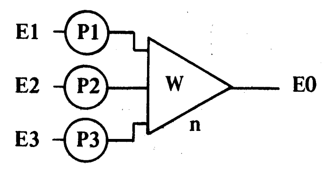
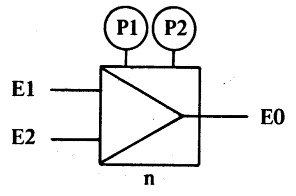
Name	CSMP Symbol	Diagrammatic Symbol	Description	Graph
Positive clipper	P		$E0 = 0 \quad \textit{if} \quad E1 \geqslant 0$ $E0 = E1 \quad \textit{if} \quad E1 < 0$	
Power	**		$E0 = (E1)^{E2}$ $(E1 \geqslant 0)$	
Quit	Q		Terminates run if $E1 > E2$	
Relay	R		$E0 = E2 \quad \textit{if} \quad E1 \geqslant 0$ $E0 = E3 \quad \textit{if} \quad E1 < 0$	

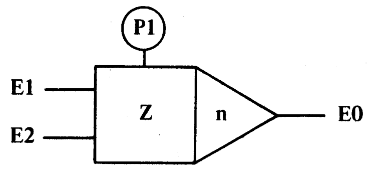
Name	CSMP Symbol	Diagrammatic Symbol	Description
Sign inverter	–		$E0 = -E1$
Sine	S		$E0 = \sin(E1)$ (E1 in radians)
Sine wave generator	SG		$E0 = 0.0$ *for* $t < P3$ $E0 = \sin[2\pi P1 \cdot (t - P3) + P2]$ *for* $t \geqslant P3$ *where* $P1$ = Frequency (hertz) $P2$ = Phase shift (degrees) $P3$ = Time delay
Summer[1]	+		$E0 = \pm E1 \pm E2 \pm E3$

1 - The Summer block is the only block which may specify negative block input values.

Name	CSMP Symbol	Diagrammatic Symbol	Description
Time pulse generator	T		Generates pulse train with period equal to P1. First pulse occurs when E1 $\geqslant$ 0. Magnitude of pulse is equal to 1. E0 $=$ 0 *when* E1 $<$ 0. To restart pulse train after E1 has been less than 0, E1 must be made equal to or greater than 0.
Unit delay	U		E0 $=$ P1 *when* t $=$ 0 E0 $=$ E1 *at* t$-$DT *when* t $>$ 0[1]
User defined	S1$-$S20		User-defined block
Vacuous	V		Used with wye element[2]

1 - DT, the integration interval, is described on page 35.
2 - The wye and vacuous blocks are discussed on page 19.

Name	CSMP Symbol	Diagrammatic Symbol	Description
Weighted summer	W		$E0 = P1 \cdot E1 + P2 \cdot E2 + P3 \cdot E3$
Wye	Y		Logical branch element, used for implicit operations.[1]

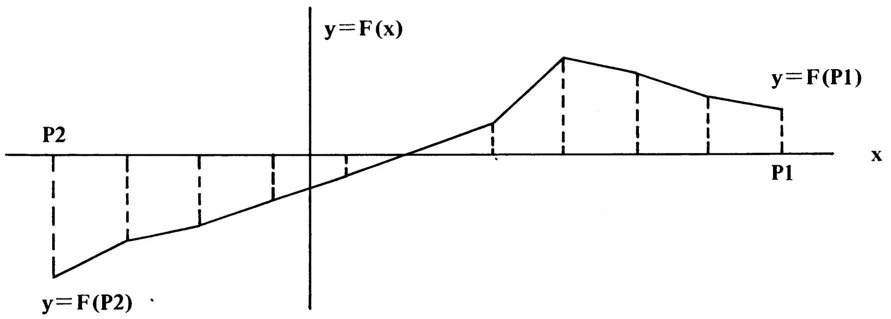
Zero order hold	Z		E0 = E1 *for* $E2 > 0$
			E0 unchanged *for* $E2 \leqslant 0$
			E0 = P1 *at* $t = 0$

1 - The wye and vacuous blocks are discussed on page 19.

FUNCTION GENERATOR BLOCKS

The function generator is used to simulate parts of the model where some relationship, Y=f(X), is available in tabular or graphic form. The relationship to be simulated must be one that can be approximated adequately by ten straight-line segments evenly spaced between the minimum and maximum values of X, specified by P2 and P1, respectively. Y can be a function of time; that is, block 301 can be the input to the function generator block.

The diagram below illustrates a functional relationship which a CSMP user might wish to express by means of a function generator block.



The CSMP user may include as many as ten different function generator blocks in the same problem. If the user does not enter the intercept values at the time he specifies the block, he may enter them with the FUNCTION command. For example, if the block data is on a file, the intercept values may not have been included. The FUNCTION command may also be used to modify one or more intercept values. To modify any one of the intercept values, the user must reenter all the intercept values for that function generator.

The user may display the current intercept values for a function generator block by giving the DISPLAY command. DISPLAY n, where n is the number of a function generator block, prints the basic block information, including parameters, in addition to the intercept values.

```
:DISPLAY FUNCTION↵
```

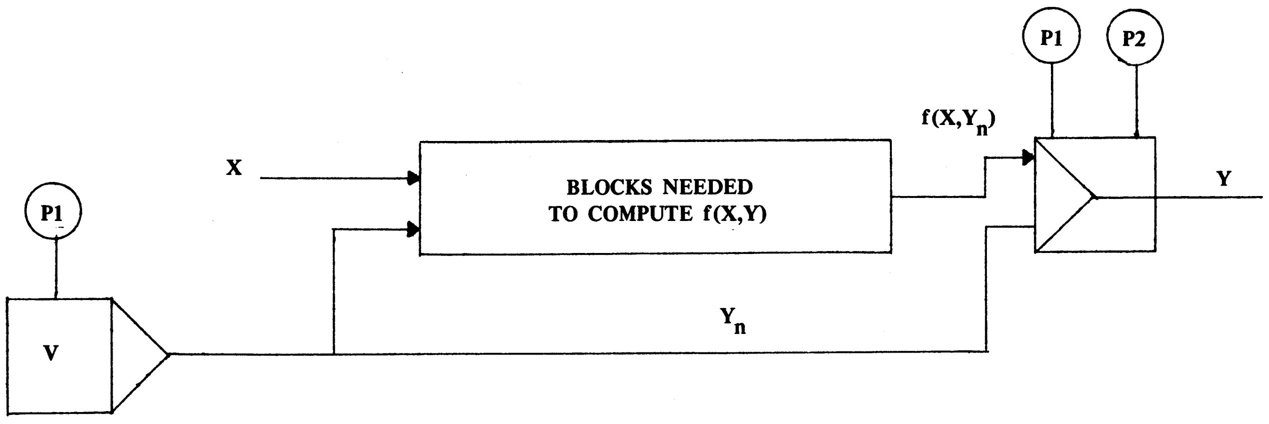
prints the number and intercept values for all function generators in the current problem.

Example 3, page 78, demonstrates the use of a function generator block.

THE WYE AND VACUOUS BLOCKS

The wye and vacuous blocks are used to solve implicit equations of the form Y=f(X,Y). The wye block has two parameters: P1 is the permissible relative error; P2 is the convergence factor. The vacuous block has one parameter, the initial estimate of the output.

The wye and vacuous blocks are used as follows:



The initial value, Y_1, is supplied by the parameter of the vacuous block. The wye and vacuous blocks then cause Y to be iterated until

$$\left| \frac{f(X,Y_n)}{f(X,Y_{n-1})} - 1 \right| \leqslant \epsilon$$

where ϵ equals P1, the first parameter of the wye block. When this criterion is met, $Y = Y_n = f(X,Y_n)$.

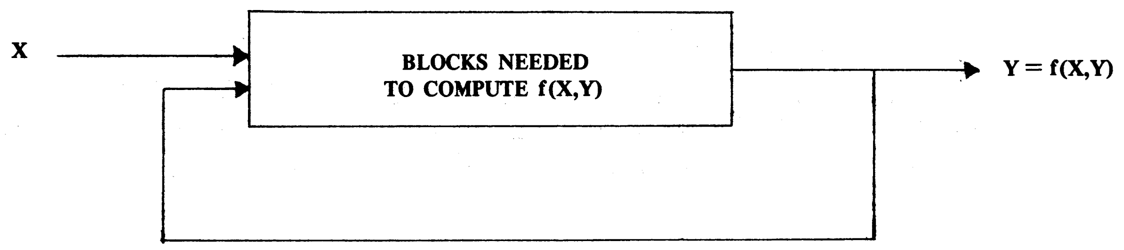
If the error criterion is not met, a new Y value is calculated as follows:

$$Y_{n+1} = (1.0 - P2)\ f(X,Y_n) + P2 \cdot Y_n$$

where $0 < P2 \leqslant 1$ and is the second parameter associated with the wye block.

It is necessary to experiment with various values of the parameters of both the wye and vacuous blocks in order to achieve convergence. There is no limit set for the number of iterations allowed to achieve convergence. Therefore, these blocks should be used with caution.

Consider the following configuration:



This configuration contains an algebraic loop and will give a sort failure error message. However, the wye and vacuous blocks may be used to accommodate this algebraic loop as discussed above.

UNIT DELAY BLOCKS

The unit delay block is used to delay the input to the block by DT, the integration interval.[1] However, since the amount of delay is a function of the integration interval, the delay block may not be used with the variable step integration routine.

USER-DEFINED BLOCKS

The most powerful block in CSMP is the user-defined block. This block makes it possible to combine BATCH FORTRAN IV function subroutines with user-defined subroutines and the CSMP language statements. User-defined blocks should be used whenever possible; they often reduce drastically the number of blocks in a configuration. Since execution speed is proportional to the number of blocks used, a reduction in the number of blocks is reflected in the amount of computer resources used. Tymshare CSMP allows as many as 20 different user-defined blocks in a single configuration.

User-defined blocks are extremely convenient. With approximately 2100 words of storage available for user-defined subroutines, the user may create programs far larger than would be possible with individual blocks alone.

For example, assume the following relationship is to be modeled in CSMP:

$$\mathrm{E0} = \tan^{-1}\left\{\frac{\mathrm{E1}\cdot\mathrm{P1}}{[(\mathrm{E1}\cdot\mathrm{P1})^2 + (\mathrm{E2}\cdot\mathrm{P2})^2 + (\mathrm{E3}\cdot\mathrm{P3})^2]^{1/2}}\right\}$$

Where E1, E2, and E3 are outputs of blocks 1, 2, and 3, respectively. The CSMP configuration might be written as follows:

```
 4  G   1
 5  G   2
 6  G   3
 7  X   4   4
 8  X   5   5
 9  X   6   6
10  +   7   8   9
11  H  10
12  /   4  11
13  A  12
```

1 - The integration interval, DT, is described on page 35.

Ten blocks were used to derive the desired result, block 13. However, one user-supplied block achieves the same result. For example, the following function subroutine generates the same result as the previous configuration.

```
FUNCTION S1(E1,E2,E3,P1,P2,P3)
A = E1 * P1
B = E2 * P2
C = E3 * P3
D = A/SQRT(A * A + B * B + C * C)
S1 = ATAN(D)
RETURN
END
```

The SUBROUTINES command loads into CSMP the function subroutines which are to be used as user-defined blocks. This command may be given at any time.

Before he calls CSMP, the user must compile all the function subroutines in BATCH FORTRAN IV onto a file named CSMPSUBS. The function subroutine names must be

```
FUNCTION S1(E1,E2,E3,P1,P2,P3)
FUNCTION S2(E1,E2,E3,P1,P2,P3)
      :
FUNCTION S20(E1,E2,E3,P1,P2,P3)
```

where E1, E2, and E3 are input variables.
P1, P2, and P3 are parameters.

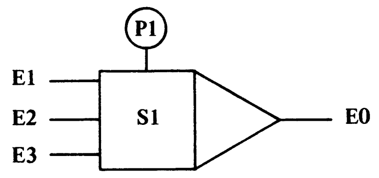
Note that the argument list for user-defined function subroutines is block oriented, containing three inputs and three parameters. This does not limit the ability to incorporate other user-defined routines into CSMP. A user may call any number of subroutines within the user-defined function subroutine as long as he compiles them on the file CSMPSUBS.

For example, if a user has written a function subroutine which generates Bessel functions and wishes to use it in CSMP, he might do it as follows:

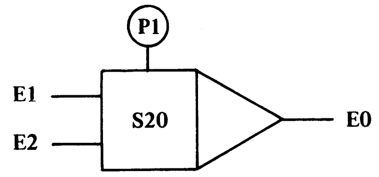
```
FUNCTION S2(E1,E2,E3,P1,P2,P3)
S2 = BESSEL(E1,P1)
RETURN
END

FUNCTION BESSEL(X,A)
:
END
```

An example illustrates the ease with which special blocks or subroutines may be incorporated into CSMP.

The user wishes to define two new blocks, S1 and S20:



$$E0 = \frac{(E1)^2 + (E2)^2 + (E3)^2 - \cos(2E1 \cdot E2)}{\sin[P1 \cdot E2 - \mathrm{atan}(E1/E3)]}$$



$$E0 = \left[\sin(E1 \cdot E2) \cdot e^{P1 \cdot E1}\right]$$

The user creates his own BATCH FORTRAN IV subroutine in Tymshare's EDITOR.[1]

```
-EDITOR↵
*APPEND↵
FUNCTION S1(E1,E2,E3,P1,P2,P3)↵
A=SQRT(E1*E1+E2*E2+E3*E3-COS(2.*E1*E2))↵
B=SIN(E2*P1-ATAN(E1,E3))↵
S1=A/B↵
RETURN↵
END↵
FUNCTION S20(E1,E2,E3,P1,P2,P3)↵
S20=SIN(E1*E2)*EXP(P1*E1)↵
RETURN↵
END↵
*WRITE SUBS↵   The user terminates the APPEND command with a Control D.
 NEW FILE↵
183 CHARACTERS
*QUIT↵

-BFORTRAN↵

+COMPILE SUBS,CSMPSUBS↵  The file SUBS is compiled to the file CSMPSUBS.[2]
 OLD FILE↵   The previous contents of file CSMPSUBS are replaced.

FUNCTION S1(E1,E2,E3,P1,P2,P3)
END

FUNCTION S20(E1,E2,E3,P1,P2,P3)
END

+QUIT↵
```

1 - For further details about EDITOR, consult the *Tymshare EDITOR Reference Manual*.

2 - When using user-defined blocks, the user may obtain additional space if he gives the OFF DEBUG and OFF ARG commands in BATCH FORTRAN IV before compiling the subroutines. In this case, he cannot use the CSMP DEBUG command, described on page 25.

```
-CSMP ↵            The user calls CSMP and indicates by the command SUBROUTINES that
                   he wishes to use his previously compiled subroutines in this problem.
: SUBROUTINES ↵

LOADING LIBRARY

WORDS USED
  PROGRAM: 5774
  STORAGE: 2193
  SHARED:  5550
  DEBUG:   179
WORDS UNUSED            The unused words are available for user-defined blocks.
  PROGRAM: 2113
  SHARED:  594

:
```

If the file CSMPSUBS does not contain all the required subroutines, the names of the missing subroutines are listed after the message LOADING LIBRARY. The file CSMPSUBS must be recompiled with the missing subroutines.

