DSL/90—A DIGITAL SIMULATION PROGRAM FOR CONTINUOUS SYSTEM MODELING

W. M. Syn
Systems Development Division
and
Robert N. Linebarger
Data Processing Division
IBM Corporation, San Jose, California

INTRODUCTION

Computer simulation has been used for some time in the analysis and design of dynamic systems. With recent advancements in computer performance, the field of dynamic simulation—long the exclusive domain of the analog computer—has begun to utilize digital methods. No less than a score of digital simulation programs have appeared since R. G. Selfridge's pioneering effort in 1955; and the number is ever-increasing. These programs offer a convenient method of simulating continuous system dynamics employing well-known and easy-to-use analog computer programming techniques. The common starting point for such simulation is the conventional analog block diagram, and the common approach is the breakdown of the mathematical system model into its component parts or functional blocks. These blocks, having a near one-to-one correspondence with analog computing elements such as integrators, summers, limiters, etc., usually appear as subroutines within the simulator program. Using one of the simulation packages, "programming" involves no more than merely interconnecting the functional blocks by a sequence of connection statements according to the rules laid down by the input language. This interconnecting of blocks is analogous to the wiring of the patchboard on an analog computer. Therefore, these digital-analog simulation programs combine the best features of the analog and digital computers: the flexibility of block connection structure of the former and the accuracy and reliability of the latter.

DSL/90 is a new digital simulation package for the 7090 family of computers. The program is available from the SHARE library (IWDSL No. 3358). Its development, from drawing board to production code, was guided by the following broad objectives:

- To incorporate within it all the desirable and proven features of its predecessors;
- To make this useful technique of digital simulation attractive to a group of users who are not analog-computer-oriented, yet retain the large following of analog programmers who are devoted to the building-block approach to system analysis;
- To provide a "continuous system simulator" program that is applicable to a broad range of continuous system analysis and not restrained by conventional digital-analog simulator techniques.

Some of the DSL/90 features are:

- A library of DSL system blocks such as integrator, limiter, summer, etc.;
- A simple nonprocedural applications-oriented input language specifying the rules for connecting the library blocks together;
- An input routine which permits quick and easy parameter entry and data changes;
- Complete print output routines including a graphical output facility;
- Choice of numerical integration routines with or without error bounds using centralized or noncentralized integration schemes;
- Automatic sequencing of input language statements (this is called "sorting" in programs such as ASTRAL and MIDAS);
- Facility to add to the DSL/90 library any user-defined blocks in the form of subroutines (FORTRAN, MAP or binary decks);
- Intermixing of DSL and FORTRAN language statements;
- Repeatability of language statements (macro-generation);
- Dynamic storage of data.

Although DSL/90's input language statements are block-oriented, they are not restricted solely to block notation. DSL/90 permits an intermixing of its input language statements (henceforth called DSL statements) and FORTRAN IV statements. Thus, the power of FORTRAN is made available to the problem solver. One far-reaching implication of this language feature is that simulation "programming" may begin anywhere from the analog block diagram formulation of the problem to the higher-level mathematical model in the form of ordinary differential equations.

OPERATIONAL FEATURES

Basic Language Features

The DSL/90 language statements may be classified into three general categories: 1) structure or connection statements which define the interconnection of the functional blocks, 2) data statements which permit the entry of alphanumeric information, and 3) simulation control statements.

The Connection Statements. In the DSL/90 input language, the basic functional block is characterized by an output (outputs) that is functionally related to one or more inputs. Parameter names and initial conditions, if any, are also included in the statement which has the following general form:

Outputs = Block name (Initial conditions, Parameters, Inputs)

Below are examples of basic DSL connection or structure statements:

1. OUTNAM = SQRT (TEMP)

In the block diagram representation (Fig. 1), SQRT is the name of the functional block. It has a single input called TEMP and the output is given the name OUTNAM.

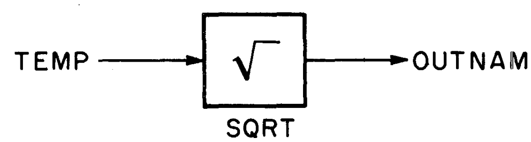


Figure 1.

2. Y = INTGRL (IC2, YDOT)

Figure 2 represents the block INTGRL which is the basic DSL/90 integrator block. IC2 and YDOT are its initial condition and input name respectively.

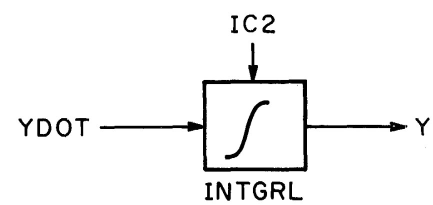


Figure 2.

3. OUT1, OUT2 = VALVE (LEVEL, INHI, INMED, INLO)

Figure 3 illustrates a user-supplied functional block named VALVE with two outputs OUT1 and OUT2. LEVEL is a unique parameter name se-

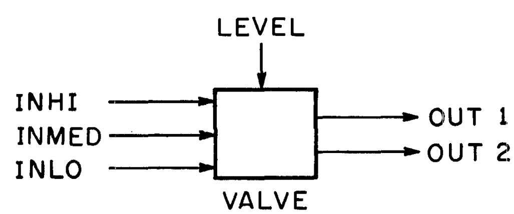


Figure 3.

lected by the user, and INHI, INMED and INLO are the names of the three input variables to the block.

From the above illustrations, it should be evident that a functional block in the DSL/90 language is completely specified by the unique names assigned to the inputs and outputs of each block. The user is free to select names meaningful to his process simulation, the only restriction being that a name consists of no more than 6 alphanumeric characters, the first of which is alphabetic. User-supplied blocks may have any name following the same restriction above. However, the names of standard blocks supplied as part of the DSL/90 simulation package are preassigned. DSL/90 provides an extensive library of functional blocks which are listed in Table 1.

The above format for characterizing functional blocks in DSL/90 is consistently adhered to. However, there are these exceptions: the basic operations of multiplying, dividing, summing and subtracting are replaced by the operators *,/, + and −, respectively. To this list of operators we add ** for exponentiation. Let us illustrate one of these operations by simulating a multiplier output (Fig. 4),

$$OUT = A \cdot B.$$

Table 1. Functional Description of Standard DSL/90 Blocks

	GENERAL FORM	FUNCTION
**	Y = INTGRL (IC, X) Y(O) = IC INTEGRATOR	$Y = \int_0^t X\,dt + IC$ EQUIVALENT LAPLACE TRANSFORM : $\frac{1}{S}$
*	Y = MODINT (IC, P_1, P_2, X) MODE-CONTROLLED INTEGRATOR	$Y = \int_0^t X\,dt + IC$ $P_1 = 1,\ P_2 = 0$ Y = IC $P_1 = 0,\ P_2 = 1$ Y = LAST OUTPUT $P_1 = 0,\ P_2 = 0$
*	Y = REALPL (IC, P, X) Y(O) = IC 1ST ORDER SYSTEM (REAL POLE)	$P\dot{Y} + Y = X$ EQUIVALENT LAPLACE TRANSFORM : $\frac{1}{PS+1}$
*	Y = LEDLAG (IC, P_1, P_2, X) Y(O) = IC LEAD-LAG	$P_2\dot{Y} + Y = P_1\dot{X} + X$ EQUIVALENT LAPLACE TRANSFORM : $\frac{P_1S+1}{P_2S+1}$
*	Y = CMPXPL (IC_1, IC_2, P_1, P_2, X) Y(O) = IC_1 $\dot{Y}$(O) = IC_2 2ND ORDER SYSTEM (COMPLEX POLE)	$\ddot{Y} + 2P_1P_2\dot{Y} + P_2^2Y = X$ EQUIVALENT LAPLACE TRANSFORM : $\frac{1}{S^2+2P_1P_2S+P_2^2}$
	Y = DERIV (IC, X) Y(O) = IC DERIVATIVE	$Y = \frac{dx}{dt}$ QUADRATIC INTERPOLATION EQUIVALENT LAPLACE TRANSFORM : S
	Y = DELAY (N, P, X) P = TOTAL DELAY IN TERMS OF INDEPENDENT VAR. N = MAX NO. OF POINTS DELAY DEAD TIME (DELAY)	$Y(t) = X(t-P)$ $t \geq P$ $Y = 0$ $t < P$ EQUIVALENT LAPLACE TRANSFORM : e^{-PS}
	Y = ZHOLD (P, X) Y(O) = O ZERO-ORDER HOLD	Y = X P = 1 Y = LAST OUTPUT P = 0 EQUIVALENT LAPLACE TRANSFORM : $\frac{1}{S}(1-e^{-st})$
	Y = IMPL (IC, ERROR, FUNCT) IMPLICIT FUNCTION	Y = IC t = 0 FIRST ENTRY Y = FUNCT(Y) $t \geq 0$ $\lvert Y - FUNCT(Y)\rvert \leq ERROR \cdot \lvert Y\rvert$

SWITCHING FUNCTIONS

$Y = FCNSW\ (P, X_1, X_2, X_3)$ FUNCTION SWITCH	$Y = X_1 \quad P < 0$ $Y = X_2 \quad P = 0$ $Y = X_3 \quad P > 0$
$Y = INSW\ (P, X_1, X_2)$ INPUT SWITCH (RELAY)	$Y = X_1 \quad P < 0$ $Y = X_2 \quad P \geq 0$
$Y_1, Y_2 = OUTSW\ (P, X)$ OUTPUT SWITCH	$Y_1 = X,\ Y_2 = 0 \quad P < 0$ $Y_1 = 0,\ Y_2 = X \quad P \geq 0$
$Y = COMPAR\ (X_1, X_2)$ COMPARATOR	$Y = 0 \quad X_1 < X_2$ $Y = 1 \quad X_1 \geq X_2$
$Y = RST\ (P_1, P_2, P_3)$ RST FLIP-FLOP	$Y = 0 \quad P_1 > 0$ $Y = 1 \quad P_2 > 0,\ (P_1 \leq 0)$ $Y = 0 \quad P_3 > 0,\ Y_{n-1} = 1,\ (P_2 \leq 0, P_1 \leq 0)$ $Y = 1 \quad P_3 > 0,\ Y_{n-1} = 0,$ " "

* THESE FOUR BLOCKS EXIST AS BUILT-IN MACRDS WITHIN DSL. IN-LINE CODE REPRESENTING AN EQUIVALENT INTEGRATOR CIRCUIT IS GENERATED FOR EACH USE TO PERMIT THE USE OF CENTRALIZED INTEGRATION SCHEMES WITHIN THE BLOCKS.

** INTGRL MUST BE THE RIGHTMOST TERM FOR EACH LEVEL OF USAGE. IF X IS A SINGLE VARIABLE NAME THEN IT MUST BE UNIQUE WITHIN THE PROBLEM. IC MUST ALSO BE UNIQUE. (-IC IS NOT VALID). A LITERAL MAY BE USED FOR IC. ALSO SEE SECT. 5-1.

We have decided not to use OUT = MULT (A, B), but simply OUT = A*B. Let us summarize these ideas by considering a solution to Mathieu's equation:

$$\ddot{y} + (1 + A\cos t)\,y = 0 \qquad \dot{y}(0) = 0, y(0) = Y0$$

As the DSL connection statements for this circuit follow a near one-to-one correspondence with the functional blocks in Fig. 5, they may be written as:

```
FCN    = A * COS (TIME)
MULT   = FCN * Y
Y2DOT  = -Y - MULT
YDOT   = INTGRL (0., Y2DOT)
Y      = INTGRL (Y0, YDOT)
```

(Note that TIME is a DSL system name representing the independent variable of integration. It may easily be renamed by the user.)

Observe that the DSL statements in the above example are also FORTRAN arithmetic statements, and the right-hand portions of the statements are merely FORTRAN expressions. Therefore, as such, their complexity is restricted only by the rules that govern arithmetic expressions in the FORTRAN language.

Furthermore, these expressions can serve as inputs to any functional block, regardless of whether it is a DSL/90 or user-supplied block. For example, the first three DSL structure statements in the problem above may be written as one statement,

```
Y2DOT = -Y - A * COS (TIME) * Y;
```

or perhaps as

```
Y2DOT = -Y * (1. + A * COS (TIME)).
```

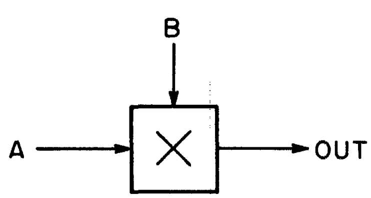


Figure 4.

