AMTRAN: Automatic Mathematical Translation

ROBERT N. SEITZ and LAWRENCE H. WOOD

Computation Laboratory, Marshall Space Flight Center, NASA, Huntsville, Alabama.

and

CHARLES A. ELY

Research Division, Brown Engineering Company, Marshall Space Flight Center, NASA, Huntsville, Alabama

I. Introduction

AMTRAN is a time-sharing remote-terminal computer system ultimately intended to permit scientists and engineers to "converse" directly with a computer in a "natural" mathematical language. Its current objectives entail attainment of the following goals:

a. "automatic" mathematical problem-solving,
b. high-speed, on-line, scientific programming,
c. a macro-compiler and operating system, and
d. the development of low-cost, $5,000–15,000, graphic display terminals.

In keeping with these objectives, AMTRAN is designed for three types of users: the scientist or engineer with little or no programming experience, the applications programmer, and the systems programmer.

Before describing the objectives of the AMTRAN system more fully and discussing our progress in attaining them, it seems appropriate to mention the current and projected embodiments of the system. A modest version of AMTRAN, including a prototype, low-cost graphics terminal, is operational on an IBM 1620 computer. The 1620 version utilizes a real-time interpreter and will be the principle topic of discussion in this paper. A more advanced but less complete version of AMTRAN is running on a Burroughs 5500 computer. It consists of an interactive "compiler" and a reentrant time-

sharing monitor with automatic disk-core overlay capability which runs within the framework of the B-5500 multiprogramming monitor. The B-5500 system is written in ALGOL 60 to facilitate conversion to other machines, and stores both source and "pseudo-object" code. The "object" code is executed interpretively but in a manner which under many conditions is almost as fast as, say, a compiled FORTRAN program. The B-5500 version is currently accessible over voice-grade telephone lines from teletypes or from a remote keyboard/Selectric printer terminal. Another implementation of AMTRAN is under development for the IBM 1130 computer with an operational $10,000 graphics terminal. This system was recently tied to the B-5500 via a voice-grade telephone line. Future plans call for the conversion of the B-5500 system to the UNIVAC 1108 computer.

It should be emphasized that AMTRAN is completely compatible with card, printer, and other batch-processing input and output devices and, on the B-5500 and IBM 1130 computers, should provide efficient batch-processing of production programs. However, its *raison d'être* is conversational use by the programmer and the nonprogrammer; and, for this application, interactive terminals are required. All three versions of AMTRAN were initially implemented using typewriter or teletype terminals to "get on the air." However, as soon as possible, we are shifting to low-cost, vector-display graphics terminals.

Such a low-cost terminal is exhibited in Fig. 1. It consists of a keyboard, a

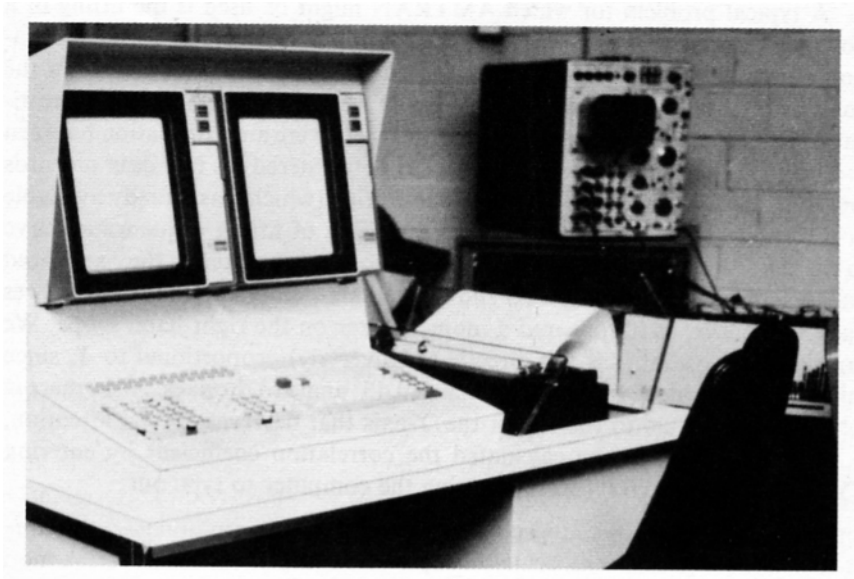

FIG. 1. AMTRAN terminal.

typewriter, and an 11-inch Tektronix storage scope for graphical output.[1] The keyboard consists of a standard typewriter keyboard, supplemented by a number of user-assignable buttons and a set of special function and operator buttons with labels such as SIN, +, d/dx, etc. In operation, the labels associated with each button are displayed on the left-hand storage scope in an automatically formatted form as the user depresses the buttons. As the user builds up his statement or mathematical expression on the left-hand scope he may modify or delete it. After he is satisfied with his statement, he releases it to the computer for processing, at which time, an identically formatted type-out of his expression is made on the typewriter. Computed results may be printed out on the typewriter or the scopes. Error and status messages are recorded in abbreviated form on the typewriter and are printed out in a more elaborate form on the scope. One of these terminals has been in operation for about 10 months and has exhibited distinct advantages over teletypes.