The SUBROUTINES command creates in the user's directory the binary file CSMP'SCR3'. CSMP automatically deletes the file CSMP'SCR3' when the user types QUIT. Because it is convenient and efficient to load the same subroutines later without reusing the SUBROUTINES command, the user may write the contents of the file CSMP'SCR3' onto another file with the DUMP SUBROUTINES command. The form of this command is

:DUMP SUBROUTINES file name ↵

which duplicates the contents of the file CSMP'SCR3' on the specified file.

The complementary command

:RECOVER SUBROUTINES file name ↵

loads the specified subroutine file previously created with the DUMP SUBROUTINES command, allowing the user to keep conveniently any number of subroutine files in his directory.

Note that it is many times more efficient to use the DUMP and RECOVER commands to load subroutines than to reuse the SUBROUTINES command. The SUBROUTINES command must be used the first time the subroutines are used.

Debugging User-Defined Blocks

The user may debug his special function subroutines within CSMP using the powerful debugging capability of BATCH FORTRAN IV. For example, if any error is detected in a user block, the computer prints an error message, the statement in which the error occurred, and a plus sign, +, indicating that the user is in the BATCH FORTRAN IV mode of CSMP. When the plus sign appears, the user may use any of the debugging commands available in BATCH FORTRAN IV. For example, the follcwing might occur in the user block S1, defined on page 23.

```
ERROR-->NEGATIVE ARGUMENT TO SQUARE ROOT. ABSOLUTE VALUE USED.
STATEMENT S1,0+1
+DISPLAY E1,E2,E3 ↵
0
0
0

+
```

To return to the CSMP mode, denoted by the colon, the user must give the SWAP CSMP and GO commands. For example,

```
+SWAP CSMP ↵

+GO ↵

:
```

The user may transfer to BATCH FORTRAN IV mode from CSMP by giving the DEBUG command. He returns to CSMP mode by giving the SWAP CSMP and GO commands described above.

In the BATCH FORTRAN mode, he can set breakpoints, change variable values, and display variable values. For example,

```
:DEBUG ↵

+DEBUG S1 ↵
+LET A=0.0 ↵
+BREAK 40+3 ↵
+DEBUG S11 ↵
+BREAK 10 ↵
+SWAP CSMP ↵

+GO ↵

:
```

Accessing Program Variables

User-defined function subroutines permit the CSMP user to change parameters, integration routines, the integration interval, and the block specifications during program execution.

All of the major variables in CSMP are global. This feature allows the user to access or modify program specifications during execution via a user-defined subroutine.

Below is a partial list of variables which are accessible through a GLOBAL declaration statement in the user's subroutine.

Variable Name	Description
T	Time; equal to the output of block 301.
DT	Same as in program.[1]
DTMIN	Same as in program.[1]
RELERR	Same as in program.[1]
FROM	Same as in program.[1]
BY	Same as in program.[1]
TO	Same as in program.[1]
E0(1) to E0(300)	Block output; E0(N) is the output of block N.[2]
PAR1(1) to PAR1(300)	Parameter 1; PAR1(N) is the first parameter for block N.
PAR2(1) to PAR2(300)	Parameter 2.
PAR3(1) to PAR3(300)	Parameter 3.

NOTE: In the GLOBAL declaration statement, the arrays E0, PAR1, PAR2, and PAR3 must be dimensioned to contain 300 elements if the user wishes to access an element of that array in his function subroutine.

If the user chooses to modify the BY variable during program execution, he should not attempt to read intervals contained on the scratch file.[3] Data on the scratch file is located by multiplying the number of points contained in the interval by the BY value. The user should erase the data from the scratch file with the ERASE command at the end of any run in which he has changed the BY value during execution.

1 - The quantities DT, DTMIN, RELERR, FROM, BY, and TO are described on page 42.

2 - E0 is the letter E followed by the digit 0.

3 - The CSMP scratch file is described in Section 9.

If DT is changed by a subroutine during execution, the following GLOBAL variables must also be changed:

DTS2 = DT/2

DTS3 = DT/3

DTS23 = DT * 2/3

DTS6 = DT/6

DTMIN should be set to a value less than DT.

If RELERR is changed within a subroutine, the GLOBAL variable RELER8 should be set to RELERR/8.

The CSMP user may change the integration algorithm during execution by changing the value of the GLOBAL array element NTEST(9).[1] The table below lists the algorithms denoted by the values of NTEST(9).

NTEST(9) Value	Integration Algorithm
1	RK2
2	Trapezoidal
3	Euler
4	Simpson's
5	RK4
6	Variable

Note that DT, its associated parameters, and the integration algorithm may be changed only if NTEST(5) = 3. The values of the variables RELERR and RELER8 may be changed at any time.

Example 1

The following function S12 calls the subroutine DTCHANGE to change the value of DT if the output of block 24 is greater than the output of block 146.

```
      FUNCTION S12(E1,E2,E3,P1,P2,P3)
      GLOBAL NTEST(9),EO(300)
*CHECK TO SEE THAT DT CAN BE CHANGED
      IF(NTEST(5).NE.3) RETURN
*IF NTEST(5) = 3 CHECK TO SEE THAT DT SHOULD BE CHANGED
      IF(EO(24).GT.EO(146)) CALL DTCHANGE
      RETURN
      END
```

1 - The integration algorithms available in CSMP are described on page 34.

```
SUBROUTINE DTCHANGE
GLOBAL DT,DTS2,DTS3,DTS23,DTS6
DT=DT*.5
DTS2=DT/2
DTS3=DT/3
DTS23=DT*2/3
DTS6=DT/6
RETURN
END
```

Example 2

The following subroutine changes the integration algorithm during program execution.

```
      FUNCTION S1(E1,E2,E3,P1,P2,P3)
      GLOBAL T,NTEST(9),PAR1(300)
*
      IF(NTEST(5).NE.3) RETURN
      IF(T.LT.P1)RETURN
      PAR1(10)=PAR1(10)+.25
      IF(NTEST(9).EQ.2) NTEST(9)=4; GO TO 10
      NTEST(9)=2
10    RETURN
      END
```

Restrictions on User-Defined Blocks

There are three important restrictions the CSMP user must remember when defining his own blocks with BATCH FORTRAN IV function subroutines. These restrictions concern unit numbers for files, transferring between CSMP and the BATCH FORTRAN mode, and the BATCH FORTRAN syntax allowed in CSMP function subroutines.

Units 3, 10, 11, 12, and 14 may not be used to open files within a CSMP user-written subroutine. These unit numbers are reserved for use within CSMP.

If the CSMP user transfers control from the debug mode to CSMP with the SWAP CSMP and GO commands, he must remember that all data has been erased from the scratch file. Thus, to begin execution, the CSMP GO command must be given; the CSMP CONTINUE command cannot be used.[1]

User-written subroutines for CSMP may employ all features of BATCH FORTRAN IV except ACCEPT, DISPLAY, and statements involving a FORMAT. CSMP contains special routines to perform the same functions without loading the BATCH FORTRAN IV format processor. This makes possible the increased space for CSMP programs.

1 - The GO and CONTINUE commands are described in Section 7.

Because FORMAT statements are not allowed, the following are illegal statements in CSMP subroutines:

```
TYPE f,l
ACCEPT f,l
WRITE(u,f)l
READ(u,f)l
```

where f is a format reference.
l is a variable list.
u is a unit number.

Note that binary READ and WRITE statements are legal. For example, if unit number u has been opened as a sequential or random binary file, the following are legal statements:

```
READ(u)l
WRITE(u)l
```

Special Format Functions and Subroutines

CSMP contains special subroutines and functions to perform formatted output. The OUTRFMT function provides formatted output for real or floating point numbers; the complementary OUTIFMT function operates on integers. The DISPLAY and OUTPUT subroutines print strings on the terminal and the designated file, respectively. The DISCHR and OUTCHR subroutines print single characters on the terminal and the designated file, respectively.

To print a real number, the CSMP user includes in his subroutine the statement

```
NUMBER = OUTRFMT(u,v,t,w,d)
```

where u is the unit number.
v is the name of the variable to be printed.
t is an integer representing the type of format.
t=1 denotes I format.
t=2 denotes F format.
t=3 denotes E format.
t=4 denotes G format.
w is an integer from 1 to 14 denoting the width of the output field. This corresponds precisely to the field width in FORTRAN FORMAT statements.
d is an integer representing the number of decimal places to be printed. Note that d should be 0 if t = 1.

The OUTIFMT function prints integers. The function must be placed in the program as

```
NUMBER = OUTIFMT(u,v,t,w,d)
```

where u, v, t, w, and d are described as above.

To perform string output, the CSMP user should call the DISPLAY or OUTPUT subroutine. The DISPLAY subroutine prints the specified string on the terminal; OUTPUT writes the string on a file. The forms of the calls to this subroutine are

CALL DISPLAY('s/')

CALL OUTPUT(u,'s/')

where s is the string.

A string may consist of an ASCII character sequence of any length provided it does not contain the character ' or "; however, if one of the characters &, $, or / is in the string, it must be preceded by an ampersand, &. For example,

CALL DISPLAY('AB&$CD&&EG&/F/')

prints

AB$CD&EG/F

on the terminal.

A dollar sign, $, not preceded by an ampersand, &, in a string is interpreted as a Carriage Return. For example,

CALL DISPLAY('ABC$DEF/')

prints ABC on the first line and DEF on the next line.

The CSMP user may print single characters with the DISCHR and OUTCHR subroutines. The calling sequences for these subroutines are

CALL DISCHR(n)

CALL OUTCHR(u,n)

where u is a unit number.

n is an integer between 0 and 127, inclusive, equal to the decimal representation of the ASCII character.[1]

The DISCHR subroutine writes a single character on the terminal; OUTCHR writes the specified character on the file opened as unit n.

1 - A table of ASCII characters and their decimal representations is included in Appendix C.

Example

The following user-defined subroutine produces the output listed below it.

```
        I=-534
        CALL DISPLAY('$I =/')
        NUMBER=OUTIFMT(1,I,1,5,1)
        CALL DISPLAY('$/')

        A=3453.34945E2
        CALL DISPLAY('$OUTPUT A TO THE TERMINAL$/')
        DO 10 ITYPE=1,4
        NUMBER=OUTRFMT(1,A,ITYPE,14,5)
        CALL OUTPUT(1,'$/')
10      CONTINUE

        CALL DISPLAY('$HERE ARE ASCII CHARACTERS FROM 1 TO 63$/')
        DO 20 N=1,63
        CALL OUTCHR(1,N)
20      CONTINUE
        CALL DISPLAY('$/')
        END
```

The above subroutine produces this output:

```
I = -534

OUTPUT A TO THE TERMINAL
        345334
  345334.94500
    .34533E 06
  345334.94500

HERE ARE ASCII CHARACTERS FROM 1 TO 63
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑←
```

```
: DO DAMPING
```
CSMP now takes commands from the file DAMPING.

```
BLOCK P1 P2 P3
```
P1 of block 5 is changed to 1.

```
BLOCK SYMBOL     MIN            MAX         INCREMENT
    1    +    -1.000E 00     1.000E 00     4.000E-02
    2    *    -3.322E-01     6.698E-01     2.004E-02

     TIME    +....+....+....+....+....+....+....+....+....+....+
   .000E 00  +                *
  2.500E-01   +                           *
  5.000E-01     +                                   *
  7.500E-01        +                                       *
  1.000E 00            +                                      *
  1.250E 00                +                                   *
  1.500E 00                    +                              *
  1.750E 00                        +                       *
  2.000E 00                            +                *
  2.250E 00                               +         *
  2.500E 00                                 +  *
  2.750E 00                               *   +
  3.000E 00                          *          +
  3.250E 00                     *               +
  3.500E 00                *                    +
  3.750E 00           *                         +
  4.000E 00       *                           +
  4.250E 00    *                             +
  4.500E 00   *                            +
  4.750E 00  *                           +
  5.000E 00  *                         +
  5.250E 00    *                     +
  5.500E 00      *                 +
  5.750E 00        *              +
  6.000E 00           *          +
  6.250E 00              *      +
  6.500E 00                  * +
  6.750E 00                     +
  7.000E 00                     +  *
  7.250E 00                      +    *
  7.500E 00                       +    *
  7.750E 00                        +     *
  8.000E 00                          +   *
  8.250E 00                           +  *
  8.500E 00                            +*
  8.750E 00                           *  +
  9.000E 00                         *     +
  9.250E 00                       *       +
  9.500E 00                     *          +
  9.750E 00                   *            +
  1.000E 01                *               +
             +....+....+....+....+....+....+....+....+....+....+
BLOCK P1 P2 P3
```
P1 is set to 2.

```
BLOCK SYMBOL    MIN          MAX          INCREMENT
    1     +  -1.000E 00   1.000E 00    4.000E-02
    2     *  -1.886E-01   5.449E-01    1.467E-02

    TIME      +....+....+....+....+....+....+....+....+....+....+
   .000E 00   +            *
  2.500E-01    +                           *
  5.000E-01      +                                     *
  7.500E-01         +                                         *
  1.000E 00            +                                        *
  1.250E 00                +                                    *
  1.500E 00                   +                               *
  1.750E 00                      +                         *
  2.000E 00                         +                  *
  2.250E 00                           +             *
  2.500E 00                             +       *
  2.750E 00                               + *
  3.000E 00                             *  +
  3.250E 00                         *       +
  3.500E 00                    *             +
  3.750E 00                *                 +
  4.000E 00            *                     +
  4.250E 00         *                       +
  4.500E 00      *                          +
  4.750E 00    *                           +
  5.000E 00   *                          +
  5.250E 00   *                         +
  5.500E 00    *                       +
  5.750E 00     *                     +
  6.000E 00       *                  +
  6.250E 00         *               +
  6.500E 00           *             +
  6.750E 00             *           +
  7.000E 00                *       +
  7.250E 00                  *      +
  7.500E 00                     *   +
  7.750E 00                      *  +
  8.000E 00                        * +
  8.250E 00                        *  +
  8.500E 00                        *   +
  8.750E 00                        *   +
  9.000E 00                       *     +
  9.250E 00                      *       +
  9.500E 00                    *         +
  9.750E 00                   *          +
  1.000E 01                 *             +
              +....+....+....+....+....+....+....+....+....+....+
```

SECTION 4
THE CSMP PROBLEM: DESCRIPTION AND SOLUTION

This section describes the procedure for solving a CSMP problem. The user must first define the problem with CSMP elements and statements. He then chooses one of the six integration algorithms available in CSMP and detailed in this section.

PROBLEM DEFINITION

Tymshare CSMP is commonly used as a general differential equation solver in which the set of linear or nonlinear differential equations is specified by connecting the blocks in the proper fashion. The CSMP user begins by analyzing the problem to be solved and developing a block diagram showing the interconnections of the elements. Each block on the simulation diagram is identified by a block number, arbitrarily assigned within the range 1 to 300. This number is usually written adjacent to or within the diagrammatic symbol. In the table of CSMP elements, the block number for each statement is represented by the letter n.

By definition, the independent variable, time, is contained in block 301 and may be used as an input to any other block.

After the user has generated a block diagram, he translates it into a program by using CSMP language statements. The two types of language statements are as follows:

1. **Configuration statements** define interconnections of the blocks and specify the desired functional operation. For example,

   ```
   1 G 301
   ```

   specifies a gain block. Its block number is 1, and its input is the time block, 301.