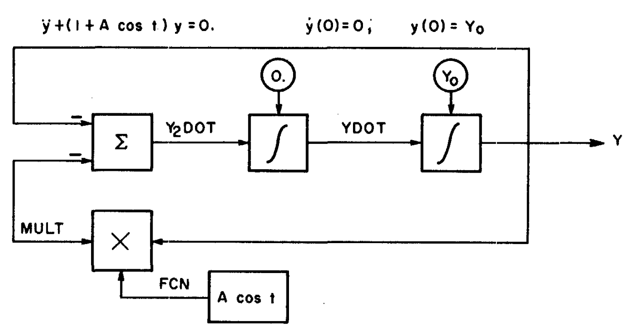
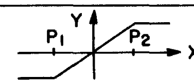
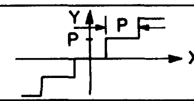
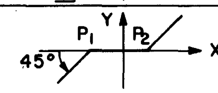
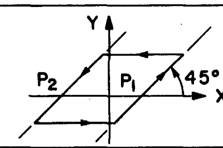
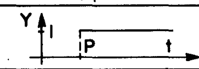
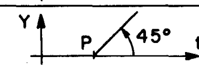
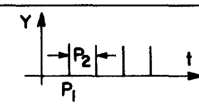
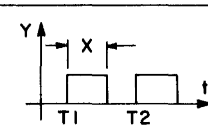
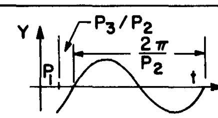
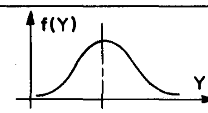
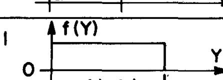
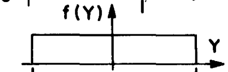
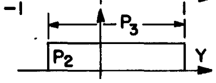


Figure 5.

FUNCTION GENERATORS

GENERAL FORM	FUNCTION	
Y = AFGEN (FUNCT, X) ARBITRARY LINEAR FUNCTION GENERATOR	Y = FUNCT (X) $X_0 \leq X \leq X_n$ LINEAR INTERPOLATION Y = FUNCT (X_0) $X < X_0$ Y = FUNCT (X_n) $X > X_n$	
Y = NLFGEN (FUNCT, X) NON-LINEAR FUNCTION GENERATOR	Y = FUNCT (X) $X_0 \leq X \leq X_n$ QUADRATIC INTERPOLATION (LA GRANGE) Y = FUNCT (X_0) $X < X_0$ Y = FUNCT (X_n) $X > X_n$	
Y = LIMIT (P_1, P_2, X) LIMITER	$Y = P_1$ $X < P_1$ $Y = P_2$ $X > P_2$ $Y = X$ $P_1 \leq X \leq P_2$	
Y = QNTZR (P, X) QUANTIZER	$Y = kP$ $(k-1/2)P < X \leq (k+1/2)P$ $k = 0, \pm 1, \pm 2, \pm 3, \ldots$	
Y = DEADSP (P_1, P_2, X) DEAD SPACE	$Y = 0$ $P_1 \leq X \leq P_2$ $Y = X - P_2$ $X > P_2$ $Y = X - P_1$ $X < P_1$	
Y = HSTRSS (IC, P_1, P_2, X) Y(0) = IC HYSTERESIS LOOP	$Y = X - P_1$ $(X - X_{n-1}) > 0$ AND $Y_{n-1} \leq (X - P_1)$ $Y = X - P_2$ $(X - X_{n-1}) < 0$ AND $Y_{n-1} \geq (X - P_2)$ OTHERWISE Y = LAST OUTPUT	
Y = STEP (P) STEP FUNCTION	$Y = 0$ $t < P$ $Y = 1$ $t \geq P$	
Y = RAMP (P) RAMP FUNCTION	$Y = 0$ $t < P$ $Y = t - P$ $t \geq P$	
Y = IMPULSE (P_1, P_2) IMPULSE GENERATOR	$Y = 0$ $t < P_1$ $Y = 1$ $(t - P_1) = kP_2$ $Y = 0$ $(t - P_1) \neq kP_2$ $k = 0, 1, 2, 3, \ldots$	
Y = PULSE (P, X) PULSE GENERATOR WITH P AS TRIGGER	$Y = 0$ INITIAL $Y = 1$ $T_k \leq t < (T_k + X)$ $Y = 0$ OTHERWISE $k = 1, 2, 3, \ldots$ T_k = t OF PULSE k, P_k	
Y = SINE (P_1, P_2, P_3) P_2 = FREQUENCY IN RADIANS/SEC. P_3 = PHASE SHIFT IN RADIANS TRIGONOMETRIC SINE WAVE WITH AMPLITUDE, PHASE, AND DELAY	$Y = 0$ $t < P_1$ $Y = \sin [P_2 \cdot (t - P_1) + P_3]$ $t \geq P_1$	
Y = NORMAL (P_1, P_2, P_3) NOISE GENERATOR (NORMAL DISTRIBUTION)	Y = GAUSSIAN DISTRIBUTION WITH MEAN, P_2, AND STANDARD DEVIATION, P_3 (P_1 = ANY ODD INTEGER)	
Y = UNZRPI (P_1) Y = UNMIPI (P_1) Y = UNATOB (P_1, P_2, P_3) NOISE GENERATOR (UNIFORM DISTRIBUTION)	Y = UNIFORM DISTRIBUTION 0 TO 1 (P_1 = ANY ODD INTEGER) Y = UNIFORM DISTRIBUTION, −1 TO +1 Y = UNIFORM DISTRIBUTION, P_2 TO $P_2 + P_3$	  

In addition, if the output, YDOT, of the first integrator is not a variable of interest, the two integrators may be "nested" as follows:

Y = INTGRL (Y0, INTGRL (0., Y2DOT)).

Finally, if the variable Y is the only one whose output is desired, the problem may be described by a single DSL connection statement, namely,

```
Y = INTGRL (Y0, INTGRL (0., - Y *
      (1. + A * COS (TIME)))).
```

The Data Statements. The subject of data entry was given prime consideration during the development of language features of DSL/90. The end result is free-form and symbolic specification of parameter values and initial conditions following a card identifier label which is punched left-adjusted in the first six columns of a data card. For example,

```
Cols 1-6      7-72
     PARAM    A = 0.5, PAR1 = 62.4,
                PAR2 = 3.215 E + 4
     INCON    ICI = 0.2, XDOT = 1.3
     CONST    C1C = 7.3, C2C = 100.,
                T = 46.25,
              EPSILN = 1.0 - 05
```

The identifying labels begin in column one. The data items, separated by commas, may be placed anywhere in columns 7–72. Blanks are ignored. Three consecutive decimal points at the end of any statement indicate that it is to be continued on the next card. Continuation may begin anywhere in columns 1–72. Data statements may be intermingled with connection statements.

The Control Statements. The statements may be conveniently grouped into three types:

1. Problem output control statements include print and plot requirements, title information and labeling of graphs, such as:

```
PRINT    .01, Y, Y2DOT
PREPAR   .005, Y, Y2DOT
GRAPH    8., 6., TIME, Y, Y2DOT
LABEL    SOLUTION OF MATHIEU'S
           EQUATION
RANGE    DELT, X
```

The above cards will cause the printing of TIME, Y, and Y2DOT at intervals of 0.01 units of time, and preparation of TIME, Y, and Y2DOT for graphing at intervals of 0.005 units of time. A single 8 × 6-inch graph properly labeled as directed, will be made with Y and Y2DOT plotted vs TIME. The maximum and minimum values attained by DELT and X will be printed at the end of the run.

2. Problem execution control statements are used to set error bounds and step size for integration routines, prescribe run cutoff conditions, and to specify other pertinent run information. Typical examples are

```
CONTRL DELT = .05, FINTIM = 2.0
ABSERR YDOT = 1.0 E - 5, Y = 5.0 E - 4.
```

The simulation will be executed from 0 to 2.0 with an integration interval of 0.05. The error bounds on YDOT and Y will be held at 1.0×10^{-5} and 5.0×10^{-4}, respectively. The latter bound will be applied to all other unspecified integrator outputs.

3. System control statements provide the user with a number of options, the most important ones being choice of integration methods, bypassing the sequencing routine, and renaming of system variables. They also include an END card which signifies the end of a logical set of data cards, and a STOP card which ends the computer run.

For example:

```
CONTIN
INTEG     MILNE
NOSORT
RENAME    TIME = X, DELT = DELX
FINISH    DIST = 0.
```

These cards cause continuation of the simulation from the last calculated point, selection of the Milne 5th-order integration scheme, exercise of the no-sort option, renaming of two systems variables, and termination of the run when the value of DIST reaches zero.

All data and control cards, with the exception of the END and STOP cards and certain logical groups of cards (such as continuation statements) may be intermixed with DSL structure statements and may appear in any order. Proper statement order is determined by an internal sort based on correct information flow. Table 2 shows a complete list of DSL/90 data and control statements. Returning to Mathieu's equation, a complete DSL/90 program for $\ddot{y} + (1 + A \cos t)\, y = 0$ may be written as follows:

```
1-6       7-72
TITLE     SOLUTION OF MATHIEU'S
            EQUATION
          Y2DOT = -Y*(1.0 + A * COS
          (TIME))
PARAM     A = 0.5
          Y = INTGRL (Y0, INTGRL (0.,
          Y2DOT))
INCON     Y0 = 20.0
INTEG     MILNE
```

TABLE 2 Summary of DSL/90 Data Statement Formats

	Label	Function (By Example)
COL.	1-6	7-72
PROBLEM DATA INPUT:		
	PARAM	TAU = 25., PAR = 3.158E3, C4 = 2.0 E-5
	CONST	CON1 = 45.3, PI = 3.14159, K = 3
	INCON	IC1 = 20., A = 50.2, IC3 = 0
	AFGEN	FCN = 3., 25., 5.2, 26.4, 6.0, 24., 7.5, 21.3
	NLFGEN	FY3 = 0., 850., 5., 1245., 8., 1.574E3, 12.4, 2.4E03
	TABLE	PAR1(8) = 4.5, INPUT(1-4) = 2., 2*8.6, 3.52E3
PROBLEM OUTPUT CONTROL:		
	PRINT	0.1, X, XDOT, VELOC
	TITLE	MASS, SPRING, DAMPER SYSTEM IN DSL/90
	PREPAR	.05, X, Y, XDOT
	GRAPH	10., 8., TIME, X, XDOT
	LABEL	MASS, SPRING, DAMPER SYSTEM - 6/1/65
	RANGE	X, XDOT, VELOC, DELT
PROBLEM EXECUTION CONTROL:		
	CONTRL	DELT = .002, FINTIM = 8.0, DELMIN = 1.0E-10
	FINISH	DIST = 0., ALT = 5000.
	RELERR	X = 1.E-4, XDOT = 5.E-5
	ABSERR	X = 1.E-3, XDOT = 1.E-4
	CONTIN	
	INTEG	MILNE
	RESET	GRAPH, PRINT
DSL/90 TRANSLATOR PSEUDO-OPERATIONS:		
	RENAME	TIME = DISPL, DELT = DELTX
	INTGER	K, GO
	MEMORY	INT(4), DELAY (100)
	STORAG	IC(6), PARAM (10)
	DECK	
	SORT	
	NOSORT	
	PROCED	X = FCN (A, B, PAR5, IC3)
	⋮	
	ENDPRO	
	MACRO	OUT = FCN2 (IC1, R, T, X)
	⋮	
	ENDMAC	
	END	
	STOP	

```
CONTRL  DELT = .02, FINTIM = 2.0
ABSERR  Y2DOT = 1.0E-5, Y = 2.0 E-5
PRINT   0.05, Y, Y2DOT
END
STOP
```

It should be apparent by now that the DSL input language is block-oriented, symbolic, and free-form. The use of FORTRAN is not limited to arithmetic statements. All FORTRAN library functions such as SQRT, SIN, COS, etc., are available. Under the rules which are clearly defined within DSL/90, a large subset of FORTRAN becomes available to the simulation user without sacrificing the ease of block notation programming. What this means to the engineer who is unskilled in FORTRAN programming is simply this: he can still perform his process simulation with a simple language, following a step-by-step building block approach. As he becomes more proficient, his programming becomes correspondingly more efficient and he may want to include elementary FORTRAN language features in his connection statements. Still later, as the complexity of his problem increases, he may use to advantage the more powerful features of DSL and FORTRAN.

Advanced Language Features

There are a number of other DSL/90 language features which are especially useful for the simulation of large or complex problems. We shall examine several of these.