Future plans call for the replacement of the Selectric by a low-cost photostatic printer. The rapid (150–300 character per second) writing rate, the graphical input and output, the semiautomatic type-out of the on-line graphics terminals, and the favorable responses of the users have led us to conclude that such terminals are worth some additional expense. It should be noted that the 11-inch Tektronix display scopes provide extraordinary resolution, linearity, and freedom from edge effects compared to real-time scopes.

A typical problem for which AMTRAN might be used is the fitting of a polynomial to a set of data values by the method of least squares. The application illustrated here is a problem in thermal conductivity which arose in the analysis of the data from a micrometeoroid satellite experiment. The investigator was attempting to determine whether there were any correlation between pressure (X) and thermal conductivity (Y). He entered his raw data on cards and then used the DISPLAY.DATA subroutine (which was already available in the subroutine library) to examine the effect of fitting a quadratic curve to the data. Figure 2 shows his commands to the computer as they appeared on the left-hand scope. Figure 3 shows the data points and the least-squares quadratic curve, as it appeared a moment later on the right-hand scope. We can see (Fig. 3) that Y is not directly (or inversely) proportional to X, since this would be evinced by a straight line at a 45° angle to the axes. Also, there is a "wild" point just to the left of the Y-axis that deserves special attention.

Next, the investigator calculated the correlation coefficient by entering TYPE CORRELATION X, Y, causing the computer to type out:

```
CORRELATION X, Y = -.19023804
```

[1] Storage scope terminals of a similar type were pioneered by G. J. Culler and B. D. Fried [1].

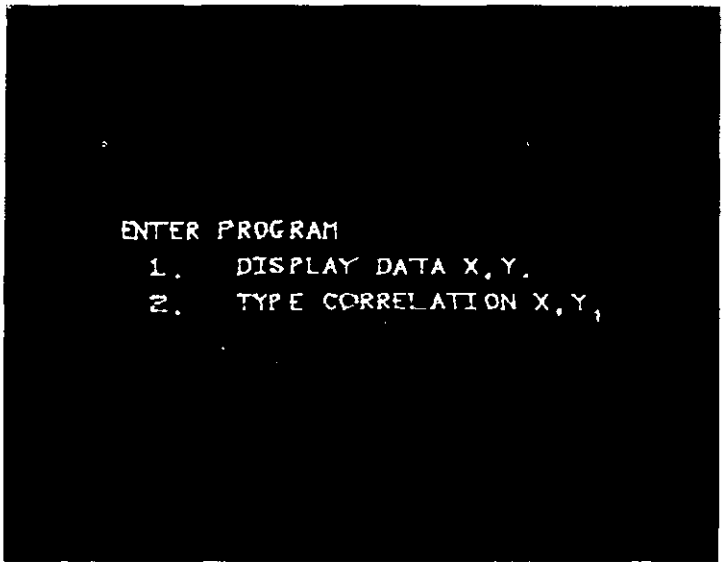

FIG. 2. Man–Computer dialogue for correlation calculations.

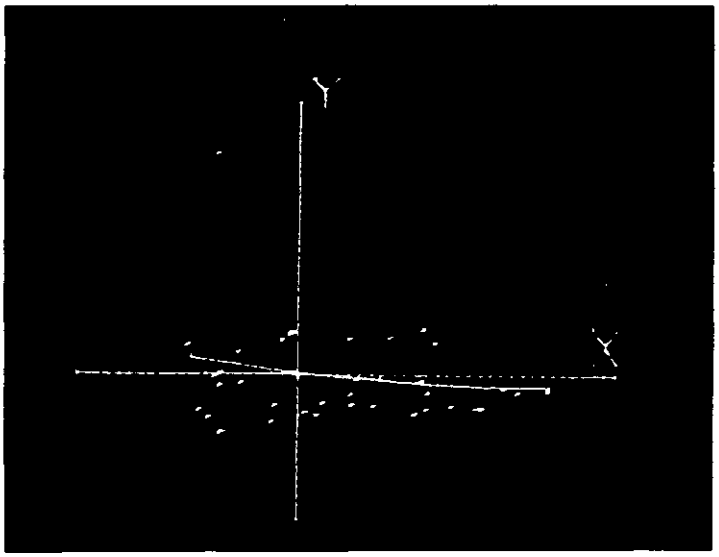

FIG. 3. Display of quadratic fit to thermal conductivity data.