2. **Parameter statements** assign numerical constants to the block elements to particularize their functional operation. At most, three initial conditions or parameters may be associated with each block. In block types that have memory, such as the integrator, unit delay, and vacuous blocks, the first parameter, P1, is used to specify the initial condition, the value of the output at time t=0. All initial conditions or parameters which are not specified by the user are automatically set to 0. For example,

   ```
   1 -.5
   ```

   specifies that the first parameter, P1, is −.5 for block 1. P2 and P3 are set to 0.

Each block in the model requires a configuration statement. Blocks which have associated parameters in their definition may also require a parameter statement. For example, the gain element must have a parameter statement in order to set P1 equal to a non-zero value. Otherwise, the output of the gain block would always be 0.

The function generator element requires information in addition to the statements described above. The configuration statement specifies the interconnections; the parameter statement specifies the maximum and minimum values of the input variable. The user must then specify the value of the output variable at each of the 11 intercept points.

The CSMP problem size limits are:

300 blocks

150 integrators

10 function generators

50 unit delays

50 zero order hold blocks

50 time pulse generators

20 user-defined subroutines

1500 output time intervals

Unlike some digital programs in which the order of coding is important, the order in which the configuration statements are entered into Tymshare CSMP does not affect the solution. After each change in the simulation configuration, a sorting algorithm determines the proper order in which to evaluate the new set of functional elements. No block can be processed until updated values of its input variables are available. At each time interval, the output of all the memory elements and certain constants are assumed to be available. By using these outputs and constants, one or more additional blocks can be processed by the sorting algorithm. Sorting is performed automatically and requires just a few seconds. A diagnostic message is printed if the sorting algorithm indicates an inconsistent configuration. The most common cause of such an error is an algebraic loop which is a closed pathway in the configuration and does not include one of the two memory elements: integrator and unit delay. If a simulation requires an implicit operation, the wye and vacuous elements should be used to break the resulting algebraic loop.[1]

CSMP SOLUTION METHODS

CSMP uses six different centralized integration routines which are listed below in increasing order of sophistication. They are specified at CSMP command level by the commands given in parentheses.

1. Euler (EULER)
2. Trapezoidal (TRAPEZOIDAL)
3. Runge-Kutta second order (RK2)
4. Simpson's Rule (SIMPSONS)
5. Runge-Kutta fourth order (RK4)
6. Variable step size (VARIABLE)

If the user does not specify a routine, RK2 is used. The user may change the integration routine or the integration interval during the solution of a problem and continue the computation with the CONTINUE command.

The first five routines have a fixed step size; the user supplies the step size, or integration interval requested by CSMP, as the parameter DT. The sixth routine is a variable step size Runge-Kutta third order routine. The integration interval, DT, is automatically adjusted to meet a user-supplied relative error bound, RELERR, on the solution accuracy.

1 - The wye and vacuous elements are described on page 19.

The goal in choosing an integration algorithm is to find one which offers the necessary accuracy, numerical stability, and the shortest execution time. The selection of the best routine for a particular problem is arrived at by experimentation with the algorithm parameters DT and RELERR. RELERR affects the solution accuracy only with the VARIABLE integration routine.

The user may request that different integration algorithms be used in different stages of a solution. See page 28 for an example of accomplishing this by means of user-defined subroutines.

The third order Runge-Kutta algorithm was selected for the variable step algorithm because its stability is nearly that of the higher order routines. Furthermore, the test to determine if DT should be increased or decreased is much more efficient than with higher order routines.

The variable step routine has the advantage of providing a stable solution for a specified numerical accuracy. However, if the accuracy criterion RELERR is set too stringently, execution time will be excessive. Execution time will likewise be excessive for fixed step routines if the step size DT chosen is too small.

The mathematics of the routines is as follows. Assume a set of differential equations expressed in the "state variable" form,

$$\frac{dy}{dt} = y' = f(y,t)$$

where y, y′, and f are vectors representing a set of k first order differential equations,

$$\begin{aligned} y_1' &= f_1(y_1, y_2, \ldots, y_k, t) \\ y_2' &= f_2(y_1, y_2, \ldots, y_k, t) \\ &\vdots \\ y_k' &= f_k(y_1, y_2, \ldots, y_k, t) \end{aligned}$$

The quantity y_k represents the output of the kth integrator, and t represents time. In the following formulas, n refers to the nth integration step; DT is the step size.

Euler

$$y_{n+1} = y_n + DT \cdot y_n'$$

Trapezoidal

$$y_{n+1} = y_n + \frac{DT}{2}(y_n' + y_{n+1}')$$

Runge-Kutta second order

$$k_1 = DT \cdot y'_n(t_n, y_n)$$

$$k_2 = DT \cdot y'_{n+½}\left(t_n + \frac{DT}{2}, y_n + \frac{k_1}{2}\right)$$

$$y_{n+1} = y_n + \frac{(k_1 + k_2)}{2}$$

Runge-Kutta third order (variable)

$$k_1 = DT \cdot y'_n(t_n, y_n)$$

$$k_2 = DT \cdot y'_{n+½}\left(t_n + \frac{DT}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = DT \cdot y'_{n+1}(t_n + DT, y_n + 2k_2 - k_1)$$

$$y_{n+1} = y_n + \frac{(k_1 + 4k_2 + k_3)}{6}$$

Runge-Kutta fourth order

$$k_1 = DT \cdot y'_n(t_n, y_n)$$

$$k_2 = DT \cdot y'_{n+½}\left(t_n + \frac{DT}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = DT \cdot y'_{n+½}\left(t_n + \frac{DT}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = DT \cdot y_{n+1}(t_n + DT, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{(k_1 + 2k_2 + 2k_3 + k_4)}{6}$$

Simpson's Rule

$$y_{n+1} = y_n + \frac{(y'_n + 4y'_{n+½} + y'_{n+1})}{6}$$

The variable step error criterion is calculated as follows. ERR_n is the maximum of the relative errors of all the integrators which occur during the integration step n. In other words,

$$ERR_n = \left| \frac{y_{n+1}^{(RK3)} - y_{n+1}^{(RK2)}}{y_{n+1}^{(RK3)}} \right|$$

where $y_{n+1}^{(RK3)}$ is the output of the kth integrator at step n+1 as computed by the RK3 routine, $y_{n+1}^{(RK2)}$ is the output of the kth integrator at step n+1 computed by the RK2 routine, and so on.

The following rules are used to calculate the step size in the variable step size routine.

If $ERR_n \geqslant RELERR, \quad DT = \frac{DT}{2}$

If $\frac{RELERR}{8} \leqslant ERR_n < RELERR, \quad DT = DT$ (unchanged)

If $ERR_n < \frac{RELERR}{8}, \quad DT = 2 \cdot DT$

The user must supply an initial value for DT, the integration step size. It is better to choose a small DT than one too large because the program will have to perform unnecessary computations in order to reduce the step size so that computation of the solution may begin. The accuracy of the solution is governed by RELERR; the maximum integration interval is set by the output interval BY.[1]

1 - The output interval BY is discussed on page 42.

SECTION 5
PROBLEM ENTRY

CSMP is called by typing CSMP and a Carriage Return in the EXECUTIVE. The program responds with a colon, indicating its readiness to accept commands.

Any CSMP command may be given when a colon appears and must be terminated by a Carriage Return. Each command may be abbreviated to the least number of characters needed to uniquely identify it. Appendix A contains a table of minimum abbreviations.

This section explains the various methods available for describing the block configuration, initial conditions, and parameters to CSMP. In addition, the commands used to specify timing information are discussed. Commands for output, execution, and other CSMP features are discussed in succeeding sections.

ENTERING THE CSMP DESCRIPTION

There are two ways to enter specifications into Tymshare CSMP: from the terminal or from a file. Data files used in CSMP are usually created in a previous CSMP session, as with the DUMP command, or in Tymshare's EDITOR.[1] The data in each file must be written in the same form as it would be entered at the terminal. This form is discussed for each individual command. Files created in CSMP are automatically written in this form. The commands used to create and load files are discussed in Section 8.

If the user makes an error while entering data, he may use a Control A, Control Q, or Control W to correct it. These control characters may be used in both CSMP and EDITOR.

Control A deletes the preceding character. On some terminals, Control A is acknowledged by a back arrow (←). For example,

$\underline{5\qquad 2.51A^{C}}\leftarrow\underline{7}$↵

The user meant to type 2.57 as the value of the first parameter. Instead, he typed 2.51. He then typed Control A to erase the 1. The Control A was acknowledged with a back arrow. The user then typed the digit 7 and a Carriage Return. Several Control A's may be typed consecutively to delete successive preceding characters.

Control Q erases all characters, including data values, from the statement in which it appears. The entire line of data must be reentered. On some terminals, Control Q prints an up arrow (↑) to indicate its use. For example,

INT (0) = $\underline{2.19E-2}$↵
INT (1) = $\underline{2.19E-2Q^{C}}$ ↑
$\underline{3.24E-1}$↵

The user retyped the value for INT (0) instead of the desired value for INT (1). Since he had not typed a Carriage Return, he typed Control Q. When Control Q acknowledged with the up arrow and returned the carriage, the user typed the desired value for INT (1).

1 - See the *Tymshare EDITOR Reference Manual* for further details.

Control W deletes the preceding word in the line being typed. The preceding word is defined as including the immediately preceding blanks, if any, plus the immediately preceding non-blank characters, up to but not including the first blank preceding them. On some terminals, a back slash (\) is printed when Control W is used. For example,

<u>19 W 400W</u>^{C}\\<u>300 11</u>↩

The user meant to type 300 as the first input value. He used a Control W to delete the 400 and correctly completed the line.

NOTE: Control characters are valid methods to edit data only if the Carriage Return has not yet been typed. If the data has already been entered in CSMP and the Carriage Return typed, the only way to correct the data value is to reexecute the command or line in which the data was entered.

Numbers in CSMP may be expressed in integer form, decimal notation, or E format. Note that the E format cannot stand alone; thus, 1000 may be written as 1E3, 1.E3, or 10.0E2, but not as E3.

For a particular numerical entry, the user may type as many digits as he wishes. However, any number containing more than 11 significant digits will be rounded to 11 digits. CSMP carries all calculations to 11-digit accuracy, but the final results are printed in E format to four-digit accuracy.

NOTE: All block specifications, initial conditions, and parameters are set to 0 if not specified otherwise.

Comments may be included by preceding them with a semicolon. For example,

```
19 W 52 11 ; WEIGHTED SUMMER
11 J                 ; RANDOM NUMBER GENERATOR
```

LANGUAGE STATEMENT COMMANDS

The three language statement commands, BLOCKS, INITIAL, and FUNCTION, specify the block configuration and all initial conditions.

The BLOCKS Command

The BLOCKS command is used to enter, alter, or add block configuration specifications from the terminal. A block statement consists of the block number, block type, and block inputs (E1, E2, and E3). All data for a single block must be entered on one line and terminated by a Carriage Return. The data items may be separated by spaces or commas. Line Feeds are ignored.

The form of the command is:

:<u>BLOCKS</u>↩

The command then prints the heading

BLOCK TYPE E1 E2 E3

after which the user enters his block statements. The command is terminated by an additional Carriage Return or the word END followed by a Carriage Return. For example,

```
:BLOCKS ↵

BLOCK TYPE E1  E2  E3

99 K ↵
1 I 2 ↵
5 W 2 1 ↵
6 / 5 99 ↵
2 I 6 ↵
↵                An extra Carriage Return indicates the completion of block specifications.
BEGIN SORT

:
```

NOTE: The block numbers specified for the inputs may range from 1 through 301. If the user specifies a 0 for an input, the value 0 is used.

The INITIAL Command

The INITIAL command is used to alter or enter from the terminal the initial conditions and/or parameters associated with the blocks specified with the BLOCKS command. The INITIAL command cannot be given until the block configuration has been specified.

The INITIAL statement consists of the block number, initial condition P1, and associated parameters P1 and P2. The data items may be separated by spaces or commas.

The form of the command is:

:**INITIAL** ↵

The command then prints the heading

BLOCK P1 P2 P3

after which the user enters his statements. The command is terminated with an additional Carriage Return or the word END followed by a Carriage Return. For example,

```
:INITIAL ↵

BLOCK P1 P2 P3

99 -1.0 ↵
1 -1.0 ↵
5 2.0 1.0 ↵
↵                The Carriage Return ends the INITIAL command.

:
```

The FUNCTION Command

The FUNCTION command allows the user to enter function generator specifications from the terminal or to modify a previously entered function generator. The block number is requested, and then the 11 function values are requested individually. Finally, the user must specify the maximum and minimum input values for the function generator block input, unless he has specified them with the INITIAL command.

TIMING COMMANDS

The following seven timing commands may be used to assign or alter timing specifications.

Command	Description
DT	Specifies the integration interval or step size.
FROM	Specifies output start time.
BY	Specifies the output interval.
TO	Specifies the end time.
TIMING	Prompts the user for DT, FROM, BY, and TO.
DTMIN	Specifies the minimum integration interval used by the VARIABLE integration routine. If the user does not specify this value, DTMIN is assigned the value DT · 10^{-6}.
RELERR	Specifies the relative error bound on the solution when the VARIABLE integration routine is used.[1] If RELERR is not specified, .001 is used.

The timing specifications may be entered on one line in any order, separated by spaces or commas. For example,

```
:DT .1  FROM 10  TO 100,BY .1↲
```

If any one of the variables DT, FROM, BY, or TO is specified, CSMP automatically assigns values to the remaining variables according to the following conventions:

1. DT is specified:

If BY is not specified,	BY = 10 · DT
If TO is not specified,	TO = 200 · DT

2. BY is specified:

If DT is not specified,	DT = BY/10
If TO is not specified,	TO = 20 · BY

3. TO is specified:

If DT is not specified,	DT = TO/200
If BY is not specified,	BY = TO/20

1 - The VARIABLE routine is described on page 34.

For all cases, if FROM is not specified, it is assigned the value 0, or if the SAVE command[1] has been given, FROM is assigned the value T_{ZERO}.[2]

Values for RELERR and DTMIN affect the solution only when the variable step size integration routine is used. CSMP sets these values automatically; the user may reset them as illustrated below. There are two forms for each command.

```
:RELERR ↵      The user specifies the relative error in two ways.

RELATIVE ERROR = .02 ↵

:RELERR .02 ↵

:DTMIN ↵ The variable DTMIN is set to .0000008.

MINIMUM INTEGRATION INTERVAL = .0000008 ↵

:DTMIN .0000008 ↵

:
```

The TIMING command prompts the user for the following timing specifications: DT, the integration interval or step size; FROM, the output start time; BY, the output interval; and TO, the end time. For example,

```
:TIMING ↵

DT: .1 ↵

OUTPUT
FROM: 0 ↵
BY: .5 ↵
TO: 6 ↵

:
```

1 - The SAVE command is described on page 61.

2 - T_{ZERO} is defined on page 55.