Procedural Statements. Recall that the order in which DSL statements are entered is unimportant because connection statements are separated from the rest and sequenced (or "sorted") by the DSL processor (unless a "no-sort" option is exercised). In other words, the DSL/90 language may be considered as nonprocedural. In contrast, FORTRAN is a procedural language since FORTRAN statements are executed in the order in which they are written. Frequently, in a complex process simulation, it is desirable to introduce procedural statements within the simulation program. The purpose may be to control signal flow in certain portions of the program, or perhaps to compute a large number of parameter values once and only once. DSL/90 uses a pair of pseudo-operations, PROCED and ENDPRO, punched in columns 1–6, to designate the beginning and end of a block of procedural statements (they may be DSL or FORTRAN statements). Input and output names may be specified on the PROCED card to allow the procedural statements to be sorted as a block relative to other DSL statements. For example:

```
PROCED  TEMP = BLOCKA (TEST, IN)
        IF (TEST) 10, 10, 20
   10   TEMP = LIMIT (PAR1, PAR2, IN)
        GO TO 30
   20   TEMP = IN + TEST
   30   CONTINUE
ENDPRO
```

During the sequencing of DSL statements, the above procedural statements will be treated as a single functional block with output TEMP and inputs TEST and IN, as illustrated in Fig. 6. The order of the statements within the procedural block remains unchanged.

Macro-Generation. Pseudo-operations MACRO and ENDMAC, which are punched in columns

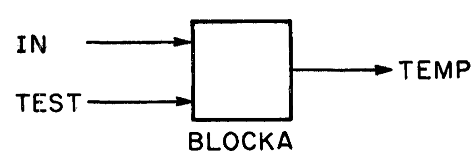


Figure 6.

1–6, are used to define a macro block. One may think of a macro as a repeatable procedural block with parameter variations. This is best illustrated by example. The following statements constitute a macro-definition:

```
1–6       7–72
MACRO     OUT = FILTER (V1, V2, K, IN)
          V1 = (IN - V2)/K
          V2 = INTGRL (0., V1)
          OUT = V2 + 0.5*V1
ENDMAC
```

During the definition of the macro, no language statements are produced. The name of this macro, FILTER, must be unique. However, the output name OUT and the input names, V1, V2, K, and IN, are dummy symbols which will be replaced by the actual names specified at the time when the macro is used. The subsequent appearance of the statement

```
LINE 1 = FILTER (A1, A2, TAU, XIN)
```

will cause the following three statements to be generated in-line:

```
A1      = (XIN - A2)/TAU
A2      = INTGRL (0., A1)
LINE1   = A2 + 0.5*A1
```

Just as in the case of the procedural block, these statements will be sequenced as a single functional block with LINE1 as output and A1, A2, TAU and XIN as inputs (see Fig. 7). The statements within the block are not sorted. Both DSL and FORTRAN statements may appear within a macro.

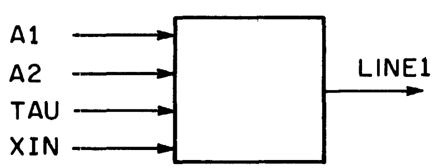


Figure 7.

Implicit Function Block. DSL/90 provides an implicit function block called IMPL for the solution of an implicit equation $f(y) = 0$ expressed in the form of $y = f(y)$. Clearly some iterative technique must be employed. These iterations must be performed within each integration interval until a convergence criterion is satisfied. The program for IMPL uses the direct iteration method developed by Wegstein. If there is no convergence after some preassigned maximum number of iterations, the simulation of the problem is terminated with appropriate diagnostic printout.

To use the implicit function block, one writes the DSL statement,

```
Y = IMPL (YO, ERROR, FOFY)
```

followed by the set of DSL or FORTRAN (or both) statements evaluating FOFY. Y, YO, ERROR and FOFY are symbolic names selected by the user. The DSL/90 system then sets up the necessary iterative loop. Let us illustrate by solving the implicit equation

$$y = \frac{C \cdot (e^y - 1)}{e^y} \qquad \text{(C is some constant)}$$

One simply writes:

```
Y     = IMPL (YO, ERROR, FOFY)
A     = EXP(Y)
FOFY  = C* (A - 1.0) / A
```

The DSL/90 translator will automatically generate the following statements:

```
30001Y = IMPL (YO, ERROR, FOFY)
      IF (NALARM .LE.0) GO TO 30002
      A     = EXP (Y)
      FOFY  = C* (A - 1.0) / A
      GO TO 30001
30002 CONTINUE
```

Note that three statements, and only those three, are added to the ones written by the user. The first time the IMPL routine is entered, NALARM is set to one, and Y is given the initial guess Y0. After each calculation of $f(y)$, program flow returns to the IMPL subroutine where the convergence criterion is tested. If satisfied, NALARM is set equal to zero and y assumes the most recently calculated value of $f(y)$. Otherwise the iteration continues.

User-Supplied Functional Blocks. Although DSL/90 provides an extensive library of operational blocks, there are occasions when special blocks are required to simulate specific process elements. These special blocks are programmed by the user as subroutines either in FORTRAN or MAP and simply added to the data at the time the simulation run is made. The user may treat these special blocks like all other DSL library blocks, interconnecting them to build a complex system model.

As an example of the use of special blocks, consider the modeling of the analog-to-digital converter shown as a nonlinear stepwise quantization in Fig. 8. If no such general block existed in the DSL library, it would be difficult to construct such a characteristic from the standard blocks available. How-

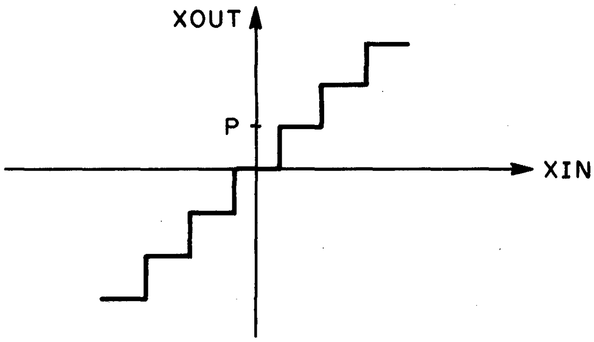


Figure 8.

ever, the quantization effect is easily modeled by the following FORTRAN statements:

```
FUNCTION QNTZR (P, XIN)
QNT = AINT (0.5 + ABS (XIN)/P)
QNTZR = SIGN (P*QNT, XIN)
RETURN
END
```

The parameter named P containing the value of the quanta step size is the only parameter supplied to the QNTZR block. This value of P is entered into the simulation program in exactly the same way as any other DSL parameter—on a PARAM card. Note also that the two blocks AINT (for truncation) and SIGN (for transfer of sign) are standard subroutines of the FORTRAN library. The above FORTRAN subprogram for the quantizer may be entered directly with the data cards for the simulation run, or as an alternative, it may be compiled independently and the resulting machine language deck (binary deck) added to the data deck. This functional block may even be added to the permanent DSL library by simply loading it on the library tape. In fact this was the case with the QNTZR block when we found it to be sufficiently useful to warrant a place in the DSL library. The ease with which a difficult nonlinearity has been modeled in a few lines of FORTRAN coding is quite apparent and typifies the flexibility of DSL/90 for handling nonlinear functions and special blocks.

Arbitrary Functions. DSL/90 provides two functional blocks, AFGEN and NLFGEN, for handling arbitrary functions of one variable. The x, y coordinates of the function points are entered sequentially following an identifying label and the symbolic name of the function, e.g.:

```
1–6     7–72
AFGEN   FC1 = –10.2, 2.3, –5.6, 6.4, 1.0, 5.9, etc.
```

Although the total number of data storage locations is necessarily fixed by machine size, there is no restriction on the number of points one may use to define any function. The only requirement is that the x coordinates in the sequence $x_1, y_1, x_2, y_2, \ldots$ are monotonically increasing. Any number of arbitrary functions may be defined, identified only by their symbolic names assigned by the user. As an example, the DSL statement Y3 = AFGEN (FC1, XIN) will refer to the function called FC1. AFGEN provides linear interpolation between consecutive points, while NLFGEN uses a second-order Lagrange interpolation formula.

Tabular Data. This feature of DSL/90 allows blocks of data to be transmitted to the UPDATE subroutine in tabular form. In the construction of a special block, the user may have to consider sets of initial conditions, history and input parameters. This DSL/90 feature will eliminate the need for a lengthy subroutine argument string. To illustrate, suppose we wish to build a special block called SPEC which requires two initial conditions and 10 parameters. We begin by writing the following two DSL statements:

```
1–6      7–72
STORAG   IC(2), PAR(10)
TABLE    IC(1) = 2.0, IC(2) = 0.0, PAR(1)
           = 4., PAR(2–10) = 9*1.5
```

The first statement instructs the DSL/90 system to assign a total of 12 locations—2 for the array IC and 10 for PAR. The second statement illustrates the manner in which numeric values are entered into these reserved locations. Now, when we subsequently use a statement such as

```
YOUT = SPEC (IC, PAR, XINPUT)
```

DSL/90 system will replace the names IC and PAR with the addresses of the first locations of the arrays IC and PAR respectively. Obviously, the user when programming his subroutine SPEC must realize that the first two arguments in SPEC are location pointers to his arrays. His subroutine could begin with the following:

```
FUNCTION SPEC (LOCIC, LOCPAR, XIN)
COMMON/CURVAL/C(1)
I = LOCIC
J = LOCPAR
```

CURVAL is the labeled common where the current values of all variables are stored, and I and J are indices referencing the first initial conditions IC and parameter values PAR.

System Features

DSL/90 System Organization. The DSL/90 Operating System is separated into two major functions: language translation and model simulation. Each function operates independently under standard IBSYS control but as one continuous single-pass operating system. The transition is made by having the translator develop on an IBSYS scratch tape all the elements of a standard IBSYS job as well as the representation of the model to be simulated. This tape is then switched in as the standard IBSYS input for compilation and execution to complete the simulation. Diagnostics are printed if errors are found in translation or simulation. Elements which may appear as input to the translator are: 1) DSL/90 problem-oriented language sentences to describe the model, 2) data input to the model for parameter values and control of the simulation and output, 3) binary and BCD subroutines and functions supplied by the user for the simulation, and 4) appropriate controls to load binary or BCD subroutines and functions from a library tape. The entire system may be placed at any level of a standard batched IBSYS run. Three additional tape drives are required—two auxiliary and one for plotting.

DSL/90 may be run as an independent program or it may be used as a subprogram of a conventional FORTRAN program for control purposes.

Sort. A nonprocedural input language such as DSL/90 transfers the responsibility of establishing the execution sequence from the user to the program. To accomplish this DSL/90 alters the sequence of input statements according to the rule: an operational element (or statement) is properly sequenced if all its inputs are available either as input parameters or initial conditions or as previously computed values in the current iteration cycle. Unspecified algebraic loops are identified and, if any, the run is halted. The result of this sequencing operation is a properly organized FORTRAN IV subprogram.

Main Program Control. DSL/90 provides for calling the simulation routines from a MAIN program specified by the user. Hence the actual digital simulation may be placed under control of a FORTRAN routine compiled at execution time. This feature allows for testing of response conditions, matching boundary values, and dynamic alteration of parameters, initial conditions, or run control data between parameter studies.

Centralized Integration. By use of the block name, INTGRL, a user may specify that centralized integration is desired. The translator sets up statements so as to compute all inputs to the integrators but bypass computation of outputs until the end of the iteration cycle. At this time, all integrator outputs are updated simultaneously. A choice can be made between the 5th-order Milne Predictor-Corrector, 4th-order Runge-Kutta, Simpson's Trapezoidal, or Rectangular Integration methods. The first three allow the integration interval to be adjusted by the system to meet a specified error criterion, a factor which allows it to take large or small steps depending on the rate of change of one or more variables. There is provision in DSL/90 for the user to supply his own integration scheme, which may or may not be centralized.

Dynamic Storage Allocation. Data in DSL/90 is stored in a single vector including current values of structure variables and table values for function generators, integration history, error bounds, STORAG variables, etc. The storage is allocated dynamically (i.e., at execution time) according to what portions of the simulator are used and how many integrators, tables, and structure variables are in the simulation model. Standard DSL/90 blocks are loaded only if used.

APPLICATIONS

Having illustrated operational features of the DSL/90 digital simulation program, we will now draw upon the previous introduction to show how DSL/90 has been flexibly applied to simulation problems. Three specific simulations will be considered: 1) a biomedical block notation problem involving a respiratory servomechanism; 2) a process analysis problem involving the simulation of heat transfer dynamics of a recirculating furnace used in the glass industry; and 3) the simulation of the flight dynamics of a portion of the SATURN V booster rocket.