Although in this case, the raw data was entered from cards, it could also be entered through the keyboard or, in principle, from tape or on-line data input sources.

(It is possible that a full-scale, on-line statistical analysis system could have important implications for the social scientist. It might permit him to analyze and manipulate data as easily as the physical scientist manipulates analytical expressions.)

II. Mathematical Problem-Solving Systems

Several institutions in the United States are attempting to develop "automatic" mathematical problem-solving systems which will either solve the user's problems or will warn him if they cannot. Such systems would examine and classify the user's input expressions, would interrogate him if additional information were necessary, would select the appropriate algorithm for solution of the problem, and would monitor error during the course of the computation. This is an ambitious endeavor. Before describing any AMTRAN efforts in this domain, we wish to examine certain general questions regarding "automatic" mathematical problem-solving systems.

First of all, how far can such systems go? Where will the line be drawn between the user and the computer?

The answer to these questions will have to be determined through experience. However, many problems in applied mathematics, although they call for a wide range of mathematical knowledge, do not require the development of new problem-solving techniques. This suggests that many of the common problems in science and engineering, such as the solution of well-behaved linear and nonlinear ordinary and partial differential equations, the evaluation of integrals, the solution of simultaneous algebraic equations, and the least-squares or mini-max fitting of curves to data, should be potentially solvable by such a system.

Second, how useful would such systems be to the scientist or engineer? Many of the problems brought to the computer require "custom" treatment. Could a "prefabricated" mathematical problem-solving system solve enough real-world problems to make its development worthwhile? We think that the answer to this question is an emphatic "yes."

The ability to key into a computer, say, a system of nonlinear differential equations and to receive an automatically scaled and formatted plot of the solution curves almost immediately, might revolutionize science and technology. In principle, this can be done today, but in practice it is a tedious and cumbersome task. To the computer scientist who is skilled in numerical and programming techniques, an automatic mathematical problem-solving system

may generate little interest, but to the user who doesn't want to spend the several years necessary to become a programming and numerical expert, an on-line problem-solving system becomes a passport to a new world of non-linear mathematics.

Third, if some computer scientists are unenthusiastic about automatic mathematical problem-solving, it should be mentioned that some mathematicians also feel the same way. Two reasons given for this reaction are that such a system would reduce the user's need or desire to study mathematics, and that such a system would lead the user down a primrose path by masking the inherent difficulty and variety of numerical analysis. We feel that the first of these reasons is not a cause for concern. The introduction of the slide rule and the desk calculator didn't eliminate the need for teaching arithmetic. The second concern, that of the user developing misplaced confidence in his automatic problem-solving system, would seem to be more serious. Yet this problem arises in other areas. It would seem that we should not refrain from developing the best system we can because it may be misused. We would hope that the philosophical problems would work themselves out later.

A fourth matter relating to automatic mathematical problem-solving systems is that they would seem to be logical points of departure for problem-oriented languages (at least in science and engineering). A wide variety of problem-oriented languages is possible but they all have one thing in common: applied mathematics. They are only as good as the mathematical subroutines which underlie them. Since these problem-oriented languages are all built upon a foundation of applied mathematics, it would seem that, in the interest of efficiency, it is here that a major effort should be placed. Even problems which require the trial and error application of various mathematical techniques are potential candidates for solution, although this may get into a gray area of the man–machine interface in which it is more economical for the qualified numerical analyst to make the decision rather than the computer. In some situations, the computer might make the trial and error selection itself.