For computational purposes, the program requires that the output interval BY be an even multiple of the integration interval DT. However, it is not necessary for the user to choose BY to be an even multiple of the integration interval DT. DT is automatically adjusted by the program so that BY/DT is an integer. For example, in

```
:TIMING ↵

DT: .3 ↵

OUTPUT
FROM: 0 ↵
BY: 1 ↵
TO: 10 ↵

:DISPLAY DT ↵
3.333E-01

:
```

the ratio BY/DT is 1/.3, which is not an integer. Thus, the program adjusts DT to .3333. *BY is never adjusted.* This fact should be remembered when modeling with unit delay blocks.[1]

1 - Unit delay blocks are discussed on page 21.

SECTION 6
OUTPUT COMMANDS

The results of a CSMP computation may be displayed in a numerical table, plot, or both. Numerical output may be printed in either four- or eight-digit accuracy. The time span over which the data is displayed can be varied, and headings may optionally be suppressed. Both the tables and plots may be printed on the terminal or written on a file for future reference. This section discusses these output commands and also explains how interrupts are handled.

THE PRINT COMMAND

The PRINT command is used to print the outputs of as many as ten blocks at a time. There are two forms of the command:

:PRINT block numbers↵

and

:PRINT↵
OUTPUT BLOCKS: block numbers↵

The block numbers may be separated by spaces or commas.

NOTE: The PRINT command is not actually executed until a GO or CONTINUE command is given.[1]

THE PLOT COMMAND

The PLOT command is used to plot the outputs for one to ten blocks on a single set of axes. These plots may either be scaled by the user or automatically scaled by the program. Automatic scaling requires more time because all the points to be plotted must be computed to determine the largest and smallest values. If the automatic scaling option is not chosen, the user must supply the maximum and minimum y-axis values. The form of the command is:

:PLOT block number↵

The program responds with:

AUTOMATIC SCALING?

If the user answers YES, the PLOT command requests the next block number. If the user answers NO, the PLOT command requests the values for YMAX and YMIN, the maximum and minimum values to be plotted. The command is completed by typing a Carriage Return after PLOT BLOCK. For example,

```
:PLOT 4↵
AUTOMATIC SCALING? YES↵
PLOT BLOCK: 9↵
AUTOMATIC SCALING? NO↵
YMAX = 100↵
YMIN = 58↵
PLOT BLOCK ↵
```
The Carriage Return terminates the command.

1 - The GO and CONTINUE commands are described in Section 7.

plots values for the blocks 4 and 9. Block 4 is automatically scaled; block 9 is plotted between the values 58 and 100.

The symbols used for the plots, beginning with the first plot, are:

+ * . , 0 @ # : – ↑

If two or more points are to be plotted at the same location, the symbol associated with the block requested first is used.

The user may specify a series of block numbers after the PLOT command. In this case, CSMP automatically scales all the plots. For example,

```
:PLOT 1,3,5 7 17↵
```

automatically scales each of the plots.

NOTE: The PLOT command is not actually executed until a GO or CONTINUE command is given.[1]

THE IPLOT COMMAND

The IPLOT command is similar to the PLOT command. IPLOT is used for multiple plotting when the user wishes to specify the minimum and maximum y-axis values for each block to be plotted. As many as ten blocks may be plotted on a single set of axes. For example,

```
:IPLOT↵

PLOT BLOCK: 1↵
YMAX = 5.0↵
YMIN = -1.5↵
PLOT BLOCK↵    The Carriage Return terminates the command.

:
```

plots the values for block 1.

NOTE: The IPLOT command is not actually executed until a GO or CONTINUE command is given.[1]

THE DOUBLE AND SINGLE COMMANDS

CSMP performs all arithmetic operations in 11-digit accuracy at all times. Normally, results are printed in four-digit accuracy. The DOUBLE command specifies that all numerical output be printed in eight-digit accuracy. DOUBLE does not affect timing or solution accuracy; it is merely a format command.

The SINGLE command returns to the standard method of displaying output in four-digit accuracy.

1 - The GO and CONTINUE commands are described in Section 7.

THE NO HEADING COMMAND

The NO HEADING command suppresses the headings normally printed with the PRINT, PLOT, and IPLOT commands. This command is given with the execution commands. For example,

```
:GO NO HEADING↩
```

THE OUTPUT COMMAND

The OUTPUT command writes on a file the solution just as it would appear on the terminal. It is useful when the user wishes to list the solution at a later time. The command forms are

```
:OUTPUT file name↩
```

and

```
:OUTPUT↩
TO: file name↩
```

where file name is the output file. The OLD FILE/NEW FILE message is printed and requires a Carriage Return to confirm or an Alt Mode/Escape to abort the command.

The OUTPUT command can be used when the user wishes to store the solution data on a file to be examined at a future time.

This command opens the file for symbolic output; it remains open until the OUTPUT command is given with TERMINAL (or T) specified as the output file. If NOTHING is specified as the file name, the output is suppressed.

INTERRUPTS

If the user types one Alt Mode/Escape, he is returned to CSMP command level after the output buffer has been emptied. For example,

```
:GO↩

  TIME            1

  .000E 00     4.000E 01
 1.000E 00     4.441E 01⊕
 2.000E 00     4.477E 01
 3.000E 00     4.501E 01
 4.000E 00     4.527E 01
```

The user typed an Alt Mode/Escape here. The next three lines were in the output buffer. After they were printed, the user was returned to CSMP command level.

```
:CONTINUE ↩

   TIME           1

 5.000E 00    4.535E 01
 6.000E 00    4.546E 01
 7.000E 00    4.548E 01
 8.000E 00    4.554E 01
 9.000E 00    4.561E 01
 1.000E 01    4.562E 01

:
```

Typing an Alt Mode/Escape two or more times returns the user to CSMP command level immediately. Any data in the output buffer is not displayed on the terminal. For example,

```
:GO ↩

   TIME           1

  .000E 00    4.000E 01
 1.000E 00    4.441E ⊕⊕
```

The user typed two Alt Mode/Escapes here. He was returned immediately to CSMP command level.

```
:CONTINUE ↩

   TIME           1

 7.000E 00    4.548E 01
 8.000E 00    4.554E 01
 9.000E 00    4.561E 01
 1.000E 01    4.562E 01
```

The data for block 1 from time 2 through time 6 was in the output buffer and was not displayed on the terminal. This data was written on the scratch file.[1]

If the user types the Alt Mode/Escape two or more times and the message

ESCAPE HIT WHILE IN BLOCK: block number

is printed, continuation is not possible. This message indicates that the program is locked in a block; for example, an infinite loop in a user-defined subroutine would cause this condition.

Any time the user types the Emergency Exit Key, the system returns immediately to the EXECUTIVE.[2] The user should be aware that some scratch files used by CSMP will remain in his directory in this case.

1 - The scratch file is discussed in Section 9.

2 - The Emergency Exit Key is described in the *Tymshare EXECUTIVE Reference Manual.*

SECTION 7
EXECUTION COMMANDS

THE GO COMMAND

The CSMP user may give a GO command to begin program execution or to read data from the scratch file.[1] An alternate form of the GO command is EXECUTE. The two forms are equivalent.

THE CONTINUE COMMAND

The CONTINUE command is used to continue solution computation. Computation may not be continued if the block configuration has been changed or if the output interval has been changed.

THE RUN COMMAND

The RUN command requests block configurations, initial conditions and parameters, function generator specifications (if a function generator block has been named in the configurations), timing data, and output information. RUN then begins computation. If a numerical table has been specified, it is printed. CSMP then asks the user if he wishes the output of any blocks plotted.

The responses requested by the RUN command may be entered from the terminal or from a file. If the user types an Alt Mode/Escape during the RUN command, specifications entered before the interruption are retained.

1 - The scratch file is described in Section 9.

SECTION 8
FILE COMMANDS

CSMP contains several commands which allow the user to save on a file program information for use at another time. Additional commands load these and other previously created files into CSMP and allow the user to copy files. This section documents these commands; several of them are demonstrated in the examples in Section 11.

THE READ COMMAND

The READ command requests the name of a previously created symbolic file and reads from that file the block configuration specifications, the initial conditions and parameters, and the function generator specifications, if any. This data must be in the same format used to enter data from the terminal.

THE WRITE COMMAND

The WRITE command saves the current block configuration specifications, initial conditions and parameters, and function generator specifications on the symbolic file named by the user. The forms of the command are:

```
:WRITE file name↵
```

and

```
:WRITE↵
ON FILE: file name↵
```

The system prints the OLD FILE/NEW FILE message which requires a Carriage Return to confirm or an Alt Mode/Escape to abort the command.

THE DUMP AND RECOVER COMMANDS

The DUMP command creates a small binary file in the user's directory. This file contains all of the block specifications, initial conditions and parameters, function generator specifications, timing specifications, integration algorithm, etc. The DUMP command may be given at any time. Its form is

```
:DUMP file name↵
```

where file name is the name of the binary file to be created.

The RECOVER command is the complement of the DUMP command; it restores the conditions that existed at the time the DUMP command was given. Its form is:

```
:RECOVER file name↵
```

With these two commands, the user may save all of his program specifications on a file to be loaded into CSMP at another time. After the file is loaded, the user may begin computation with the GO command, or he may first alter various specifications and then begin execution with GO.

The DUMP and RECOVER commands are similar to the WRITE and READ commands. However, even though they save more information than the WRITE and READ commands, the DUMP and RECOVER commands are much faster to use.

DUMP SUBROUTINES and RECOVER SUBROUTINES

The DUMP SUBROUTINES command allows the user to create a binary file which contains his subroutines and which is compatible with CSMP. After the user gives the SUBROUTINES command, he will usually want to save the special file which links the subroutines to CSMP by using the DUMP SUBROUTINES command.[1]

The RECOVER SUBROUTINES command loads the subroutine files previously created with DUMP SUBROUTINES. Files created with the DUMP SUBROUTINES command may be recovered much faster with the RECOVER SUBROUTINES command than with the reuse of the SUBROUTINES command.

The forms of these commands are

```
:DUMP SUBROUTINES file name↵
```

and

```
:RECOVER SUBROUTINES file name↵
```

where the subroutines are written on or recovered from the specified file.

The DUMP SUBROUTINES and RECOVER SUBROUTINES commands are illustrated in Example 4, page 83.

DUMP ALL and RECOVER ALL

The DUMP ALL command saves on a file all the current conditions and specifications existing in a CSMP program and all the data recorded on the scratch file.[2]

This command creates a binary file in the same manner as the DUMP command described above, but the DUMP ALL command appends to the file the data recorded on the scratch file. Thus, the size of the file depends on the amount of data that has been written on the scratch file at the time the DUMP ALL command was given.

The DUMP ALL command may be given at any time. Its form is:

```
:DUMP ALL file name↵
```

The form of the RECOVER ALL command is:

```
:RECOVER ALL file name↵
```

The RECOVER ALL command restores all the initial conditions, specifications, and data which existed at the time the DUMP ALL command was given.

For example, the user may interrupt the program during execution with an Alt Mode/Escape, give the DUMP ALL command, give the QUIT command, and log out. If the program contains user-defined blocks, he must also give the DUMP SUBROUTINES command. At any later time, he may return to CSMP, give the RECOVER ALL command for that file, and type CONTINUE. Execution proceeds as if no interruption has occurred.

Instead of continuing execution from the point of interruption, the user may give the GO command in conjunction with a command such as PLOT to examine the data already created.

1 - The SUBROUTINES command is described on page 22.

2 - The scratch file is discussed in Section 9.

THE COPY COMMAND

The COPY command is used to create files, to list the contents of files, and to copy the contents of one file to another. The general form of the command is

```
:COPY source TO destination↵
```

where the source and destination may be file names or the terminal. A space or comma may be used instead of TO. The COPY command causes the contents of the source to be duplicated in the destination but does not affect the contents of the source.

If the destination is a file, a check is made to see if a file by that name already exists. The system types either

OLD FILE

if the file name already exists, or

NEW FILE

if a new file name is being created, and waits for the user either to confirm or to abort the command. The command is confirmed simply by typing a Carriage Return. If the command is confirmed, the contents of an old file are completely erased and replaced by the contents of the source. To save the contents of the old file, the command can be aborted by typing the Alt Mode/Escape.

If the source is a file, the file must be a symbolic or binary file in the user's directory. For example,

```
:COPY DATA TO DATA1↵
```

copies the contents of the file DATA to the file DATA1.

The command

```
:COPY T TO CONTROL↵
```

allows the user to enter the contents of the file CONTROL from the terminal.

When the source is the terminal, the user may edit the line he is typing with Control A, Control Q, and Control W.[1] Terminal input is terminated with a Control D.

Note that the user cannot copy files to or from a directory other than his own.

The COPY command is especially useful for creating a command file in CSMP.[2] For example, the following is a command file created by the COPY command.

```
:COPY T TO A ↵
 NEW FILE ↵

INITIAL↵
44 3.E4 -6E3↵
27 1 0↵
↵              The single Carriage Return terminates the INITIAL command.
DT .05 ↵
GO ↵
Dᶜ  The COPY command is terminated with Control D.
:
```

1 - These control characters are described on page 39.

2 - Command files are executed with the DO command, discussed in Section 10. The COPY and DO commands are illustrated in Examples 2 and 4, pages 69 and 83.

If a file has been opened with the OUTPUT command, it may be copied to the terminal; its contents are not affected. A file opened with the OUTPUT command remains open for solution output until the user gives the command:

```
:OUTPUT TERMINAL↵
```

If the user copies a file opened with the OUTPUT command while it is still open, the message

```
FILE BUSY
```

is printed. However, the copy operation will be completed correctly, and the user should ignore this message. It is merely a reminder that the file is open.

THE CLOSE COMMAND

The CLOSE command enables the CSMP user to close specified files. If a file number is included after the command, CSMP closes the file with that unit number. For example,

```
:CLOSE 6↵
```

closes file number 6, previously opened by the user. If the file is not open when the CLOSE command is given, CSMP takes no action.

If the CSMP user specifies no file number after the CLOSE command, CSMP closes the units 2, 4, 5, 6, 7, 8, 9, 13, and 15. Note that these are the only file numbers the user may assign within a user-defined subroutine. Thus,

```
:CLOSE↵
```

closes all files which the user has opened.

SECTION 9
THE SCRATCH FILE

At each output interval, specified with the BY command, CSMP writes the output of each of the blocks in the configuration on a binary scratch file, CSMP'SCR1', in the user's directory. CSMP automatically deletes this file when the user gives the QUIT command. If the user leaves in a manner other than by typing QUIT, he should delete the scratch file. The easiest way is to call CSMP and give the QUIT command.

The advantage of the scratch file is that *without recomputing* the solution, the outputs of any block in the configuration may be printed or plotted for any time interval contained within the scratch file. An understanding of the scratch file allows the user to make the most efficient use of the GO, CONTINUE, and SAVE commands.[1]

Within the scratch file, time is referenced to the variable T_{ZERO}. Initially, $T_{ZERO} = 0$. If the SAVE command is given, $T_{ZERO} = T_{SAVE}$, where T_{SAVE} is the time at which the SAVE command is executed. The time at the end of a run is T_{END}.

A run may be terminated by: (1) TO, which is specified by the user, (2) the user's typing an Alt Mode/Escape, (3) a quit block in the configuration, and (4) an error in the user's program (overflow, division by 0, etc.). Thus, the scratch file contains the outputs of all of the blocks in the program from T_{ZERO} to T_{END} in intervals specified by BY.