DSL/90 provides special programming features such as different integration methods, sorting, special blocks, etc., which make it attractive to the user for continuous system simulation. Several of these features will be illustrated in the examples to follow.

Application No. 1—Respiratory Servo Simulation

This problem involves evaluating the response of a proposed model for respiratory control of CO_2

partial pressure in the venous and arterial blood streams of a human. De Fares et al performed the original study on an analog computer and represented the basic CO_2 control mechanism in respiration by the three-compartment model shown in Fig. 9. Using the original study as a guide, this first example will illustrate the ease of handling conventional analog simulation problems using DSL/90.

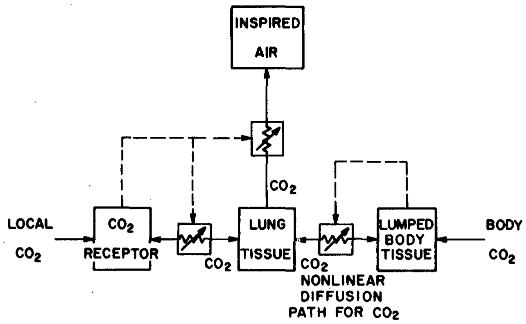


Figure 9. CO_2 control model.

The CO_2 control system operates as follows: The alveolar tissue in the lung serves as an exit sink for CO_2 production and possesses both CO_2 capacity and conductance characteristics. In a similar manner, body tissue can be considered as having an equivalent CO_2 capacitance and conductance. CO_2 produced by the body is partially stored in the local body tissue, raising the local body tissue partial pressure of CO_2. The CO_2 produced is simultaneously diffused through the tissue and picked up by the blood stream (venous path). The CO_2 is then carried to the lung and subsequently diffused to the alveolar tissue, raising its CO_2 partial pressure. Simultaneously, CO_2 is produced in the region of a receptor (CO_2 detector) in the medulla. This CO_2 is similarly diffused and carried to the alveolar tissue through the venous blood stream. It can be shown that the basic controlled variable in this system model is the partial pressure of CO_2 in the receptor tissue located in the medulla.

If CO_2-enriched air is also brought into the lungs, it simultaneously affects the CO_2 diffusion and buildup in the alveolar lung tissue. De Fares et al have shown that the partial pressure of CO_2 in the receptor can serve as an effective mechanism for controlling diffusion of CO_2 from the receptor and from inspired air. In this study, the CO_2 partial pressures of mixed venous blood flow and body tissue will be assumed equal. Similarly, the CO_2 partial pressures of arterial blood flow and alveolar lung tissue will be assumed equal.

By introducing disturbances in the CO_2 content of inspired air, the dynamics of such a control model may be studied. The objective of this model is to hold constant the partial pressure of the CO_2 in the receptor by controlling the diffusion conductance of CO_2 from the receptor area and of the inspired gas to the alveolar lung tissue. Thus, the CO_2 partial pressures of alveolar tissue and local body tissue will respond dynamically to changes in CO_2 content of the inspired air.

Network Model. Because of the dynamic analogies existing between the gas dynamics of the CO_2 diffusion model above and conventional circuit dynamics, it is convenient to represent the biological model by an equivalent circuit model. Figure 10 shows three capacitors tied together with variable nonlinear conductances, which represent the diffusion characteristics of the separate tissue/blood interface. The capacitors represent local tissue CO_2

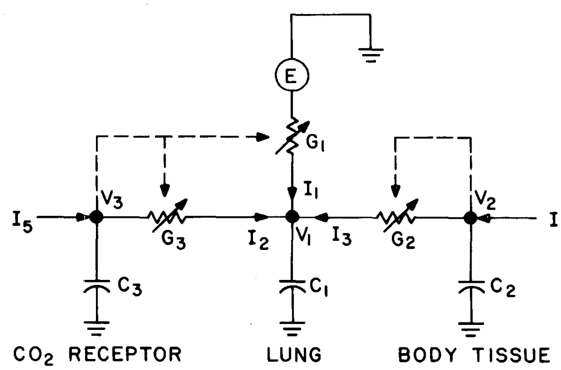


Figure 10. Equivalent network model.

capacity, and the voltages become the respective CO_2 partial pressures. The voltage source E represents the partial pressure of CO_2-enriched inspired air and is defined by the following relation:

$$E = Fi\,(B\text{-}47)$$
$$Fi = \%\ CO_2 \text{ content in inspired air}$$

where B = atmospheric pressure in mm Hg.

Table 3 lists the electrical network parameters and variables together with their physiological equivalents.

Digital-Analog Simulation. As a first example of DSL/90 application flexibility, conventional analog

Table 3. Electrical and Physiological Equivalents, Application No. 1

Elec.	Physiological	
Symbol	Quantity	*Units
G_1	CO_2 conductance-air to lung tissue	Liters (gas)/min/mm Hg (gas)
G_2	CO_2 conductance-body tissue to lung	Liters (CO_2)/min/mm Hg (CO_2)
G_3	CO_2 conductance-receptor to lung	Liters (CO_2)/min/mm Hg (CO_2)
C_1	Capacity of lung tissue	Liters (gas)/mm Hg (gas)
C_2	Capacity of body tissue	Liters (CO_2)/mm Hg (CO_2)
C_3	Capacity of receptor tissue	Liters (CO_2)/mm Hg (CO_2)
V_1	CO_2 partial pressure of lung tissue	mm Hg (CO_2)
V_2	CO_2 partial pressure of body tissue	mm Hg (CO_2)
V_3	CO_2 partial pressure of receptor tissue	mm Hg (CO_2)
E	Partial pressure of CO_2 in inspired air	mm Hg (CO_2)
I_4	Body CO_2 production	Liters (CO_2)/min
I_5	Receptor CO_2 production	Liters (CO_2)/min
I_1	CO_2 diffusion from inspired air to lung tissue	Liters (gas)/min
I_2	CO_2 diffusion from body tissue to lung tissue	Liters (CO_2)/min
I_3	CO_2 diffusion from receptor tissue to lung tissue	Liters (CO_2)/min

*Units are liters BTPS, m.m. Hg, minutes

block notation will be used to program the simulation. Figure 11 represents a DSL/90 digital-analog simulation block diagram of the network model shown in Fig. 10. Since DSL/90 operations are in floating-point arithmetic, no problem scaling is required and the parameters may be entered directly in terms of their conductances are given by the following relations:

$$G_1 = \psi_1 * V_3 - \Theta_1$$
$$G_2 = \psi_2 * V_2 - \Theta_2$$
$$G_3 = \psi_3 * V_3 - \Theta_3$$

where ψ is proportional to the slope of the experimentally determined steady-state cardiac output versus CO_2 partial pressure curves—liters (CO_2)/min/mm^{2}Hg (CO_2); and $\Theta \doteq$ initial value of G, liters (CO_2)/min/mm Hg (CO_2).

Using data from respiratory experiments, the following parameters and initial values hold for the simulation:

$V_1(0)$	$= 40.0$	C_1	$= 0.00344$
$V_2(0)$	$= 45.0$	C_2	$= 0.17$
$V_3(0)$	$= 45.0$	C_3	$= 0.0008$
ψ_1	$= 0.0038$	Θ_1	$= 0.1648$
ψ_2	$= 0.0025$	Θ_2	$= 0.0625$
ψ_3	$= 0.0002$	Θ_3	$= 0.0007$
I_4	$= 0.25$	I_5	$= 0.001$

The DSL/90 statements which describe this simulator follow.

```
TITLE RESPIRATION SERVO PROBLEM - ANALOG MODE SOLUTION  6-1-65 RUN 1

      EIN=E*(1.-STEP(TDELAY))
      ADR2=EIN-V1
      G1=PSI1*V3-THETA1
      I1=G1*ADR2
      V1=INTGRL(V1IC,(I1+I2+I3)/C1)
      ADR4=V2-V1                                  Connection
      G2=PSI2*V2-THETA2                           Statements
      I2=G2*ADR4
      V2=INTGRL(V2IC,(I4-I2)/C2)
      ADR7=V3-V1
      G3=PSI3*V3-THETA3
      I3=G3*ADR7
      V3=INTGRL(V3IC,(I5-I3)/C3)

PARAM C1=0.00344, C2=0.17, C3=0.0008,...
      PSI1=0.0038, PSI2=0.0025, PSI3=0.00002,...         Parameters
      THETA1=0.1648, THETA2=0.0625, THETA3= 0.0007, E.21.4  and IC's
CONST I4=0.25, I5=0.001, TDELAY=20.0
INCON V1IC=40.0, V2IC=45.0, V3IC=45.0

CONTRL  FINTIM=36.0, DELT=0.05                  Run
RELERR  V1=0.001                                Control
INTEG MILNE

PRINT   0.1, V1, V2, V3, G1, G2, G3, I1, I2, I3
PREPAR 0.05, V1, V2, V3, G1, G2, G3, I1, I2, I3
GRAPH 6.0, 4.0, TIME, V1, V2, V3                   Print and
LABEL PAR PRESS  3.0 PRCNT CO2  RUN 1  6-1-65      Plot Output
GRAPH 6.0, 4.0, TIME, G1, G2, G3
LABEL CONDUCTANCE  3.0 PRCNT CO2  RUN 1  6-1-65
GRAPH 6.0, 4.0, TIME, I1, I2, I3
LABEL CO2 DIFFUSION  3.0 PRCNT CO2  RUN 1  6-1-65

END
STOP
```

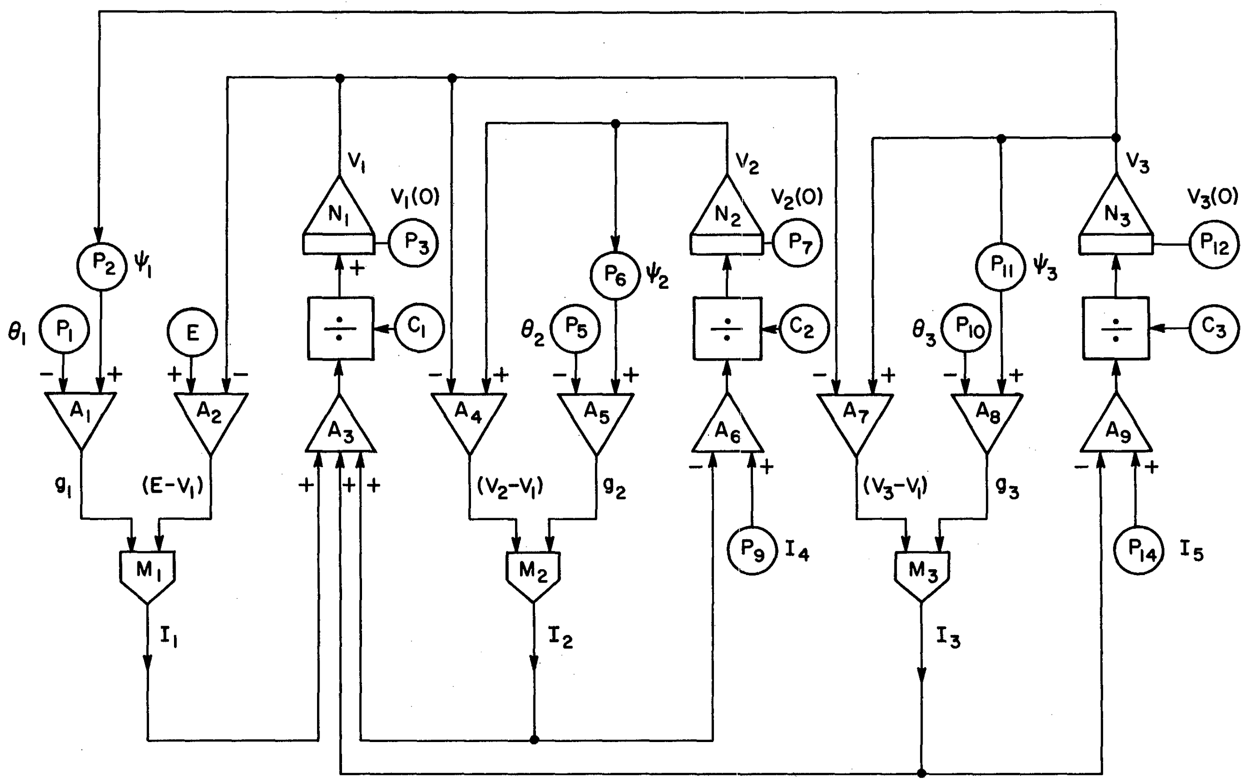


Figure 11. Digital-analog simulator block diagram.