This suggests three guidelines which we think could be observed advantageously by the designers of automatic mathematical systems.

a. We should begin with rote-mechanical types of solution procedures, and should work up gradually to more flexible pattern recognition or trial and error techniques. The "cookbook" approach, in which the human programmer tries to anticipate and to provide for the problems that may arise, should yield big dividends soon. This, in turn, should generate the support necessary for the large-scale task of developing economical, higher-level adaptive systems.

b. The user ought to be able to override the automatic problem-solving system at any time. He ought to be able to use only selected portions of the system, or to block it out altogether, if this is his desire.

c. Certain major advantages may accrue to an integrated automatic problem-solving system that are not available to libraries of isolated subroutines. In the integrated system, a division of labor occurs. The functions or the data points may be examined by the system when they are generated or entered, and may be monitored thereafter for error (e.g., differencing error) by the system as the calculation proceeds. This means that, after the initial examination, subsequent operations may be carried out by fast, simple, general-purpose subroutines which depend upon the checks and standard format established when the function was originally generated. This is in contrast to the situation with isolated subroutines. Here, the user must either pick out an efficient special-purpose subroutine or must draw upon a general-purpose subroutine which must make elaborate tests and must be self-sufficient.

Within the AMTRAN system the principal effort to develop such a mathematical system has been invested in a basic package for numerically representing formulae, locating real zeros and extrema of functions, integrating and differentiating functions, numerically solving ordinary and partial differential equations, etc. Primary emphasis in the future will probably be upon those problem areas which are not currently included in Purdue's NAPSS system [2]. Some automatic mathematical routines which are currently available in AMTRAN are listed in Table I.

TABLE I

ADAPTIVE NUMERICAL ANALYTICAL PROBLEM-SOLVING ROUTINES

REPRESENT	Numerically represents mathematical formulae, selecting near optimum step sizes, assuming a cubic polynomial fit. Automatically detects singularities, cusps, discontinuties, and "hairpin" extrema.
∫	Numerically integrates mathematical formulae using variable-step-size error-controlled algorithms. Accommodates the general integral $\int_{g(x)}^{h(x)} f(x, x')\,dx'$ including special cases such as $\int_A^B f(x')\,dx'$, $\int_x^B f(x')\,dx'$, etc.
DERIV	Calculates the numerical derivative using a cubic fit (variable interval spacing.)
SYMDIF	Symbolically derives the analytical derivative, given the analytical formula. The resulting analytical derivative is in the form of an executable code string, which may be evaluated to give numerical values for the derivative.

TABLE I continued

SOLVE	This operator solves sets of simultaneous algebraic or ordinary differential equations. The differential equation solver currently uses a variable-step-size, Runge–Kutta integration package developed at Aerospace Corporation which will handle any number of simultaneous first- or second-order differential equations. Error control is provided by a Simpson's rule check on the fourth-order Runge–Kutta integration formula. The algebraic equation solver currently uses the Crout reduction technique. It can also solve partial differential equations by the method of characteristics.
INVERT	The INVERT routine, when applied to a scalar, gives the reciprocal; when applied to a numerical representation of a monotonic function $Y(X)$, gives the functional inverse $X(Y)$; and when applied to a matrix A, gives the inverse matrix A^{-1}.
ZEROES	Locates all real zeroes within the range of definition of functions which have been generated by the REPRESENT operator. Gives warning if multiple roots are possible.
MINIMAX	Locates all relative extrema within the range of definition of functions which have been generated by the REPRESENT operator.
LET	The LET operator causes a numerical change of variables (for functions known only in tabular form).
STEP . FCT	STEP . FCT (T) is the unit step function defined by $u(t) = 0, \quad t < 0$ $u(t) = 1, \quad t \geq 0.$
INTERPOLATE	Given two monotonic sets of numbers X and Y in one-to-one correspondence with each other, and a new set of x's ($X1$), the INTERPOLATE operator provides a new set of y's ($Y1$) corresponding to the $X1$'s. A Newton third-order interpolation formula is used. The $X1$ array need not be the same size as the X and Y arrays and, for example, may consist of only a single number. The form is $Y1 =$ INTERPOLATE $X1$, X, Y.
CUBIC	CUBIC accepts X and Y as inputs and generates four arrays A, B, C, and D of coefficients for the overlapping cubic $AX^3 + BX^2 + CX + D$ fits for $Y(X)$.
SCOPE	This nonmathematical automatic-formatting display operator may be followed by various modifiers and combinations of modifiers such as: POLAR, LOG.X, LOG.Y, VECTOR, GRID, HACHURE, MAGNIFY, etc. One or more curves may be plotted simultaneously with the scale factor automatically determined so that it is common to all the curves. In the absence of any such modifiers, the system displays the data in Cartesian form, selecting one of 25 different plotting formats, based upon the data, with printed scale factors and labeled axes.