The GO command is used to read data from the scratch file for any time interval contained in the file. For instance, if the GO command has been given to execute the problem and computation is complete, any of the following commands may be given, followed by another GO, without destroying the data contained in the scratch file.

PROBE	OUTPUT
DUMP	PLOT
DUMP ALL	IPLOT
WRITE	DISPLAY
SINGLE	GO, EXECUTE
DOUBLE	EXPERT
FROM	NONEXPERT
TO	NO HEADINGS

In addition to the above commands, the user may change

TO	RELERR
DT	DTMIN

and use the CONTINUE command without erasing data from the scratch file.

Notice, however, that changing the output interval BY always erases the scratch file data.

1 - The GO and CONTINUE commands are described in Section 7. The SAVE command is discussed in Section 10.

The SCRATCH command prints the status of the scratch file, indicating the period of data contained on the file which can be listed on the terminal or written on a file without recalculation. If the scratch file is empty, CSMP prints the message

DATA FILE EMPTY

after the command.

The ERASE command erases all the data in the scratch file.

Example

```
:SCRATCH
DATA FILE
FROM:    .000E 00
BY:  2.500E-01
TO:  1.000E 01

:ERASE

:SCRATCH
DATA FILE EMPTY
```
The scratch file is empty after the ERASE command.

```
:
```

Assume the user has already entered a problem, giving the block configuration and initial conditions. He then specifies the timing parameters and executes the program with the following sequence of commands:

```
:TIMING

DT: .01

OUTPUT
FROM: 0
BY: .1
TO: 10.0

:PRINT
PRINT BLOCKS: 1 2 3 6 7 8

:GO
```

At this point, CSMP has calculated the output values for all blocks from time 0 to 10 in intervals of .1 and has printed these values for blocks 1, 2, 3, 6, 7, and 8. $T_{ZERO} = 0$, $T_{END} = 10.0$, and any of the commands listed under the GO command may now be given without destroying the data on the scratch file.

For instance, the user continues the above program by plotting blocks 4 and 9 from 2 to 8 (the interval .1 must remain the same). To do this, he gives the commands:

```
:FROM 2 ↵

:TO 8 ↵

:PLOT 4 ↵
AUTOMATIC SCALING? YES ↵
PLOT BLOCK: 9 ↵
AUTOMATIC SCALING? NO ↵
YMAX = 100 ↵
YMIN = 58 ↵
PLOT BLOCK: ↵

:GO ↵
```

CSMP plots, in intervals of .1, the values for blocks 4 and 9 on the same axes for the time period 2 through 8. The values are read from the scratch file and are *not* recomputed.

The contents of the scratch file are erased completely if the output interval BY is changed or if a FROM value is specified which is less than T_{ZERO}. The SAVE command erases the contents of the scratch file up to but not including the values at time T_{SAVE}.

As another example, assume that all program specifications except the timing specifications have been entered. The commands

```
:TO .01 FROM .005 BY .0004 ↵

:PLOT  7 ↵
AUTOMATIC SCALING? YES ↵
PLOT BLOCK: ↵

:GO ↵
```

plot block 7 from time .005 through .01 in intervals of .004. At this point, $T_{ZERO} = 0$, $T_{END} = .01$, and the outputs of all blocks in the program, including block 7, are recorded on the scratch file from T_{ZERO} through T_{END}. Now the user wishes to continue computation and extend the time interval to .015.

```
:TO .015 ↵

:CONTINUE ↵
```

These commands instruct CSMP to scale and plot block 7 from time .005 to .015. T_{ZERO} = 0.0, and T_{END} = .015. The scratch file contains values for all blocks from time 0 to .015.

```
:SAVE ↵        This command sets TZERO = TSAVE = .015.

:RK4 ↵         The user now changes the integration algorithm.

:TO .02 ↵      This command changes the output end value TO to .02.

:GO ↵
```

CSMP scales and plots block 7 from time .015 to .02 in intervals of .0004. The values were calculated using the RK4 algorithm. T_{ZERO} was equal to .015 since the SAVE command was given. The scratch file contains output for all the blocks from T_{ZERO}=.015 to T_{END}=.02.

The user now wishes to plot block 8 from .0001 to .02:

```
:TO .02 FROM .0001 ↵

:PLOT 8 ↵
AUTOMATIC SCALING? YES ↵
PLOT BLOCK ↵

:GO ↵
```

Because the user has requested an output start time (.0001) which is not in the interval T_{ZERO} to T_{END} (.015 to .02), a new solution must be computed from T_{ZERO} = 0. Even though the scratch file contains useful data from .015 to .02, a new solution is calculated from time 0 to .02.

SECTION 10
DEBUGGING AND UTILITY COMMANDS

DEBUGGING COMMANDS

CSMP contains six debugging commands to give the user additional information about the problem configuration. These commands are FIND, DISPLAY, DELETE, PROBE, SAVE, and CPU.

The FIND Command

The FIND command lists all references to either a block type or a block number or both. For example,

```
:FIND I,F 3

  1     I     0     31     0
  2     I     0     32     0
  3     I     0     33     0

 13     F     3      0     0

  3     I     0     33     0
 13     F     3      0     0
 41     G     3      0     0
 43     G     3      0     0
 53     +    -1      3     0
```

The user may separate the quantities after the FIND command with commas or spaces.

The DISPLAY Command

The DISPLAY command prints specified quantities on the terminal. The quantities which may be specified are:

Quantity	Description
BLOCKS	Lists block configuration specifications, initial conditions, and parameters.
INITIAL	Lists initial conditions and parameters.
FUNCTION	Lists function generator specifications.
TIMING	Lists timing specifications.
RELERR	Prints the current RELERR value.

Quantity	Description
DT	Prints the current DT value.
DTMIN	Prints the current DTMIN value.
FROM	Prints the current FROM value.
BY	Prints the current BY value.
TO	Prints the current TO value.
ALGORITHM	Prints the name of the current integration algorithm.
any valid block number	Displays the specified block.
ALL	Displays all of the above.

The quantities to be displayed may be entered on one line separated by commas or spaces. For example,

```
:DISPLAY DT ALGORITHM RELERR 14 20↵
```

The command prompts the user for the quantities if he types a Carriage Return after DISPLAY. The program responds with DISPLAY:, after which the user enters the quantities on one line as described above. For example,

```
:DISPLAY↵
DISPLAY: DT ALGORITHM RELERR 14 20↵
```

The quantities to be displayed may be entered on more than one line if all lines except the last are terminated with a Line Feed. The command including quantities may contain as many as 256 characters excluding Line Feeds.

NOTE: The command

DISPLAY FUNCTION

displays all function generators. To display only one function generator, the user types

DISPLAY block number

where block number is the number of the function generator.

The LIST command is identical to the DISPLAY command.

The DELETE Command

The DELETE command is used to delete specifications from the CSMP program. The following quantities may be deleted:

ALL	Completely clears CSMP so that another problem may be entered.
BLOCKS	Same as ALL.
FUNCTION	Erases the function generator specifications.
INITIAL	Erases the initial conditions and parameters.
TIMING	Erases previous timing specifications.

There are two forms of the command:

```
:DELETE quantity↵
```

and

```
:DELETE↵
DELETE: quantity↵
```

The PROBE Command

The PROBE command prints the last value computed for specified blocks in the configuration. Time, block 301, may be specified. The blocks to be probed are entered on one line separated by spaces or commas. They may follow the command or the user may be prompted for the block numbers. For example,

```
:PROBE 1 4 10 13 200 301↵
```

and

```
:PROBE↵
BLOCK: 1 4 10 13 200 301↵
```

both print the last computed values for blocks 1, 4, 10, 13, 200, and 301.

Example

```
:PROBE 82 5 84 86 78↵    The user requests the current values of blocks 82, 5, 84, 86, and 78.
 82:   5.896E 01
  5:    .000E 00
 84:   5.896E 01
 86:   5.896E 01
 78:   3.474E-03

:
```

The SAVE Command

The SAVE command is useful for studying a system's response after a given time T. A system may be quite stable until a certain time. At this time, the SAVE command could be given and the system analyzed using various integration intervals, integration algorithms, output intervals, etc. Execution time would be reduced since all computations begin at T_{SAVE} instead of 0.

The SAVE command performs the following functions:

1. The time at which the command is given, T_{SAVE}, becomes the reference time. In other words, $T_{ZERO} = T_{SAVE}$.
2. It saves the output values and parameters of all blocks in the configuration at time T_{SAVE}.
3. The output start time is set equal to T_{SAVE}; in other words, FROM = T_{SAVE}.
4. It erases all of the data on the scratch file[1] up to but not including the data computed at T_{SAVE}.

1 - The scratch file is discussed in Section 9.

After the SAVE command has been given, the GO command begins computations at the time and conditions at T_{SAVE} instead of 0.

The form of the command is simply:

```
:SAVE
```

In previous versions of Tymshare CSMP it was necessary to use a RESTORE command in conjunction with the SAVE command. There is no longer a RESTORE command, since GO performs this function automatically.

The SAVE command is illustrated in Example 4 on page 83.

The CPU Command

The CPU command prints the total CPU time used since log in and DELTA CPU. DELTA CPU is initialized to 0 upon entering CSMP and each time the command is given. For example,

```
:CPU
TOTAL CPU:    304 SEC
DELTA CPU:     24 SEC
```

This command is helpful for choosing the most economical integration algorithm and integration interval for a particular problem.

UTILITY COMMANDS

CSMP contains several utility commands to facilitate use. The most useful of these commands is the DO command which executes a command file in CSMP. It has the form

```
:DO file name
```

where the file name is the name of the command file. The DO command is illustrated in Examples 2 and 4, pages 69 and 83.

The table on the following page lists the other CSMP utility commands.

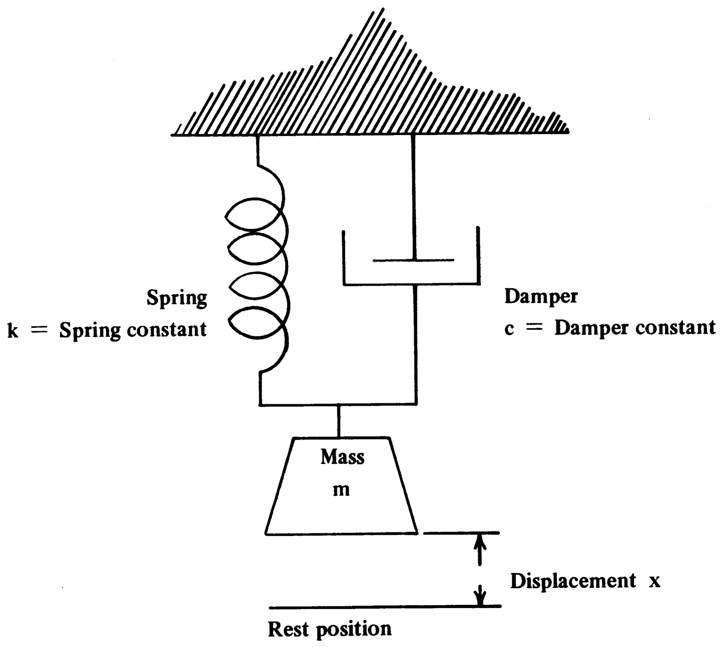
Command	Description
CAPABILITIES	Describes program capabilities.
CHARGES	Lists additional cost, if any.
CREDITS	Prints credit for implementation of CSMP.
EXPERT	Used by those familiar with CSMP. Prints minimum information when requesting data.
HELP *or* ?	Prints a list of CSMP commands with descriptions.
INSTRUCTIONS	Prints a list of instructions used to execute the program.
NONEXPERT	Returns the user from the EXPERT mode to the standard NON-EXPERT mode.
QUIT	Returns the user to the EXECUTIVE and deletes all scratch files.
VERSION	Lists the number and date of the last update of CSMP.

SECTION 11
SAMPLE PROBLEMS

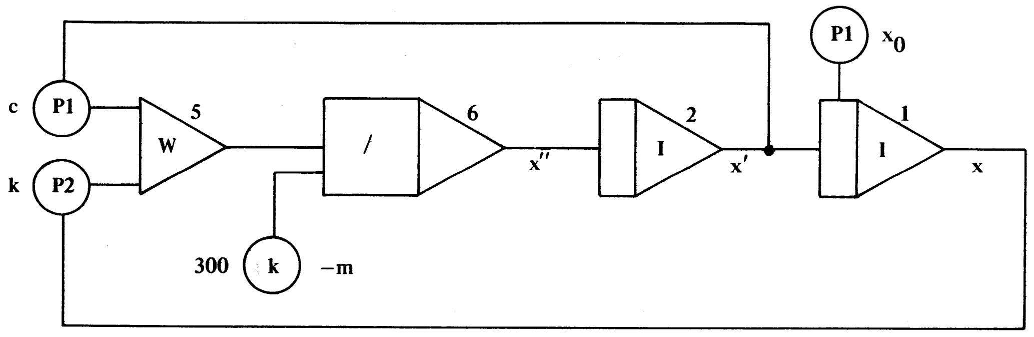
This section contains four sample problems illustrating the major CSMP commands. Example 1 illustrates the basic CSMP problem entry and execution. Example 2 illustrates the recovery and modification of a previous problem; Example 2 also utilizes a command file to produce four successive plots with a minor modification in each plot. Example 3 contains a function generator block. Example 4 illustrates the use of a user-defined block and a command file.

Example 1 – SPRING, MASS, DAMPER SYSTEM

The diagram below shows a simple mechanical system consisting of a spring, a mass, and a damper suspended from a fixed reference position. If the mass is displaced from its rest position and then released, it oscillates until the energy is dissipated by the damper. The purpose of a simulation might be to analyze the effect of different spring constants on the motion of the mass. One possible simulation diagram for representing the system is shown below the system illustration.