Figures 12 and 13 show nonretouched DSL/90 plots of CO_2 partial pressures and tissue conductances. Inhaled air containing 3% CO_2 was assumed for 20 minutes followed by a 20-minute span of normal room air with no CO_2 content.

During the first 20 minutes, the receptor tissue (medulla), body tissue, and aveolar lung tissue all take up CO_2. The second 20-minute span shows the nonlinear response during purging of body CO_2.

Figure 14 shows part of the results printout and input data format.

After the initial runs were completed, a change in the G_3 conductance characteristic was suggested by medical research personnel. Instead of a linear relationship between G_3 and receptor CO_2 partial pressure, a smoothwise increasing empirical function as shown in Fig. 15 was substituted. To do

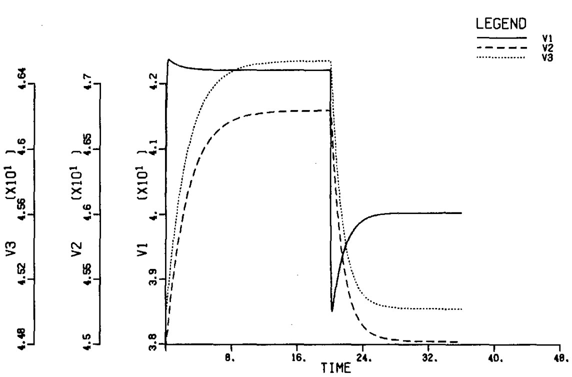


Figure 12. Par press 3.0% CO_2 run 1, 6-1-65.

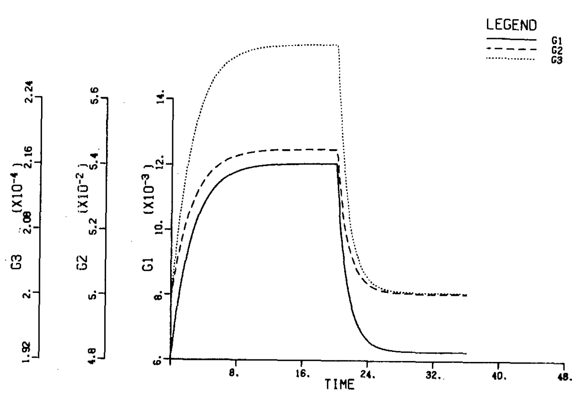


Figure 13. Conductance 3.0% CO_2 run 1, 6-1-65.

```
*** DSL/90 SIMULATION DATA ***
TITLE RESPIRATORY SERVO PROBLEM - NETWORK MODE SOLUTION  6-1-65 RUN 4
PARAM C1=0.00344, C2=0.17, C3=0.0008....
      PSI1=0.0038, PSI2=0.0025, PSI3=0.00002....
      THETA1=0.1648, THETA2=0.0625, THETA3= 0.0007, E=21.4
CONST I4=0.25, I5=0.001, TDELAY=20.0
INCON V1IC=40.0, V2IC=45.0, V3IC=45.0
CONTRL  FINTIM=36.0, DELT=0.05
RELERR  V1=0.001
INTEG MILNE
PRINT   0.1, V1, V2, V3, G1, G2, G3
PREPAR 0.05, V1, V2, V3, G1, G2, G3
GRAPH 6.0, 4.0, TIME, V1, V2, V3
LABEL PAR PRESS  3.0 PRCNT CO2  RUN 4  6-1-65
GRAPH 6.0, 4.0, TIME, G1, G2, G3
LABEL CONDUCTANCE  3.0 PRCNT CO2  RUN 4  6-1-65
END
```

Figure 14a. DSL/90 simulation data.

this, it was necessary to redefine the G_3 conductance characteristic as the output of an arbitrary function generator block as follows:

$$G3 = \text{AFGEN (F3, V3)}$$

where the G_3 characteristic is given in a sequence of X and F(X) values.

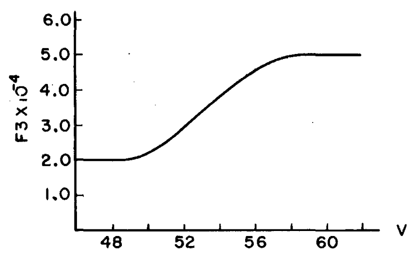
AFGEN F3 = 0.0, .0002, 48., .0002, 49., .00021, ... 50., .00023, 51., .00027, 52., .00031, 53., .00035, ... 54., .00039, 55., .00043, 56., .000465, 57., .00048, ... 58., .00049, .59., .000495, 60., .0005, 80., .0005

In addition to the analog model approach shown here, two other methods were programmed in DSL/90 involving the network equations directly and

TIME	V1	V2	V3	G1	G2	G3
0.	4.0000E 01	4.5000E 01	4.5000E 01	6.2000E-03	5.0000E-02	2.0000E-04
10.000E-02	4.1937E 01	4.5036E 01	4.5030E 01	6.3147E-03	5.0089E-02	2.0060E-04
2.000E-01	4.2319E 01	4.5097E 01	4.5082E 01	6.5132E-03	5.0242E-02	2.0165E-04
3.000E-01	4.2369E 01	4.5162E 01	4.5138E 01	6.7232E-03	5.0404E-02	2.0275E-04
4.000E-01	4.2366E 01	4.5225E 01	4.5192E 01	6.9284E-03	5.0562E-02	2.0383E-04
5.000E-01	4.2359E 01	4.5286E 01	4.5244E 01	7.1262E-03	5.0714E-02	2.0487E-04
6.000E-01	4.2350E 01	4.5344E 01	4.5294E 01	7.3168E-03	5.0861E-02	2.0588E-04
7.000E-01	4.2340E 01	4.5401E 01	4.5342E 01	7.5003E-03	5.1002E-02	2.0684E-04
8.000E-01	4.2331E 01	4.5455E 01	4.5389E 01	7.6769E-03	5.1137E-02	2.0777E-04
9.000E-01	4.2323E 01	4.5507E 01	4.5433E 01	7.8468E-03	5.1267E-02	2.0867E-04
10.000E-01	4.2315E 01	4.5557E 01	4.5476E 01	8.0103E-03	5.1392E-02	2.0953E-04
1.100E 00	4.2308E 01	4.5605E 01	4.5518E 01	8.1676E-03	5.1513E-02	2.1036E-04
1.200E 00	4.2301E 01	4.5651E 01	4.5558E 01	8.3189E-03	5.1628E-02	2.1115E-04
1.300E 00	4.2295E 01	4.5696E 01	4.5596E 01	8.4644E-03	5.1739E-02	2.1192E-04
1.400E 00	4.2289E 01	4.5738E 01	4.5633E 01	8.6043E-03	5.1846E-02	2.1265E-04
1.500E 00	4.2284E 01	4.5779E 01	4.5668E 01	8.7389E-03	5.1949E-02	2.1336E-04
1.600E 00	4.2279E 01	4.5819E 01	4.5702E 01	8.8684E-03	5.2047E-02	2.1404E-04
1.700E 00	4.2274E 01	4.5857E 01	4.5735E 01	8.9928E-03	5.2142E-02	2.1470E-04
1.800E 00	4.2270E 01	4.5893E 01	4.5766E 01	9.1125E-03	5.2233E-02	2.1533E-04
1.900E 00	4.2265E 01	4.5928E 01	4.5797E 01	9.2276E-03	5.2321E-02	2.1593E-04
2.000E 00	4.2262E 01	4.5962E 01	4.5826E 01	9.3382E-03	5.2405E-02	2.1652E-04
[illegible]	4.2258E 01	4.5994E 01	4.5[illegible]	[illegible]5E-03	5.2486E-02	2.1708E-0[illegible]
[illegible]	[illegible]5E 01	4.6025E 01			5.2563E-02	2.[illegible]
			[illegible]5019E 01			
[illegible]		[illegible] 01	4.5019E 01	6.2[illegible]		[illegible]038E-04
3.500E 01	[illegible]	4.5023E 01	4.5019E 01	6.2715E-0[illegible]	[illegible]02	2.0038E-04
3.510E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2715E-03	5.0057E-02	2.0038E-04
3.520E 01	4.0028E 01	4.5023E 01	4.5019E 01	6.2714E-03	5.0057E-02	2.0038E-04
3.530E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2714E-03	5.0057E-02	2.0038E-04
3.540E 01	4.0028E 01	4.5023E 01	4.5019E 01	6.2714E-03	5.0057E-02	2.0038E-04
3.550E 01	4.0028E 01	4.5023E 01	4.5019E 01	6.2713E-03	5.0057E-02	2.0038E-04
3.560E 01	4.0028E 01	4.5023E 01	4.5019E 01	6.2713E-03	5.0057E-02	2.0038E-04
3.570E 01	4.0028E 01	4.5023E 01	4.5019E 01	6.2713E-03	5.0057E-02	2.0038E-04
3.580E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2713E-03	5.0057E-02	2.0038E-04
3.590E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2712E-03	5.0057E-02	2.0037E-04
3.600E 01	4.0023E 01	4.5023E 01	4.5019E 01	6.2712E-03	5.0057E-02	2.0037E-04

DSL/90 SIMULATION TIME =   13.892 SECONDS

Figure 14b. Respiratory servo problem—network mode solution.



V	F3
0.0	.0002
48	.0002
49	.00021
50	.00023
51	.00027
52	.00031
53	.00035
54	.00039
55	.00043
56	.000465
57	.00048
58	.00049
59	.000495
60	.0005
80	.0005

Figure 15. G_3 conductance characteristic.

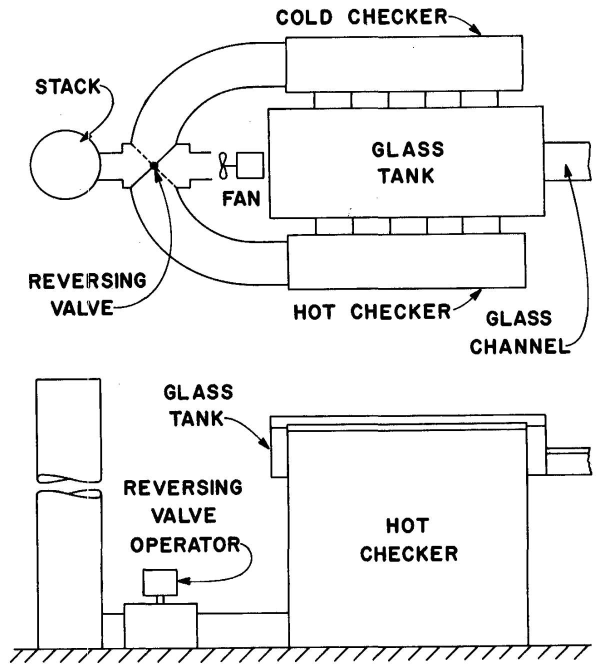


Figure 16. Schematic diagram—reversing furnace.

fundamental compartment models. This last approach has proven particularly attractive since the biomedical user can directly program his own simulation problem without learning an artifax tool such as analog computer notation, network analysis, or FORTRAN programming. These techniques result in a major reduction in the user time required from initial problem coding to achieving final results. In addition, complete printouts and digital plots are available for each problem run, considerably simplifying the simulation documentation problem.

Application No. 2—Glass Tank Recirculating Furnace

This second example involves the analysis of the heat transfer dynamics of a recirculating furnace used for preheating combustion air on a glass tank. The problem illustrates the ease of using generalized block notation in DSL/90 for performing continuous system simulations. In this case, the example was drawn from the industrial process control field. The technique, however, is broadly applicable to any continuous system analysis problem.

As shown in Fig. 16, air is forced through a large preheating chamber, called a checker, filled with bricks cross-stacked to allow passage of the air around the brick surface, thereby preheating the cold air from the brick. The preheated air is then mixed with fuel, fired, and the resultant flame front melts the glass material in the tank. The hot combustion gases are forced through another checker, heating up the cold brick, and finally forced out the stack. After a period of time, usually about 15 minutes, the flow direction valve is reversed so that the cold checker that had been heated by the hot gases now becomes the preheating checker for the cold incoming air. Similarly, the previous hot checker that had been cooled by the cold input air now receives hot combustion gases which heat it up again. The object of the simulation is to study the heat transfer dynamics of the recirculation furnace during the heating and cooling cycles induced by air flow reversals.