TABLE I continued

SUM . OF . SERIES	SUM . OF . SERIES operator is used to expedite the generation of functions by series expansions.
ERF (X)	The ERF (X) operator accepts an array expression as input and yields an array of error function values defined by $$\frac{2}{\sqrt{\pi}}\int_0^x \exp(-t^2)\,dt$$ as the result.
LAPLACE	The LAPLACE operator accepts an array $f(t)$ of exponential order as input and delivers the numerical representation of the Laplace transform, $F(s)$, as output.

The AVERAGE, SIGMA, MOMENTS, REGRESSION, and CORRELATION routines are self-explanatory statistical operators. The LEAST, SQUARES operator provides the coefficients for a quadratic least-squares fit to numerical data $Y(X)$. All the trigonometric and hyperbolic operators are also present.

Certain other operators based upon these fundamental algorithms are either available or are readily constructed.

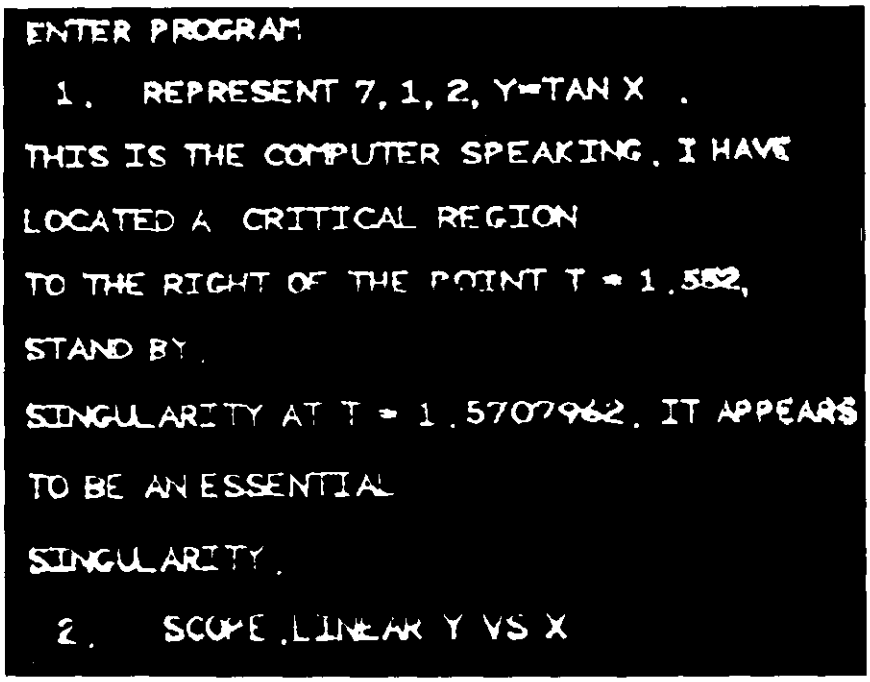

FIG. 4. Man–Computer dialogue for automatic representation of tan x.

To give a specific example of how the automatic mathematical system works, we consider the following problem:

1. REPRESENT 7, 1, 2, Y = TAN X.
2. SCOPE VECTOR, Y VS X.

In statement 1, the first entry after REPRESENT gives the desired number of decimal places (7) accuracy required. The 2nd and 3rd entries give the desired range of X (1 to 2) over which the function is to be represented. The 4th entry (Y = TAN X) is the expression which is to be represented. Figure 4 shows the response from which the computer when it encounters the singularity at $X = \pi/2$ in representing the function TAN X. In statement 2, the results are displayed on the scope, using the automatic formatting routine (see Fig. 5).