Model Equation: $mx'' + cx' + kx = 0$



```
-CSMP ↩

:BLOCKS ↩

BLOCK TYPE E1  E2  E3

300  K ↩
1  I  2 ↩
5  W  2  1 ↩
6  /  5  300 ↩
2  I  6 ↩
↩
BEGIN  SORT

:INITIAL ↩

BLOCK  P1  P2  P3

300  -1 ↩
1  -1 ↩
5  .5  1 ↩
↩
```

```
:TIMING↵

DT:  .025↵

OUTPUT
FROM:  10↵  The output specifications suppress terminal output until time=10.  Data
BY:  .25↵   is written on the scratch file at intervals of .25 from time=0 to time=10.
TO:  10↵

:PRINT  1  2  6↵

:DUMP  SPRING↵       The user creates a binary file containing all program parameters.
 OLD  FILE↵


:GO↵

   TIME              1               2              6

 1.000E 01     8.472E-02     -2.165E-02     -7.389E-02

:VARIABLE↵  The user checks the accuracy of the solution with a second integration algorithm.

:GO  NO  HEADING↵

 1.000E 01     8.439E-02     -2.182E-02     -7.348E-02
                    The accuracy is sufficient for plotting.

:FROM  0↵  The user changes the start time to 0.

:PLOT  1↵
AUTOMATIC  SCALING?  NO↵
YMAX  =  1↵
YMIN  =  -1↵
PLOT  BLOCK:  2↵
AUTOMATIC  SCALING?  YES↵
PLOT  BLOCK:↵
```

:GO ↵ *The program does not recalculate solutions; it reads the scratch file data to produce the plot.*

```
BLOCK SYMBOL    MIN          MAX        INCREMENT
    1    +   -1.000E 00    1.000E 00    4.000E-02
    2    *   -3.139E-01    7.067E-01    2.041E-02

    TIME      +....+....+....+....+....+....+....+....+....+....+
   .000E 00   +              *
  2.500E-01    +                         *
  5.000E-01      +                                *
  7.500E-01         +                                    *
  1.000E 00             +                                     *
  1.250E 00                 +                                   *
  1.500E 00                      +                              *
  1.750E 00                          +                        *
  2.000E 00                              +                *
  2.250E 00                                 +        *
  2.500E 00                                    +
  2.750E 00                              *       +
  3.000E 00                        *              +
  3.250E 00                  *                    +
  3.500E 00             *                         +
  3.750E 00         *                            +
  4.000E 00      *                             +
  4.250E 00    *                              +
  4.500E 00   *                             +
  4.750E 00   *                           +
  5.000E 00    *                        +
  5.250E 00      *                    +
  5.500E 00        *                 +
  5.750E 00           *             +
  6.000E 00             *          +
  6.250E 00                *      +
  6.500E 00                   *   +
  6.750E 00                     * +
  7.000E 00                     *+
  7.250E 00                      +
  7.500E 00                       +
  7.750E 00                       *+
  8.000E 00                       * +
  8.250E 00                       *  +
  8.500E 00                      *   +
  8.750E 00                     *     +
  9.000E 00                    *       +
  9.250E 00                   *        +
  9.500E 00                 *          +
  9.750E 00                *           +
  1.000E 01               *            +
              +....+....+....+....+....+....+....+....+....+....+
```

:QUIT ↵

-

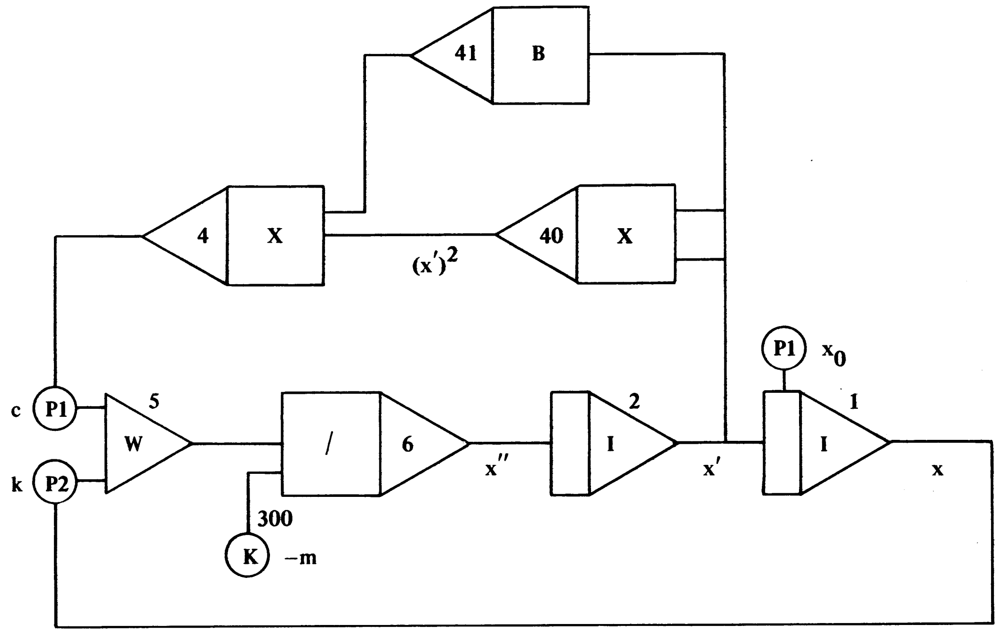
Example 2 – NONLINEAR SPRING, MASS, DAMPER SYSTEM

This example demonstrates the ease with which CSMP solves nonlinear problems. The user wishes to investigate the response of the spring, mass, and damper system in Example 1 when the damping is proportional to the square of the velocity. To make the damping force $c(x')^2$ act in the direction opposite to travel, the equation for the system is

$$mx'' + c(x')^2 + kx = 0 \quad \textit{for} \quad x' \geqslant 0$$

$$mx'' - c(x')^2 + kx = 0 \quad \textit{for} \quad x' < 0$$

A possible block diagram of the system is shown below.



```
-CSMP
```

```
: RECOVER SPRING
```

The user recovers and displays the configuration of Example 1, saved on file SPRING.

```
: DISPLAY ALL

BLOCK TYPE   E1     E2     E3       P1            P2            P3
300    K     0      0      0   -1.000E 00
  1    I     2      0      0   -1.000E 00
  5    W     2      1      0    5.000E-01    1.000E 00
  6    /     5    300      0
  2    I     6      0      0
```

```
DT:  2.500E-02

OUTPUT
FROM:  1.000E 01
BY:  2.500E-01
TO:  1.000E 01

RELERR:  1.000E-03

DTMIN:  2.500E-08

ALGORITHM: RK2

DATA FILE EMPTY

:BLOCKS↩
```
The user adds blocks and modifies the previous configuration.

```
BLOCK TYPE E1 E2 E3

40 X 2 2↩
41 B 2↩
4 X 41 40↩
5 W 4 1↩
PREVIOUS SPECIFICATION DELETED FOR BLOCK:     5
↩
BEGIN SORT

:INITIAL↩

BLOCK P1 P2 P3

5 .5 1↩
↩
:GO↩

   TIME            1             2            6

 1.000E 01    2.983E-01   -1.401E-01   -2.885E-01

:VARIABLE↩
```
The user checks the solution accuracy with a different integration algorithm.

```
:GO↩

   TIME            1             2            6

 1.000E 01    2.968E-01   -1.417E-01   -2.868E-01

:FROM 0↩
```
CSMP does not recompute the solution, but rather takes values from the scratch file.

```
:PLOT 1
AUTOMATIC SCALING? NO↩
YMAX = 1↩
YMIN = -1↩
PLOT BLOCK: 2↩
AUTOMATIC SCALING? YES↩
PLOT BLOCK:↩
```

```
: GO


BLOCK SYMBOL    MIN          MAX        INCREMENT
    1    +   -1.000E 00    1.000E 00    4.000E-02
    2    *   -5.013E-01    7.793E-01    2.561E-02

    TIME      +....+....+....+....+....+....+....+....+....+....+
   .000E 00   +                   *
  2.500E-01    +                           *
  5.000E-01      +                                  *
  7.500E-01         +                                     *
  1.000E 00              +                                    *
  1.250E 00                   +                                 *
  1.500E 00                       +                             *
  1.750E 00                            +                      *
  2.000E 00                                +               *
  2.250E 00                                    +       *
  2.500E 00                                       +
  2.750E 00                               *       +
  3.000E 00                          *              +
  3.250E 00                    *                    +
  3.500E 00              *                         +
  3.750E 00         *                             +
  4.000E 00     *                               +
  4.250E 00    *                               +
  4.500E 00   *                             +
  4.750E 00   *                          +
  5.000E 00    *                      +
  5.250E 00      *                 +
  5.500E 00         *            +
  5.750E 00            *       +
  6.000E 00                *  +
  6.250E 00                  +  *
  6.500E 00                  +      *
  6.750E 00                  +          *
  7.000E 00                   +            *
  7.250E 00                     +             *
  7.500E 00                       +            *
  7.750E 00                         +           *
  8.000E 00                            +        *
  8.250E 00                              +     *
  8.500E 00                                + *
  8.750E 00                                * +
  9.000E 00                             *     +
  9.250E 00                          *         +
  9.500E 00                       *            +
  9.750E 00                    *               +
  1.000E 01                 *                 +
              +....+....+....+....+....+....+....+....+....+....+
: DUMP NONLINEARSPRING↵
 NEW FILE↵
```

The user creates a binary file containing all program parameters.

```
: COPY TERMINAL TO DAMPING↵
 NEW FILE↵
```

The user creates a file, DAMPING, from which CSMP will take commands. The user investigates the effect of overshoot by varying P1 of block 5, the damping coefficient. The problem runs four times, with P1 equal to 1, 2, 4, and 8.

```
INITIAL↵
5 1 1↵
↵
```

The user must include the Carriage Return to terminate the INITIAL command.

```
GO↵
INITIAL↵
5 2 1↵
↵
GO
INITIAL↵
5 4 1↵
↵
GO↵
INITIAL↵
5 8 1↵
↵
GO↵
D^c
```

Control D terminates file creation.

```
: COPY DAMPING, TERMINAL↵
```

The user checks the file to see that it is correct.

```
INITIAL
5 1 1

GO
INITIAL
5 2 1

GO
INITIAL
5 4 1

GO
INITIAL
5 8 1

GO
```

```
BLOCK P1 P2 P3
```

P1 is set to 4.

```
BLOCK SYMBOL     MIN           MAX          INCREMENT
  1     +    -1.000E 00     1.000E 00      4.000E-02
  2     *    -9.772E-02     4.244E-01      1.044E-02

       TIME  +....+....+....+....+....+....+....+....+....+....+
   .000E 00  +        *
  2.500E-01   +                             *
  5.000E-01     +                                         *
  7.500E-01       +                                            *
  1.000E 00          +                                         *
  1.250E 00            +                                      *
  1.500E 00               +                                *
  1.750E 00                 +                            *
  2.000E 00                   +                       *
  2.250E 00                     +                  *
  2.500E 00                       +             *
  2.750E 00                         +        *
  3.000E 00                          +    *
  3.250E 00                           +*
  3.500E 00                        *  +
  3.750E 00                     *      +
  4.000E 00                  *          +
  4.250E 00               *             +
  4.500E 00            *                +
  4.750E 00         *                   +
  5.000E 00      *                      +
  5.250E 00    *                        +
  5.500E 00   *                        +
  5.750E 00  *                          +
  6.000E 00  *                         +
  6.250E 00  *                        +
  6.500E 00   *                       +
  6.750E 00    *                     +
  7.000E 00      *                   +
  7.250E 00       *                  +
  7.500E 00         *               +
  7.750E 00           *             +
  8.000E 00            *            +
  8.250E 00              *          +
  8.500E 00                *         +
  8.750E 00                 *        +
  9.000E 00                 *        +
  9.250E 00                 *         +
  9.500E 00                 *         +
  9.750E 00                 *         +
  1.000E 01                *           +
             +....+....+....+....+....+....+....+....+....+....+
BLOCK P1 P2 P3
```
P1 is set to 8.

```
BLOCK  SYMBOL     MIN            MAX          INCREMENT
   1     +    -1.000E 00     1.000E 00     4.000E-02
   2     *    -4.882E-02     3.208E-01     7.393E-03

    TIME      +....+....+....+....+....+....+....+....+....+....+
   .000E 00   +      *
  2.500E-01    +                                  *
  5.000E-01     +                                             *
  7.500E-01       +                                             *
  1.000E 00         +                                          *
  1.250E 00           +                                      *
  1.500E 00             +                                  *
  1.750E 00               +                              *
  2.000E 00                 +                          *
  2.250E 00                  +                       *
  2.500E 00                    +                   *
  2.750E 00                     +                *
  3.000E 00                      +             *
  3.250E 00                       +          *
  3.500E 00                        +       *
  3.750E 00                         +   *
  4.000E 00                          +*
  4.250E 00                         * +
  4.500E 00                       *    +
  4.750E 00                     *      +
  5.000E 00                   *         +
  5.250E 00                 *           +
  5.500E 00               *             +
  5.750E 00            *                 +
  6.000E 00          *                   +
  6.250E 00        *                     +
  6.500E 00      *                      +
  6.750E 00     *                       +
  7.000E 00    *                        +
  7.250E 00   *                         +
  7.500E 00   *                        +
  7.750E 00   *                        +
  8.000E 00    *                       +
  8.250E 00     *                      +
  8.500E 00      *                    +
  8.750E 00       *                   +
  9.000E 00        *                  +
  9.250E 00         *                 +
  9.500E 00           *               +
  9.750E 00            *              +
  1.000E 01             *             +
              +....+....+....+....+....+....+....+....+....+....+
```

The system appears to be well damped for c=8. The user notes that the maximum overshoot occurs between time=5 and time=7. The user calculates the maximum value by using the automatic scaling feature of the PLOT command.

```
:PLOT 1 ↵
AUTOMATIC SCALING? YES ↵
PLOT BLOCK: ↵

:FROM 5 TO 7 ↵
```

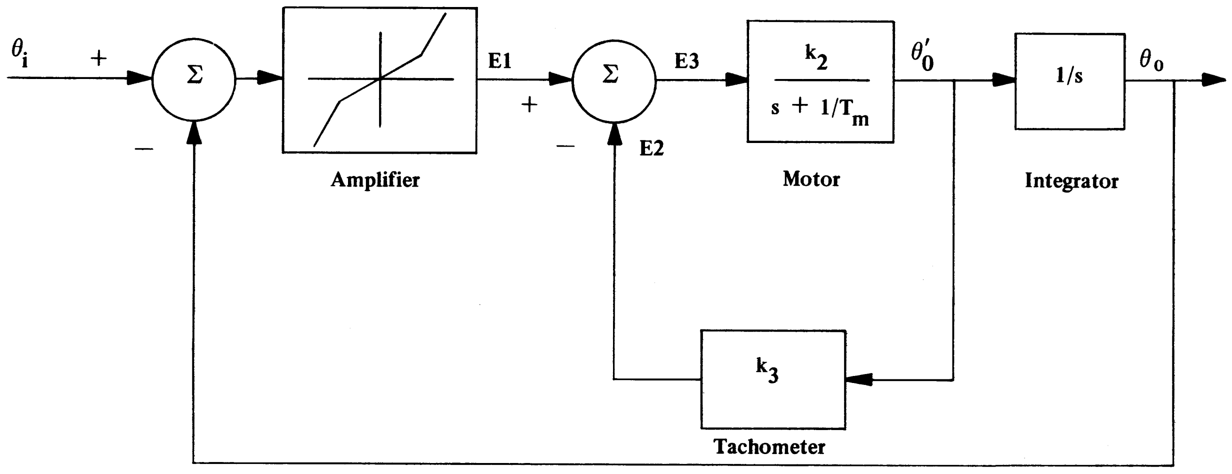
Note that CSMP allows the output of any block in any interval from 0 to 10.

```
:GO ↵


BLOCK SYMBOL    MIN           MAX          INCREMENT
  1     +     2.548E-02    6.223E-02     7.350E-04

    TIME            +....+....+....+....+....+....+....+....+....+....+
  5.000E 00         +
  5.250E 00                              +
  5.500E 00                                             +
  5.750E 00                                                       +
  6.000E 00                                                           +
  6.250E 00                                                         +
  6.500E 00                                                   +
  6.750E 00                                         +
  7.000E 00                            +
                    +....+....+....+....+....+....+....+....+....+....+

:QUIT ↵

-
```

The maximum overshoot occurred at time=6 and equaled .06223.