The first step was to divide each checker chamber into three blocks, as shown in Fig. 17, effectively breaking a continuously distributed system into a

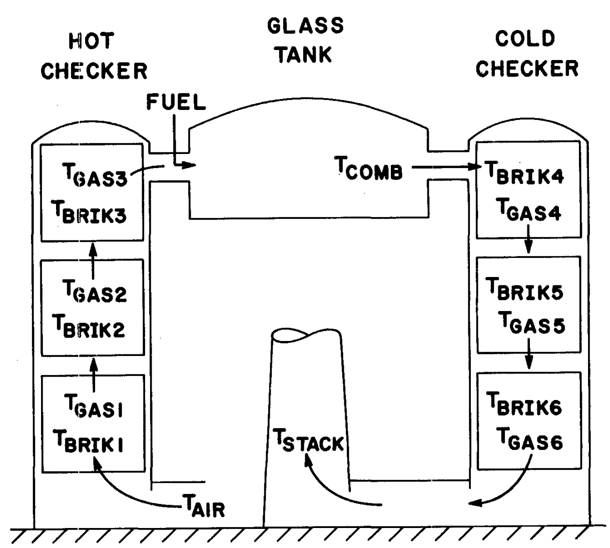


Figure 17. Reversing furnace—end view.

sequence of lumped-parameter segments. The nonlinear heat transfer relationships for each block are given by Eqs. 1 and 2.

$$\frac{d}{dt}\rho_2 V \sigma_A T_{GAS} = \sigma_A F_1 T_{IN} - \sigma_A F_1 T_{GAS} + hA_1(T_{BRICK} - T_{GAS}) + K[(T_{BRICK} + 460)^4 - (T_{GAS} + 460)^4] \quad (1)$$

$$\frac{d}{dt} M \sigma_B T_{BRICK} = hA_2(T_{BRICK} - T_{AMB}) - hA_1(T_{BRICK} - T_{GAS}) - K[(T_{BRICK} + 460)^4 - (T_{GAS} + 460)^4] \quad (2)$$

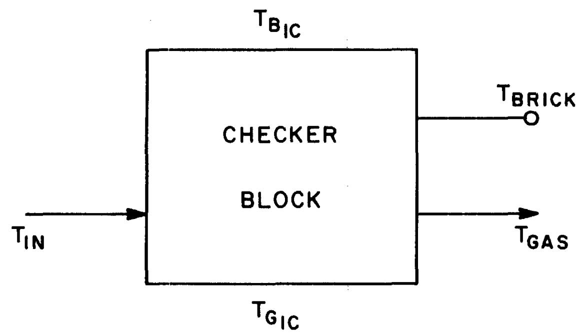


Figure 18. Checker block.

where σ_A = specific heat of the gas,
σ_B = specific heat of the brick,
V = volume of the checker,
M = mass of the checker,
ρ_2 = gas density,
F_1 = gas flow,
A_1 = heat-transfer surface area of brick,
h = conductive heat-transfer coefficient,
K = radiation heat-transfer coefficient,
T_{BRICK} = checker brick temperature,
T_{GAS} = checker gas temperature, and
T_{IN} = input gas temperature to checker.

These differential equations were programmed in FORTRAN and used to define the characteristics of a checker-block, shown in Fig. 18.

The following assumptions and approximations hold for Eqs. 1 and 2.

Assumptions

1. Heat transfer by radiation and convection.
2. Temperature of checker is a function of time and space (1-dimensional).
3. Checker temperature is uniform in any plane perpendicular to flow.
4. Gas temperature is uniform in any plane perpendicular to flow.
5. Brick thermal conductivity is infinite.

Approximation

1. Distributed temperature in each checker is represented by a lumped parameter system of three stages.

The generalized block of Fig. 18 has one input, the entering gas temperature, and two outputs, the exiting gas temperature and the internal brick temperature. Once the block has been programmed and checked out, the user can connect any number of these together to represent the system by simply using the DSL/90 statement:

TGAS, TBRIK = CHEKR(TGIC, TBIC, TIN),

where TGAS = output gas temperature of checker,
TBRIK = internal brick temperature of checker,
TGIC = initial gas temperature,
TBIC = initial brick temperature, and
TIN = input gas temperature.

Figure 19 shows the block model of one complete checker. Three checker blocks have been used

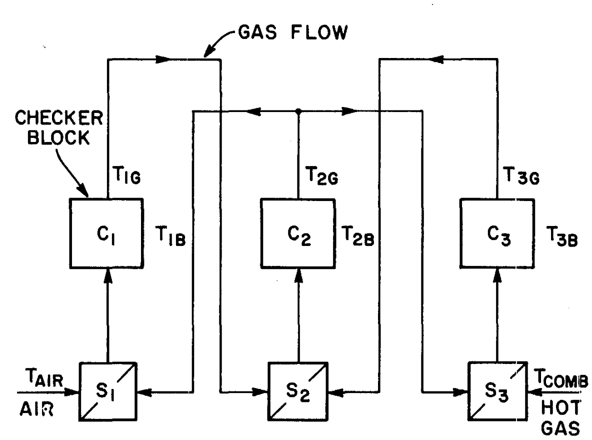


Figure 19. Block model of checker gas flow.

together with three switching blocks that reverse the flow direction through the blocks.

Now if this block model is used as a model of each checker, the DSL/90 statements which represent this system can easily be written by the user in terms of the basic checker blocks as follows:

```
* ... STRUCTURE STATEMENTS

* ... CHECKER SWITCHES
      C1IN=INSW(TRIGR,TAIR,TGAS2)
      C2IN=INSW(TRIGR,TGAS1,TGAS3)
      C3IN=INSW(TRIGR,TGAS2,TCOMB)
      C4IN=INSW(TRIGR,TCOMB,TGAS5)
      C5IN=INSW(TRIGR,TGAS4,TGAS6)
      C6IN=INSW(TRIGR,TGAS5,TAIR)
      TRIGR=-0.5+STEP(TREVRS)

* ... HOT CHECKER BLOCKS
      TGAS1,TBRIK1=CHEKR(TG1IC,TB1IC,C1IN)
      TGAS2,TBRIK2=CHEKR(TG2IC,TB2IC,C2IN)
      TGAS3,TBRIK3=CHEKR(TG3IC,TB3IC,C3IN)

* ... COLD CHECKER BLOCKS
      TGAS4,TBRIK4=CHEKR(TG4IC,TB4IC,C4IN)
      TGAS5,TBRIK5=CHEKR(TG5IC,TB5IC,C5IN)
      TGAS6,TBRIK6=CHEKR(TG6IC,TB6IC,C6IN)

* ... DATA

PARAM  F1=120000., SIGMAA=0.24, SIGMAB=0.24, ...
       TAIR=360.,  TCOMB=2800., TAMB=120., ...
       M=100000.,  A1=15000.,   A2=300., ...
       K=4.5E-06,  H=10.,       V=5000., ...
       TREVRS=15.

INCON    TG1IC=850.,  TG2IC=1300., TG3IC=1800., ...
         TB1IC=1600., TB2IC=2000., TB3IC=2500., ...
         TG4IC=2300., TG5IC=1900., TG6IC=1500., ...
         TB4IC=1300., TB5IC=1000., TB6IC=700.

PRINT    0.1, TGAS1, TGAS2, TGAS3, TGAS4, TGAS5, TGAS6, ...
         TBRIK1, TBRIK2, TBRIK3, TBRIK4, TBRIK5, TBRIK6, TRIGR, C1IN

CONTRL  FINTIM=30., DELT=0.01

PREPAR 0.05, TGAS3, TGAS6, TBRIK3, TBRIK6,TGAS1, TGAS4, TBRIK1, TBRIK4
GRAPH  6.0, 4.0, TIME, TGAS3, TBRIK3
LABEL    3RD CHECKER BLOCK TEMPS   RUN 4
GRAPH  6.0, 4.0, TGAS6, TBRIK6
LABEL    6TH CHECKER BLOCK TEMPS   RUN 4
END
STOP
```

Note that the parameter and variable names are almost direct symbolic equivalents of the physical notation used for describing the furnace.

Figures 20 and 21 show the actual plotted results of temperature variations at the outlets of the hot and cold checkers for a 15-minute flow reversal cycle. Advantages of this approach in addition to those already mentioned in example no. 1 include the ability to expand the simulation easily to include control system blocks and other system dynamics without disturbing the existing furnace simulation. This feature has proven particularly powerful in analyzing complex industrial processes.

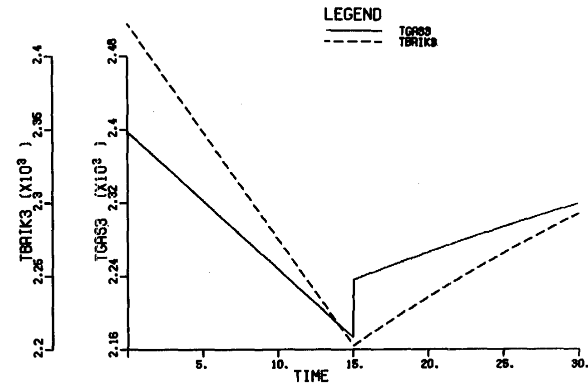


Figure 20. Third checker block temperatures, run 5.

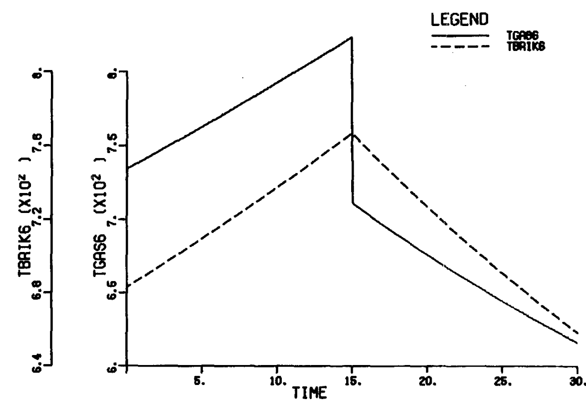


Figure 21. Sixth checker block temperatures, run 5.

Application No. 3—Saturn V Booster Rocket

Vehicle Description. This study applies digital simulation to the flight dynamics analysis of a large space vehicle booster. The problem illustrates the use of DSL/90 algebraic notation statements. In this study, the system example was drawn from the aerospace industry, but the use of DSL/90 algebraic notation can be applied to a broad range of problems including parts of the previous two examples.

The vehicle used in this study was the SATURN V launch vehicle for the APOLLO lunar mission. As shown in Fig. 22, the vehicle configuration consists of three booster stages and the APOLLO spacecraft. The overall length is 360 feet and, fully fueled, the vehicle weighs approximately 6 million pounds. The first, or S-IC, stage is powered by five F-1 engines, each of which provides a thrust of 1.5 million pounds. The four outboard engines are swiveled and provide for thrust vector control during powered flight. The SATURN V vehicle has an independent inertial navigation and guidance system from that in the APOLLO spacecraft in addition to a control computer and required sensors.

Trajectory. This simulation is concerned with the analysis of flight dynamics from launch through first-stage burnout. The booster-stage flight profile is shown in Fig. 22 and consists of a gravity turn for 150 seconds with separation occurring at approximately 60,000 meters altitude and a 2350-m/sec velocity. The rigid body equations of motion that were simulated form a perturbation set with respect to a reference frame moving along the nominal trajectory as shown in Fig. 23.