Example 3 – POSITION CONTROLLING SERVO SYSTEM

This example analyzes a simple position controlling servomechanism with rate feedback from a tachometer. The illustration below shows a model of the system and a list of equations which describe the model.



where

θ_i = Commanded position

θ_0 = Motor position

ϵ = $\theta_i - \theta_0$ = Position error

E1 = $k_1\epsilon$ = Amplifier voltage

E2 = $k_3\theta_0'$ = Tachometer voltage

E3 = E1 − E2 = Error voltage

$$\theta_0' = \left(\frac{k_2}{s + 1/T_m} \cdot \mathrm{E3}\right) = \text{Motor velocity}$$

and

k_1 = Amplifier gain (volts/radian)

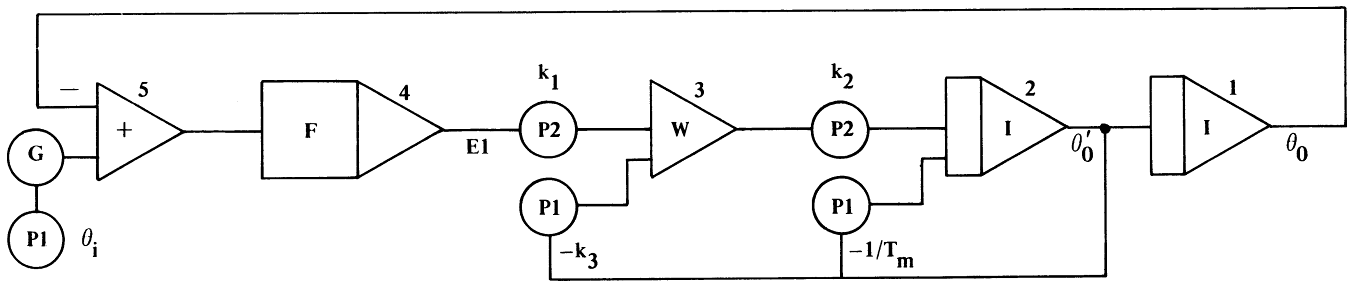
k_2 = Motor velocity constant (radians/volt-second) = 1

k_3 = Tachometer voltage constant (volt-seconds/radian)

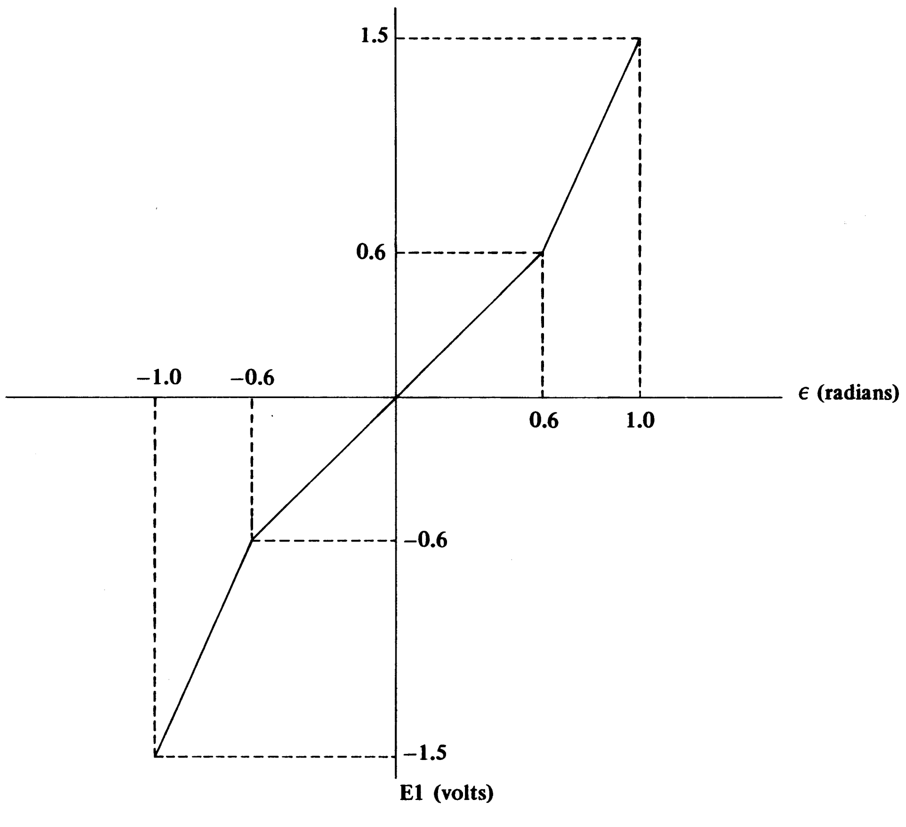
T_m = Motor electrical time constant = 0.1

s = $j\omega$

The CSMP block diagram below corresponds to this example.



Of special importance in this example is the use of the function generator to describe the output of the amplifier. The gain of the amplifier is proportional to the position error (ϵ) such that:



All function generator blocks require three statements to describe the function. For this example, the statements and the values are:

1. Configuration statement, specifying the block interconnections: 4, F, 5.
2. Parameter statement, specifying the maximum and minimum x-axis values: 3, 1.0, –1.0.
3. Function generator statement, requesting the values of the function at 11 points along the x-axis, evenly spaced between the minimum value, –1.0, and the maximum value, 1.0. Thus, values must be supplied for $f(\epsilon)=E1$ as ϵ ranges from –1.0 to 1.0 in increments of 0.2.

The table on the right is constructed from the equations below, which describe the function.

$$
\begin{aligned}
f(\epsilon) &= 2.25\epsilon + .75 && \textit{if} && -1.0 \leqslant \epsilon \leqslant -0.6 \\
&= \epsilon && \textit{if} && -0.6 < \epsilon < 0.6 \\
&= 2.25\epsilon - .75 && \textit{if} && 0.6 \leqslant \epsilon \leqslant 1.0
\end{aligned}
$$

ϵ	E1
–1.0	–1.5
–0.8	–1.05
–0.6	–0.6
–0.4	–0.4
–0.2	–0.2
0.0	0.0
0.2	0.2
0.4	0.4
0.6	0.6
0.8	1.05
1.0	1.5

```
-CSMP↵

:BLOCKS↵

BLOCK TYPE E1 E2 E3

1 I 2↵
2 I 0 2 3↵
3 W 2 4↵
4 F 5↵
5 + -1 6↵
6 K↵
↵
BEGIN SORT
```

```
:INITIAL↵

BLOCK P1 P2 P3

2 0 -10 1↵
3 -1 1↵
6 1↵
↵

FUNCTION GENERATOR SPECIFICATIONS        CSMP requests function generator y values.

BLOCK: 4↵
INT( 0) = -1.5↵
INT( 1) = -1.05↵
INT( 2) = -.6↵
INT( 3) = -.4↵
INT( 4) = -.2↵
INT( 5) = 0↵
INT( 6) = .2↵
INT( 7) = .4↵
INT( 8) = .6↵
INT( 9) = 1.05↵
INT(10) = 1.5↵

P1 MUST BE > P2 FOR BLOCK:     4   P1 and P2 were not specified above as
P1 = 1↵                            parameters; CSMP requests them here.
P2 = -1↵

BLOCK:↵    A Carriage Return ends specifications.

:DUMP SERVO↵
 NEW FILE↵

:VARIABLE↵

:PRINT 1↵

:FROM 10 BY .5 TO 10↵

:DISPLAY DT↵
 5.000E-02
```

```
: GO ↩

   TIME           1

 1.000E 01   6.380E-01

: PLOT 1 ↩
AUTOMATIC SCALING? NO ↩
YMAX = 1 ↩
YMIN = 0 ↩
PLOT BLOCK: 2 ↩
AUTOMATIC SCALING? YES ↩
PLOT BLOCK: ↩

: FROM 0 ↩
```

The CSMP scratch file contains values from 0 to 10 in intervals of .5. The user may display the contents of any interval on the scratch file without recomputing the solution.

```
: GO ↩


BLOCK SYMBOL    MIN           MAX          INCREMENT
    1    +    .000E 00     1.000E 00     2.000E-02
    2    *    .000E 00     1.270E-01     2.541E-03

  TIME      +....+....+....+....+....+....+....+....+....+....+
 .000E 00   +
5.000E-01      +                                              *
1.000E 00         +                                      *
1.500E 00           +                                *
2.000E 00              +                         *
2.500E 00                +                   *
3.000E 00                  +              *
3.500E 00                    +         *
4.000E 00                      +    *
4.500E 00                       + *
5.000E 00                        +
5.500E 00                       *  +
6.000E 00                      *    +
6.500E 00                     *      +
7.000E 00                    *        +
7.500E 00                    *         +
8.000E 00                   *           +
8.500E 00                  *             +
9.000E 00                 *               +
9.500E 00                 *                +
1.000E 01                *                  +
            +....+....+....+....+....+....+....+....+....+....+

: QUIT ↩

-
```

Example 4 — FALLING BALL PROBLEM

This example demonstrates the use of a user-defined block in a CSMP problem. The equation below can be solved in CSMP without the aid of a user-defined block. The block is used for the purpose of demonstration.

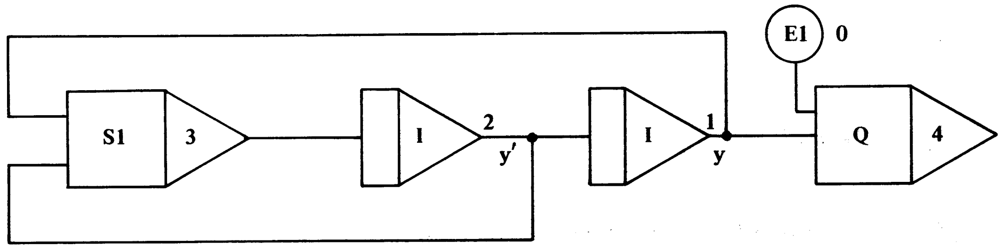
The user wishes to determine the time required for a ball dropped from height y_0 above the earth to hit the ground. He wishes to plot its position, velocity, and acceleration for $y_0=15{,}000$ feet.

The equation representing the motion of a falling body is

$$\frac{d^2y}{dt^2} = -g + A(y)\left(\frac{dy}{dt}\right)^2$$

where the term $A(y)=.001e^{-.00004328y}$ is proportional to the standard atmospheric weight density of air, and $g=32.17$, the acceleration due to gravity, assumed constant. All units are in feet, seconds, feet/second, and feet/second^2.

The block diagram is illustrated below. The quit block, Q, terminates the execution when the height y = 0.



The CSMP user writes a subroutine to calculate the right side of the equation.

```
-EDITOR↵
*APPEND↵
FUNCTION S1(Y,YDOT,E3,P1,P2,P3)↵
S1=-P1+P2*EXP(P3*Y)*YDOT*YDOT↵
RETURN↵
END↵
*WRITE SUBS1 ↵
 NEW FILE↵
73 CHARACTERS
*QUIT↵
```

```
-BFORTRAN ↵

+COMPILE SUBS1,CSMPSUBS ↵  The subroutines must be on the file CSMPSUBS.
 NEW FILE ↵

FUNCTION S1(Y,YDOT,E3,P1,P2,P3)
END

+QUIT ↵

-CSMP ↵

:BLOCKS ↵

BLOCK TYPE E1 E2 E3

1 I 2 ; Y ↵        Comments follow the semicolon.
2 I 3 ; YDOT ↵
3 S1 1 2 ↵
4 Q 0 1 ↵
↵
BEGIN SORT

:INITIAL ↵

BLOCK P1 P2 P3

1 15000 ↵
3 32.17 .001 -.00004328 ↵
↵

:SUBROUTINES ↵        The user loads his compiled subroutines.

LOADING LIBRARY

WORDS USED
  PROGRAM: 5740
  STORAGE: 2181
  SHARED:  5550
  DEBUG:   138
WORDS UNUSED
  PROGRAM: 2159       CSMP indicates the number of words of
  SHARED:  594        core available for the user's subroutines.

:DISPLAY BLOCKS ↵

BLOCK  TYPE   E1     E2     E3      P1            P2            P3
  1     I     2      0      0    1.500E 04
  3     S1    1      2      0    3.217E 01    1.000E-03    -4.328E-05
  2     I     3      0      0
  4     Q     0      1      0
```

```
:TO 100 BY 1 FROM 100↵

:PRINT 1↵

:GO NO HEADING ↵

RUN TERMINATED IN BLOCK:     4

:PROBE 301 1 2 3↵    Block 301 is time.
301:   7.600E 01    The ball hits the earth before time=76 seconds.
  1: -5.346E 01
  2: -1.812E 02
  3:  7.267E-01

:FROM 75 TO 75↵

:PRINT 1 2 3↵

:VARIABLE↵    The user chooses a different algorithm.

:GO↵

   TIME            1            2            3

 7.500E 01    1.285E 02   -1.819E 02    7.327E-01    The ball is 128.5 feet above
                                                     the ground after 75 seconds.
:DISPLAY RELERR↵
 1.000E-03

:RELERR .0001↵      The user makes RELERR smaller for better solution accuracy.

:GO NO HEADING↵

 7.500E 01    1.285E 02   -1.819E 02    7.327E-01    He gets the same answers,
                                                     indicating a good solution.
:SAVE↵ This command saves all data necessary to continue the computation from time=75.

:DUMP BREAK↵  The user is called away.  He saves the program status on the
 NEW FILE↵    file BREAK and the subroutine on the file BALLSUB.


:DUMP SUBROUTINES BALLSUB ↵
 NEW FILE↵

:QUIT↵

-
```

```
-CSMP ↵

:RECOVER SUBROUTINES BALLSUB↵
```

The user returns and resumes his previous status with two RECOVER commands.

```
:RECOVER BREAK ↵


:FROM 76 BY .01 TO 76 ↵
```

The output interval is smaller, .01.

```
:GO N↵
```

N indicates no heading.

```
RUN TERMINATED IN BLOCK:     4

:PROBE 301 1↵
301:   7.571E 01
  1:  -4.140E-01
```

The ball hits the ground between time=75.70 and 75.71 seconds.

```
:FROM 75.7↵

:GO↵

    TIME              1              2              3

  7.570E 01    1.400E 00    -1.814E 02    7.285E-01
```

The velocity just before impact is approximately 181.4 feet per second.

```
:FROM 0 BY 2 TO 74 ↵
```

The user plots the ball's position, velocity, and acceleration from time=0 to 74 seconds.

```
:PLOT 1↵
AUTOMATIC SCALING? YES↵
PLOT BLOCK: 2 ↵
AUTOMATIC SCALING? YES↵
PLOT BLOCK: 3 ↵
AUTOMATIC SCALING? YES↵
PLOT BLOCK:↵
```

: GO

```
BLOCK SYMBOL    MIN          MAX        INCREMENT
   1     +    3.108E 02   1.500E 04    2.938E 02
   2     *   -2.323E 02    .000E 00    4.645E 00
   3     .   -3.217E 01   1.067E 00    6.647E-01

    TIME      +....+....+....+....+....+....+....+....+....+....+
   .000E 00   .                                                 +
  2.000E 00      .                                 *            +
  4.000E 00              .             *                       +
  6.000E 00                  *     .                          +
  8.000E 00            *                    .                +
  1.000E 01        *                               .        +
  1.200E 01     *                                       . +
  1.400E 01    *                                         + .
  1.600E 01   *                                        +     .
  1.800E 01   *                                       +        .
  2.000E 01   *                                     +          .
  2.200E 01   *                                   +             .
  2.400E 01    *                                 +              .
  2.600E 01    *                               +                .
  2.800E 01     *                             +                 .
  3.000E 01     *                           +                   .
  3.200E 01      *                         +                    .
  3.400E 01      *                       +                      .
  3.600E 01       *                     +                       .
  3.800E 01       *                   +                         .
  4.000E 01       *                  +                          .
  4.200E 01        *               +                            .
  4.400E 01        *              +                             .
  4.600E 01         *            +                              .
  4.800E 01         *          +                                .
  5.000E 01         *         +                                 .
  5.200E 01          *      +                                   .
  5.400E 01          *     +                                    .
  5.600E 01           *   +                                     .
  5.800E 01           * +                                       .
  6.000E 01           *+                                        .
  6.200E 01           +*                                        .
  6.400E 01         +  *                                        .
  6.600E 01        +   *                                        .
  6.800E 01       +     *                                       .
  7.000E 01      +      *                                       .
  7.200E 01    +        *                                       .
  7.400E 01   +          *                                      .
              +....+....+....+....+....+....+....+....+....+....+
```

Note that the velocity is negative, as it should be. The acceleration is in the negative direction for approximately 20 seconds, then becomes slightly positive as the air molecules are providing a greater force than that due to gravity. The ball hits the ground at 181.4 feet per second, or 124 miles per hour, as determined on page 86. In the thinner atmosphere, the ball reaches a maximum velocity of 232.2 feet per second, or 158 miles per hour.

The user wishes to determine the time required to fall from 100 feet, 500 feet, 1000 feet, 10,000 feet, and 100,000 feet. Instead of giving each command individually, he creates and executes the command file HEIGHT.

```
:COPY TERMINAL,HEIGHT
 NEW FILE

INITIAL
1 100

GO N
PROBE 301 1 2 3
INI
1 500

GO N
PROBE 301,1,2,3
INI
1 1000

GO N
PRO 301 1 2 3
INI
1 1E4

GO N
PRO 301 1 2 3
INI
1 1E5

GO N
PRO 301 1 2 3
D^C
```

```
:BY 1 TO 1000 FROM 1000

:PRINT 1

:DO HEIGHT
```

CSMP takes commands from the file HEIGHT.

```
BLOCK P1 P2 P3
```

The initial value for block 1 is changed.

```
RUN TERMINATED IN BLOCK:      4
301:   2.550E 00
  1:  -1.146E 00
  2:  -7.676E 01
  3:  -2.628E 01
```

y_0 = 100 feet
time = 2.55 seconds

```
BLOCK P1 P2 P3


RUN TERMINATED IN BLOCK:      4
301:   6.050E 00
  1:  -9.360E-01
  2:  -1.429E 02
  3:  -1.176E 01
```

y_0 = 500 feet
time = 6.05 seconds

```
BLOCK P1 P2 P3


RUN TERMINATED IN BLOCK:      4
301:   9.250E 00
  1:  -7.157E 00
  2:  -1.676E 02
  3:  -4.061E 00
```

y_0 = 1000 feet
time = 9.25 seconds

```
BLOCK P1 P2 P3


RUN TERMINATED IN BLOCK:      4
301:   5.500E 01
  1:  -1.309E 02
  2:  -1.809E 02
  3:   7.241E-01
```

y_0 = 10,000 feet
time = 55 seconds

```
BLOCK P1 P2 P3


RUN TERMINATED IN BLOCK:      4
301:   2.390E 02
  1:  -1.984E 01
  2:  -1.813E 02
  3:   7.278E-01
```

y_0 = 100,000 feet
time = 239 seconds
After y_0 = 1000 feet, the terminal velocity is almost constant.

```
:PLOT 2↵
AUTOMATIC SCALING? YES↵
PLOT BLOCK: 3↵
AUTOMATIC SCALING? Y↵
PLOT BLOCK: ↵

:FROM 0↵

:GO↵
```

The scratch file contains the data from the previous run. With recomputation, the user wishes to find the maximum and minimum values of velocity and acceleration.

```
BLOCK SYMBOL    MIN            MAX          INCREMENT
   2     +    -9.210E 02      .000E 00     1.842E 01
   3     *    -3.217E 01     1.247E 01     8.927E-01

    TIME      +....+....+....+....+....+....+....+....+....+....+
  .000E 00    *                                                  +
 1.000E 00    *

:QUIT↵
```

The user wanted only the maximum and minimum values, not the plot. He hits the Alt Mode/Escape. From 100,000 feet, the maximum velocity is 921 feet per second, or 628 miles per hour.

APPENDIX A
CSMP COMMAND SUMMARY

The table below presents a concise list of CSMP commands and their descriptions. Most CSMP commands can be abbreviated to a left subset of the complete command. The minimum abbreviations are listed in the table. For example, the CLOSE command may be given as CLO, CLOS, or CLOSE.

Command	Minimum Abbreviation	Description
ALGORITHM	ALG	DISPLAY ALGORITHM prints the current integration routine.
BLOCKS	BLO	Specifies, alters, or adds block configuration specifications from the terminal.
BY	BY	Specifies output time interval.
CAPABILITIES	CA	Describes program capabilities.
CHARGES	CH	Lists additional cost, if any.
CLOSE	CLO	Closes specified input or output files.
CONTINUE	CON	Continues execution of the solution.
COPY	COP	Performs same function as the EXECUTIVE COPY command.
CREDITS	CR	Prints credits for implementation of the program.
DEBUG	DEB	Transfers control to BATCH FORTRAN IV for debugging subroutines.
DELETE	DEL	Deletes specified program quantities.
DISPLAY	DI	Prints specified program quantities.
DO	DO	Takes CSMP commands from a file.
DOUBLE	DOU	Prints numeric output with eight-digit accuracy.
DT	DT	Specifies integration interval or step size.
DTMIN	DTM	Specifies minimum integration interval for the VARIABLE integration routine.
DUMP	DU	Creates a binary file containing all program specifications and user-defined blocks, if any.

Command	Minimum Abbreviation	Description
DUMP ALL	DU ALL	Creates a binary file containing all program specifications and all solution data currently in the scratch file.
DUMP SUBROUTINES	DU SUB	Creates a binary file containing the user's subroutines to be linked with CSMP.
ERASE	ER	Erases all current data from the scratch file.
EULER	EU	Specifies the EULER integration routine.
EXPERT	EXP	Prints minimum information when requesting data.
FROM	FR	Specifies output start time.
FUNCTION	FU	Specifies or alters function generator specifications from the terminal.
GO	GO	Begins program computation or reads data from the scratch file.
GO NO HEADING	GO N	Same as GO, except headings are not printed.
HELP *or* ?	H	Prints a list of CSMP commands with descriptions.
INITIAL	INI	Specifies, alters, or adds initial conditions and/or parameters from the terminal.
INSTRUCTIONS	INS	Prints a list of instructions to execute the program.
IPLOT	IP	Specifies blocks to be plotted without automatic scaling.
NONEXPERT	N	Reverses condition set by the EXPERT command.
OUTPUT	O	Writes the output from the last PRINT or PLOT command on the specified file.
PLOT	PL	Specifies the blocks to be plotted, with or without automatic scaling.
PRINT	PRI	Specifies blocks to be printed in tabular form.
PROBE	PRO	Prints the last computed value for the specified blocks.
QUIT	Q	Returns the user to the EXECUTIVE and deletes all scratch files.

Command	Minimum Abbreviation	Description
READ	REA	Reads the block configuration specifications, initial conditions and parameters, and function generator specifications from the specified symbolic file.
RECOVER	REC	Loads the specified binary file created by the DUMP command.
RECOVER ALL	REC ALL	Loads the specified binary file created by the DUMP ALL command.
RECOVER SUBROUTINES	REC SU	Loads the specified file created by the DUMP SUBROUTINES command.
RELERR	REL	Specifies the relative error used by the VARIABLE integration routine.
RK2	RK2	Specifies the Runge-Kutta second order integration routine.
RK4	RK4	Specifies the Runge-Kutta fourth order integration routine.
RUN	RU	Requests program information and initial conditions and begins computation.
SAVE	SA	Saves all parameters at the time the command is given that are necessary to begin computation. Erases all data on the scratch file up to the time the command was given.
SCRATCH	SC	Displays scratch file information.
SIMPSONS	SIM	Specifies the Simpson's Rule integration routine.
SINGLE	SIN	Prints numerical output in four-digit accuracy.
SUBROUTINES	SU	Loads the user's subroutines into CSMP.
TIMING	TI	Prompts the user for the timing specifications.
TO	TO	Specifies the output end time.
TRAPEZOIDAL	TR	Specifies the trapezoidal integration routine.
VARIABLE	VA	Specifies the variable step size integration routine.
VERSION	VE	Prints present version number of CSMP.
WRITE	W	Saves the current block configuration specifications, initial conditions and parameters, and function generator specifications on the symbolic file named by the user.

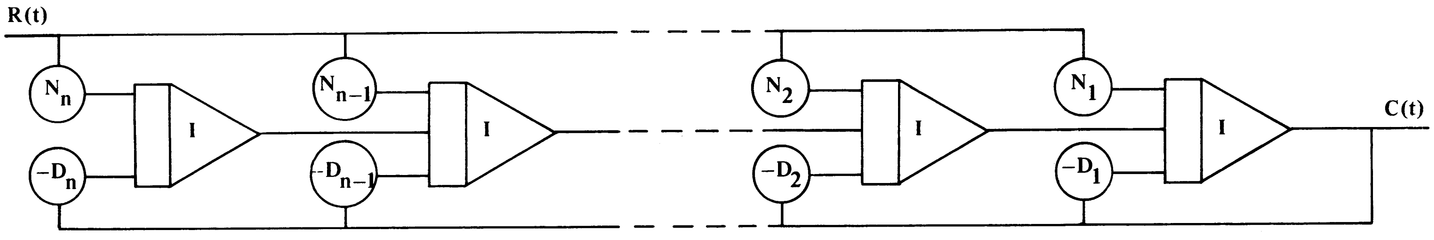
APPENDIX B
LAPLACE TRANSFORM MODELING TECHNIQUES

This appendix demonstrates an efficient method for modeling a Laplace transform expressed as the ratio of two polynomials in s, where s=jω. For example, the general Laplace transform may be expressed as:

$$G(s) = \frac{N_1 s^{n-1} + N_2 s^{n-2} + \cdots + N_{n-1} s + N_n}{s^n + D_1 s^{n-1} + D_2 s^{n-2} + \cdots + D_{n-1} s + D_n}$$

Note that the order of the numerator must be less than that of the denominator.

A block diagram suitable for the CSMP solution of the equation above is shown below. The inverse of G(s), $L^{-1}[G(s)]$, is C(t)/R(t), where t represents time.



The CSMP statements to achieve this solution are listed below. Assume that K_0 is the block number for the output, C(t), and K_i is the block number of the input, R(t), and $K_1, K_2, \ldots,$ K_n are block numbers such that

$$\begin{aligned} K_1 &= K_0 + 1 \\ K_2 &= K_1 + 1 \\ &\vdots \\ K_n &= K_{n-1} + 1 \end{aligned}$$

The configuration for the equation of G(s) becomes:

Block	Type	E1	E2	E3	P1	P2	P3
K_0	I	K_1	K_i	K_0	0	N_1	$-D_1$
K_1	I	K_2	K_i	K_0	0	N_2	$-D_2$
K_2	I	K_3	K_i	K_0	0	N_3	$-D_3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
K_{n-1}	I	K_n	K_i	K_0	0	N_{n-1}	$-D_{n-1}$
K_n	I	0	K_i	K_0	0	N_n	$-D_n$

APPENDIX C
DECIMAL REPRESENTATIONS OF ASCII CHARACTERS

Decimal Representation	ASCII Character
0	blank
1	!
2	”
3	#
4	$
5	%
6	&
7	’
8	(
9	)
10	*
11	+
12	,
13	—
14	.
15	/
16	0
17	1
18	2
19	3
20	4
21	5
22	6
23	7
24	8
25	9
26	:
27	;
28	<
29	=
30	>
31	?
32	@
33	A
34	B
35	C
36	D
37	E
38	F
39	G
40	H
41	I
42	J
43	K
44	L
45	M
46	N
47	O
48	P
49	Q
50	R
51	S
52	T
53	U
54	V
55	W
56	X
57	Y
58	Z
59	[
60	\
61	]
62	↑
63	←

Decimal Representation	ASCII Character
64	`
65	a
66	b
67	c
68	d
69	e
70	f
71	g
72	h
73	i
74	j
75	k
76	l
77	m
78	n
79	o
80	p
81	q
82	r
83	s
84	t
85	u
86	v
87	w
88	x
89	y
90	z
91	{
92	|
93	}
94	~
95	RUBOUT

Decimal Representation	ASCII Character[1]
96	P^{cs}
97	A^{c}
98	B^{c}
99	C^{c}
100	D^{c}
101	E^{c}
102	F^{c}
103	G^{c}
104	H^{c}
105	I^{c}
106	J^{c}
107	K^{c}
108	L^{c}
109	M^{c}
110	N^{c}
111	O^{c}
112	P^{c}
113	Q^{c}
114	R^{c}
115	S^{c}
116	T^{c}
117	U^{c}
118	V^{c}
119	W^{c}
120	X^{c}
121	Y^{c}
122	Z^{c}
123	K^{cs}
124	L^{cs}
125	M^{cs}
126	N^{cs}
127	O^{cs}

1 - Superscript c stands for a control character; superscript cs stands for a control shift character.

INDEX

NOTE: Page numbers which appear in **bold** *face type refer to those pages where the listed item receives the most detailed discussion.*

USER EVALUATION

Tymshare would like to improve the quality and usefulness of its publications. However, to achieve this goal, we need your help and critical evaluations. Will you please provide us with such constructive information by filling out this questionnaire and mailing it back to us?

1. (a) Is this manual a useful document? ☐ Yes ☐ No
   (b) If your answer is Yes, what features make it useful.
   (c) If your answer is No, what features prevent it from being a useful document.

2. (a) Is the text clear and readily understandable? ☐ Yes ☐ No
   (b) If your answer is No, please cite the sections, subsections, or paragraphs that are unclear or difficult to understand.

3. (a) Are you pleased with the organization of this manual? ☐ Yes ☐ No
   (b) Should the organization be changed? ☐ Yes ☐ No
   (c) What changes do you suggest?

4. (a) Are the example problems helpful and easy to understand? ☐ Yes ☐ No
   (b) Should more examples be added when this manual is revised? ☐ Yes ☐ No
   (c) What kind of programs would you like to see added?

5. (a) Should anything be deleted from this manual when it is revised? ☐ Yes ☐ No
   (b) If your answer is Yes, give us your suggestions.

6. List any further suggestions you have for the improvement of this manual.

OPTIONAL INFORMATION:

Name ______________________ Company ______________________

Street Address ______________________

City ______________ State ______________ Zip ______________

Occupation ______________________

THANK YOU FOR YOUR ASSISTANCE.

To mail this questionnaire, simply cut from manual, fold Part A back, fold Part B back, and staple where indicated. **Part A**

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

First Class
Permit No.
300
Los Altos,
California

POSTAGE WILL BE PAID BY

TYMSHARE, INC.
20705 Valley Green Drive
Cupertino, California 95014
ATTN: **Documentation Group**

Part B

STAPLE HERE