Axes X_1, X_2 X_3 form an orthogonal set, with X_2 aligned along the nominal velocity vector and axes X_1, X_2 lying in the nominal boost plane. The fuel sloshing dynamics of the first stage propellants were

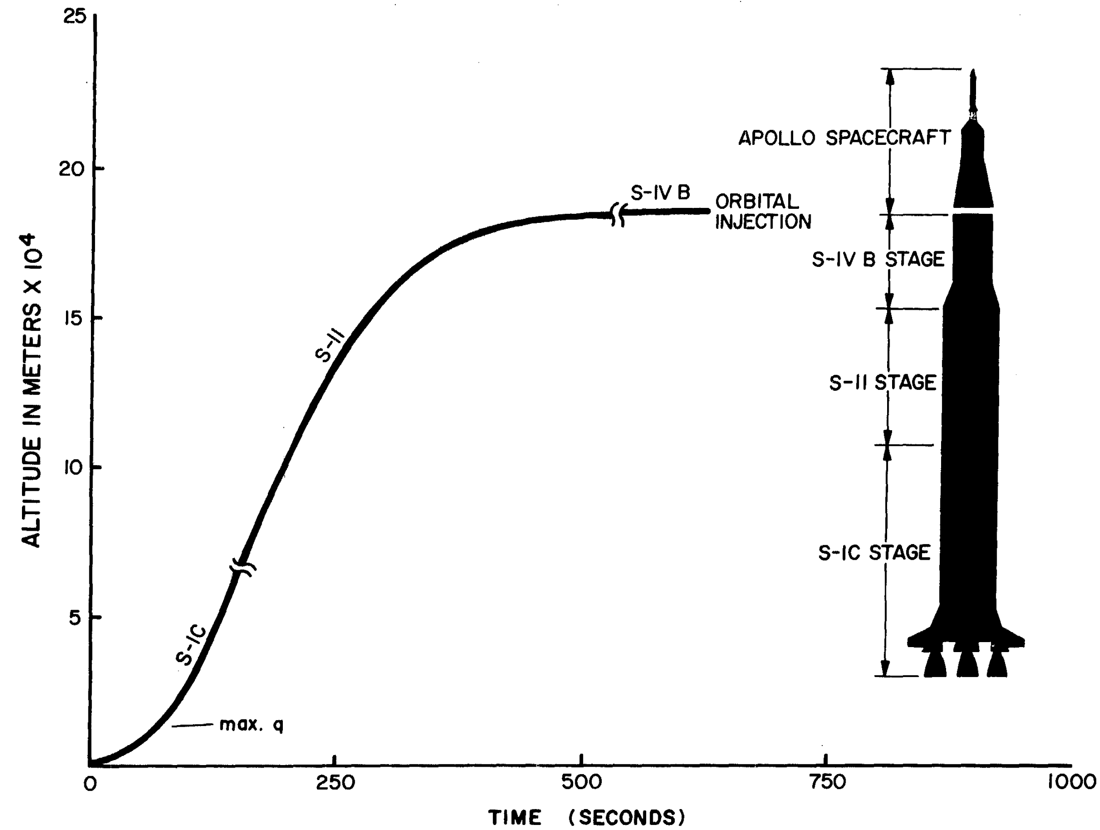


Figure 22. SATURN V configuration and flight profile (from Ref. 5).

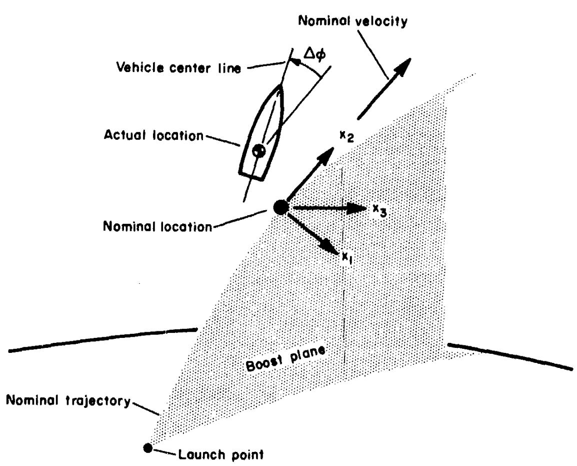


Figure 23. Reference frame axes (from Ref. 6).

included as well as the dynamic effects of elastic bending along the booster longitudinal axis. The attitude control system was also included in the simulation, together with the dynamics of the gimballed thrust VECTOR control system and hydraulic actuators for the engines, as shown in Fig. 24.

Since the defining equations of vehicle motion are far too complex for the purposes of this paper, the reader is referred to the basic documentation for the complete problem description. To illustrate the features of DSL/90, only a small portion of the larger problem will be treated—the pitch axis control system. Figure 25 is an expanded description of the control system filters, together with actuator and engine dynamics. The command signal filter block processes the pitch command signal from the control computer prior to applying it to the engine gimbal hydraulic actuators.

In order to investigate booster flight dynamics, a primary wind disturbance was applied to the vehicle during the first stage of powered flight as shown in Fig. 26. Horizontal wind loading was assumed, with varying azimuth angles for wind heading.

Referring to Fig. 25, the transfer functions for the command signal filter and engine dynamics can be expended in Laplace notation to yield the equivalent linear operational equations:

$$S^2\beta_2^c = K_1\beta_2^c|^u + K_2S\beta_2^c + K_3\beta_2^c \qquad (3)$$

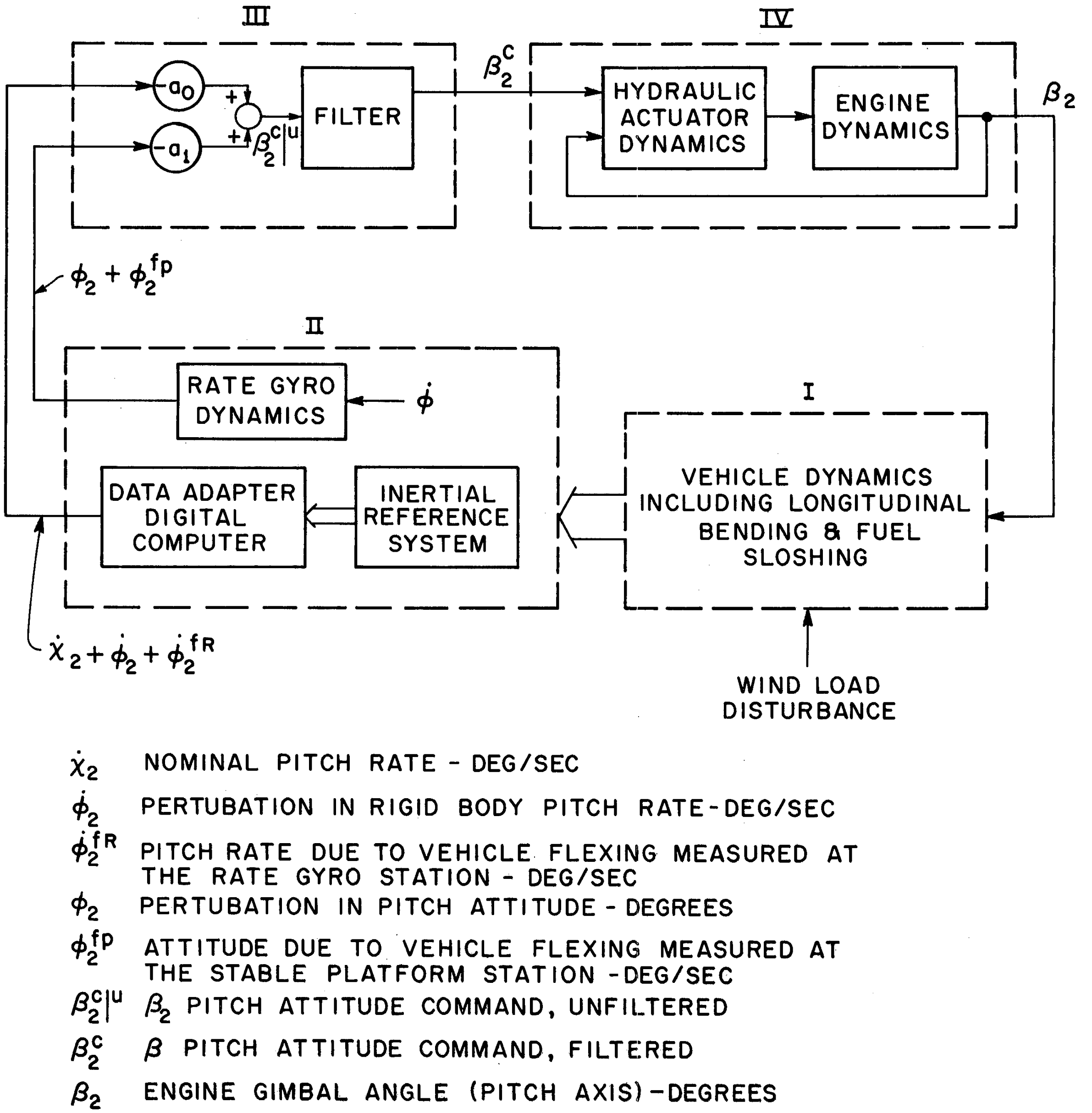


Figure 24. Simulation signal flow diagram (from Ref. 5).

and

$$S^2\beta_2 = (K_{32}\beta_2^c - K_{31}\beta_2)S + (K_{34}\beta_2^c - K_{33}\beta_2) + (K_{36}\beta_2 - K_{35}\beta_2)\frac{1}{S} \quad (4)$$

where S is the conventional Laplace operator.

From Fig. 24, the expression for the unfiltered pitch command signal $\beta_2^c|^u$ becomes:

$$\beta_2^c|^u = -[\alpha_0(\phi_2 + \phi_2^{fp}) + \alpha_1(\dot\chi + \dot\phi_2 + \dot\phi_2^{fr})] \quad (5)$$

Equations (3) through (5) can be directly programmed as DSL/90 statements as follows:

```
*   PITCH ATTITUDE CONTROL SECTION
    BET2CU = -(AO*(PH12+PH12FP)
             +A1*(CH12D+PH12D
             +PH2DFR))
    BET2CD = INTGRL(B2CDO, K1*BET2CU
             +K2*BET2CD+K3*BET2C)
```

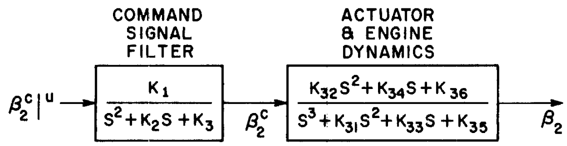


Figure 25. Pitch axes control system.

```
BET2C = INTGRL(BET2CO, BET2CD)
BET2DD = K32*BET2CD – K31*BET2D
         + K34*BET2C . . .
         – K33*BET2 + INTGRL(IC53,
         K36*BET2C – K35*BET2)
BET2D = INTGRL(BET2DO, BET2DD)
BET2 = INTGRL(BET2O, BET2D)
```

For the complete simulation, over 400 DSL/90 statements were required, not including the function generators and data statements. Both block and algebraic notation were used for describing the simulation configuration. The above small portion of problem coding is an excellent example of the ease of using both algebraic and block statements in DSL/90. Note the use of symbolic names for variable and data names which closely resemble the actual names. This feature has proven particularly helpful for large simulations.

The SATURN V flight dynamics were simulated for the first 120 seconds of powered flight. Figures 27, 28 and 29 show resultant DSL/90 plots for three of the system variables being studied.

The SATURN V simulation demonstrated several important features of digital simulation. First, a complex nonlinear aerospace problem could be successfully solved in DSL/90 by engineers relatively unskilled in programming. Second, many problems require both algebraic and block notation. The ability of DSL/90 to handle both of these requirements was amply proved. Third, problem solutions could be obtained quickly with a minimum of setup time. The original programming required approximately 16 hours of an engineer's time for problem setup. Each run of 120 seconds flight time required approximately 25 minutes of IBM 7094 computer time. In addition to the above features, DSL/90 allowed the user to model his problem in segments, checking out portions of the simulated vehicle independently, and then to hook these sections together. As an example, the trajectory equations form one section of the simulation, programmed in algebraic notation, of which the control system is another independent part programmed in block notation.

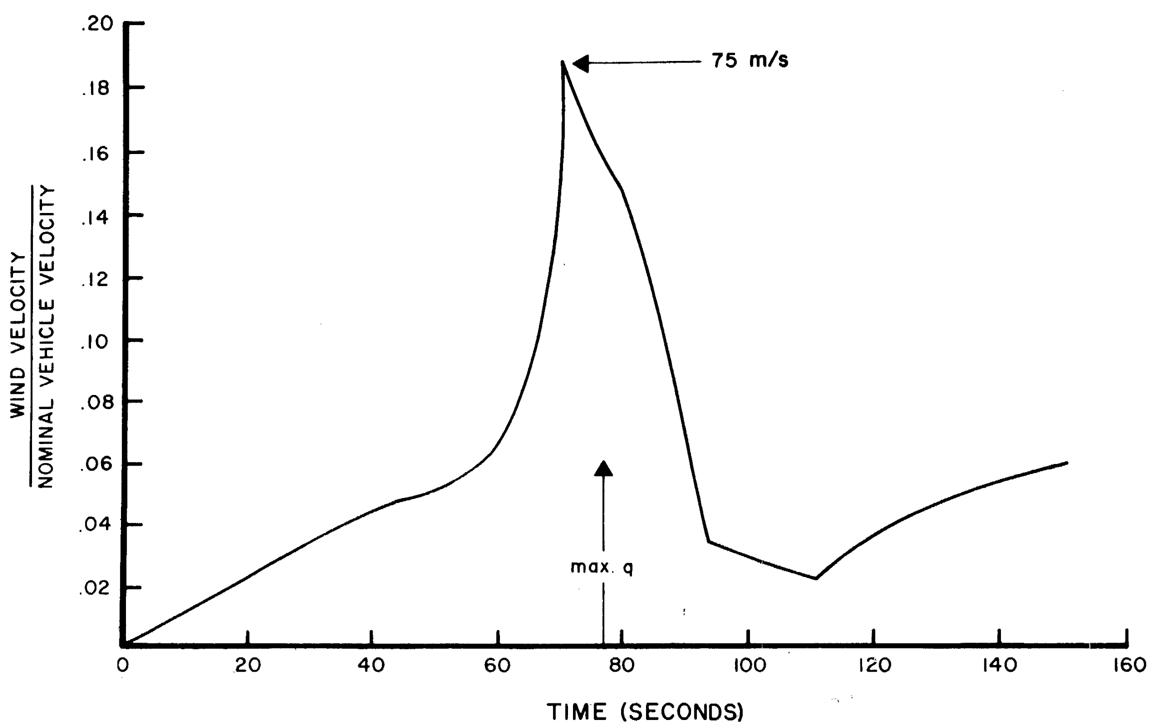


Figure 26. Primary wind disturbance (from Ref. 5).

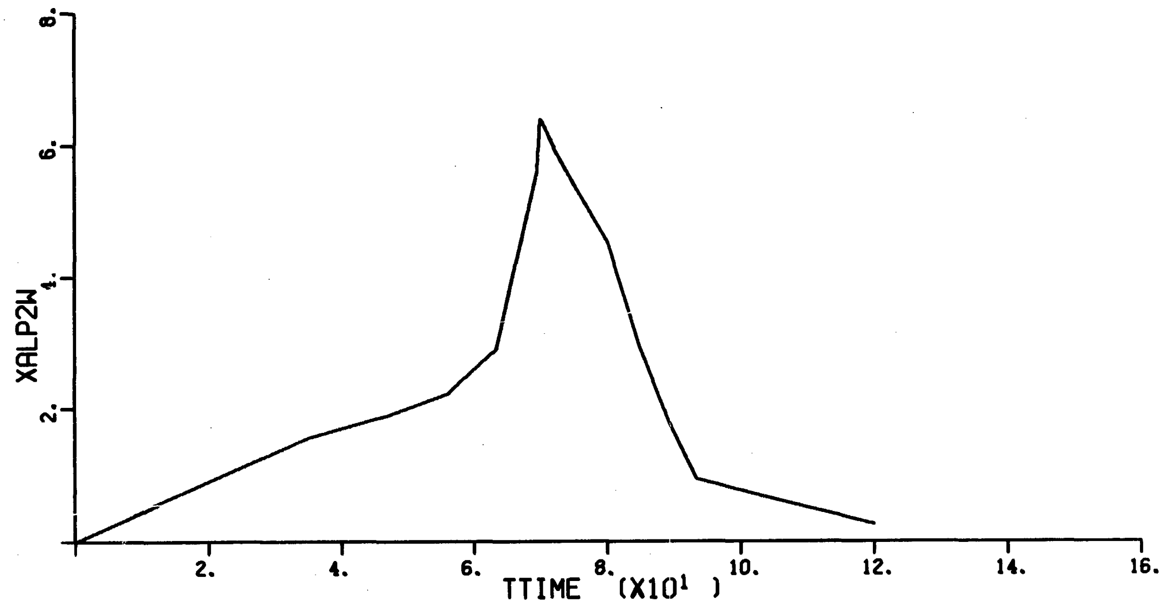


Figure 27. Pitch axis angular acceleration.

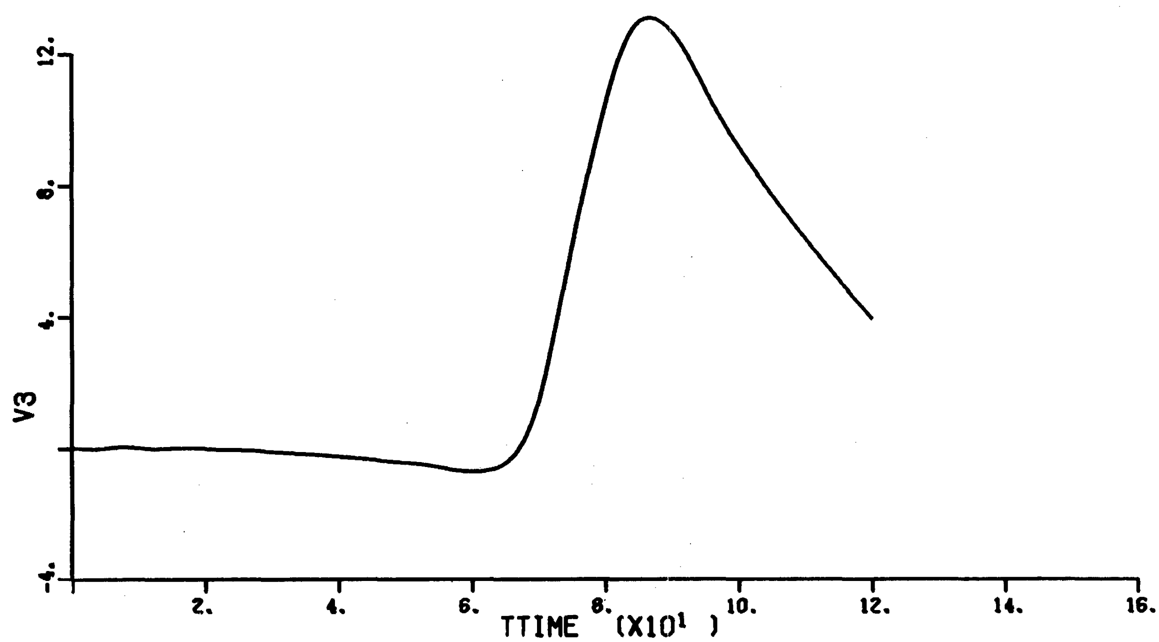


Figure 28. Velocity along X_3 axis.

CONCLUSIONS

Within IBM, DSL/90 has been used extensively in many different application areas including circuit design, mechanical dynamics, process analysis and control, servo design, aerospace flight simulation and biomedical modeling. Simplicity of the input language, clarity and completeness of both print and plot output, and the ease with which data is handled are some of the features which have made DSL/90 attractive to an increasing number of problem solvers from both camps—analog and digital. In DSL/90 workshops, it was observed that engineers with hardly any analog or digital computer ex-

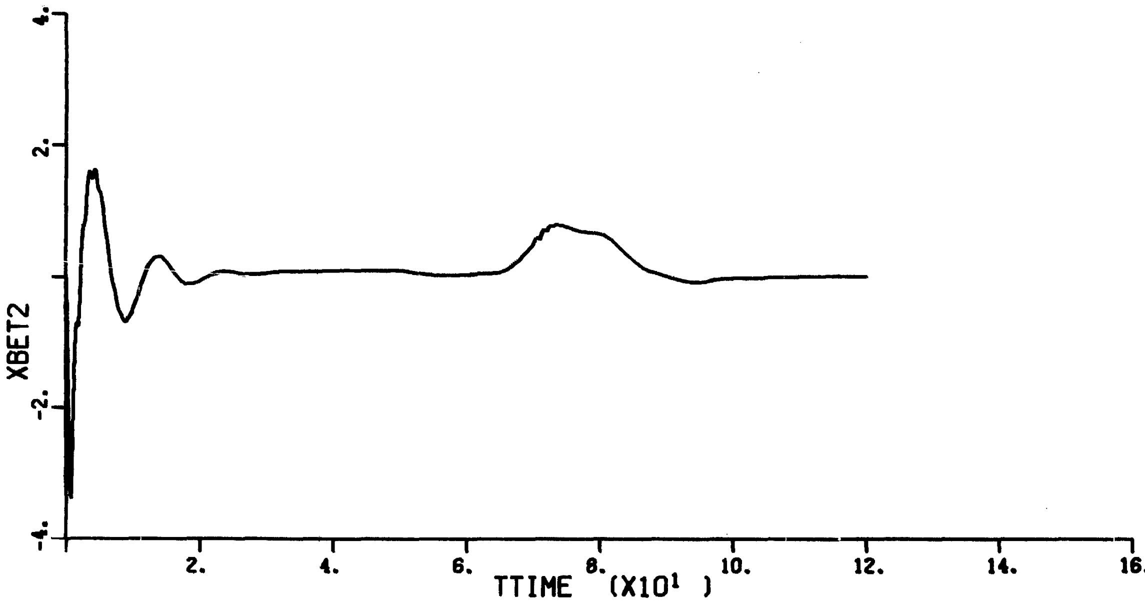


Figure 29. Engine gimbal angle for pitch axis.

perience successfully "programmed" in DSL/90 at the end of the first two-hour session. With this quick "shot" of confidence and further experience, many have proceeded to more difficult problems using the more advanced features of the language.

The examples shown indicate only a few of the broad range of problem areas to which DSL/90 can be applied. In addition to the above examples, DSL/90 has successfully simulated the process dynamics and control system responses for a paper machine dryer section control system. In this study, actual process noise gathered at the plant site was introduced into the simulation through the MAIN routine. Several nonlinear process and control elements were successfully modeled using the external block features of DSL/90, including nonlinear process controllers and scanning moisture gauges. DSL/90 was recently used for the simulation of an ammonia reaction process involving two-point boundary value matching. In this case, severe simulation problems were created by the fact that the system had two regions of time response, each governed by different differential equations and interfacing through initial values. Both the features of the "MAIN" program and the ability to introduce logical functions into the DSL/90 block structure were extensively employed.

Many of these simulation areas previously handled with analog techniques have long been troubled with problems of component reliability, accuracy, repeatability, and a lack of flexibility in modeling basic dynamic components and phenomena. In some respects, the trend toward digital simulation methods is a result of seeking answers to these problems. Some of the advantages of digital simulation as observed in the above application studies can be listed as follows:

1. Problem accuracy control.
2. Elimination of problem scaling.
3. Simulation run repeatability.
4. Reliable digital simulation elements.
5. Significantly reduced problem preparation time and simulation checkout time.
6. Simple problem coding. The majority of detailed circuit knowledge for analog programming is unnecessary.
7. Easy performance by the digital computer of some operations which at best are only approximated by analog computers.
8. Effortless provision of positive documentation of simulation configuration and parameter values.

To date, digital simulation techniques have shown themselves easy to learn, efficient to operate, accurate, and extremely flexible. They provide the engineer with an easy and quick method of digitally simulating complex systems, familiar block notation concepts, and the power of digital computation methods. The result represents a significant new simulation tool for engineering analysis and design.

ACKNOWLEDGMENTS

To our co-worker in this project, Mr. D. G. Wyman, we gratefully acknowledge his excellent contributions in both programming and concepts. We have benefited greatly from the many valuable suggestions of members of the computation laboratory, Systems Development Division, IBM, San Jose. Special thanks are due to Mr. A. H. Hoffman whose contributions to the exploratory program PLIANT paved the way for DSL/90.

The contributions of J. G. DeFares, P. E. Cowley, and F. Crane to the three application examples are particularly acknowledged.

BIBLIOGRAPHY

1. Brennan, R. D., and R. N. Linebarger, "A Survey of Digital Simulation: Digital-Analog Simulator Programs," *Simulation*, vol. 3, no. 6, pp. 23–36 (Dec. 1964).

2. ———, "A Survey of Digital Simulation: Part II—More Digital Analog Simulator Programs," ibid (to be published).

3. Dahlin, E. B., and R. N. Linebarger, "Digital Simulation Applied to Paper Machine Dryer Studies," *Proceedings*, Instrument Society of America, 6th International Pulp and Paper Instrumentation Symposium, Green Bay, Wisconsin, May 4–8, 1965.

4. DeFares, J. G., H. Hara, and E. M. Billinghurst, "The Stability of the Respiratory Servomechanism: An Analog Computer Study," *Progress in Biocybernetics*, Elsevier Publishing Company, New York, 1964, vol. 1.

5. Gunderson, R. W., and G. H. Hardy, "Piloted Guidance and Control of the SATURN V Launch Vehicle," *Proceedings*, IFAC Symposium on the Peaceful Uses of Space, Stavenger, Norway, June 1965.

6. Hardy, G. H., J. V. West and R. W. Gunderson, "Evaluation of Pilot's Ability to Stabilize a Flexible Launch Vehicle During First Stage Boost," Technical Note D-2807, National Aeronautics and Space Administration, Washington, D. C. (May 1965).

7. Shah, M. J., C. James and J. M. Duffin, "Simulation of an Ammonia Synthesis Reactor," *1966 Conference Proceedings*, International Federation of Automatic Control, London.

8. Wegstein, J., "Accelerating Convergence of Iterative Processes," National Bureau of Standards, Washington, D. C.