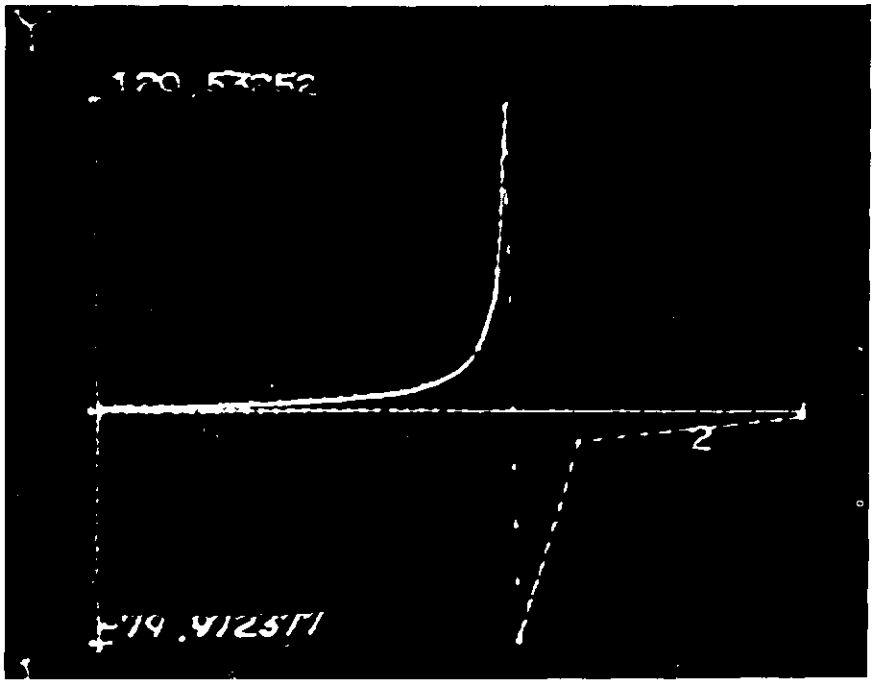

FIG. 5. Display of automatic representation of tan x.

III. AMTRAN as a Programming Language

A major goal in developing AMTRAN was to speed up and facilitate the programming process, since the programming requirements of an automatic problem-solving system are so formidable. This was to be accomplished, first, by providing on-line debugging and editing capability for rapid check-out of programs, and second, by allowing the programmer to "bootstrap" his problem-solving system by defining higher and higher-level "instructions" in order to provide fewer and larger building blocks for his programs.

The first objective was attained. Turnaround time has been greatly reduced in the debugging of programs. Since on-line terminal access to computers is becoming widespread, we will not dwell further on this aspect.

The second objective, that of reducing programmer effort, requires a description of the AMTRAN language. Only the restricted version of AMTRAN for the IBM 1620 computer will be discussed here. It most closely resembles a blend of FORTRAN and ALGOL. It resembles FORTRAN in its use of certain symbols such as the = sign and its lack of formal declaration statements. It resembles ALGOL in its ability to provide recursive procedure calls, subscripted subscripts, compound statements, free use of mathematical expressions, use of the ALGOL IF test, and in certain other ways. However, 1620 AMTRAN also differs from these languages in certain respects, including the following:

A. Automatic Definition and Dimensioning of Arrays

Arrays are automatically defined and dimensioned when they are first created. For example, the statement X = ARRAY 0, 1, 50, will generate the numerical representation of the independent variable X (i.e., a ramp function) over the numerical range from 0 to 1 with 50 intervals. Then the command

```
Y = CONCATENATE X and X.
```

will create a new array Y which is twice as large as the X array. *Y does not have to be defined by a dimension statement prior to its use.* A third example is that in which we write

```
X SUB 51 = 1.02.
```

where X SUB 51 is the (previously nonexistent) 51st subscript location in the X array. If we transfer data to a subscript location which lies outside of an array, the array is automatically enlarged so that it can accept the data.

B. Implied Array Arithmetic

With implied array arithmetic, when an array is multiplied by a scalar, the

operation is automatically carried out on each element of the array without the need for writing a DO loop (or a FOR statement). For example, the statement

```
Y = 2 SIN X,
```

will cause the computer to take the SIN of each value of X, multiply each value of SIN X by 2 and store each value of 2 SIN X in the corresponding location of Y (defining and automatically dimensioning Y, if it has not been previously defined).

One purpose of the automatic array arithmetic is to make it possible to deal with matrices and with the numerical representations of functions as mathematical entities. Of course, the user may override the array arithmetic and may write his own loops if he so desires.

The automatic array concept has also been extended to other operations such as the IF test. For example, the statement

```
IF |X| > .5, THEN X SUB INDEX = 0.
```

will cause the computer to automatically test each subscripted value of X and set equal to 0 each value of X greater than .5.

The implied array arithmetic has afforded an important fringe benefit for the 1620 AMTRAN interpreter by greatly enhancing its average speed of execution. This occurs for the following reason. Computers tend to spend most of their running time operating in program loops. When implied array operations are carried out in AMTRAN, efficiently programmed loops are employed. This means that, although the linkage from one implied array operation to the next may take much longer than it should (because of the inefficient 1620 interpreter), the overall degradation in performance is moderate because most of the time is spent in efficient operation in the implied array loops, However, in Runge–Kutta integration, in which each value depends upon the preceding value and implied array arithmetic cannot be used, the 1620 interpreter becomes exceedingly slow and inefficient. Efforts have been made to overcome such problems on the B-5500 and IBM 1130 versions of AMTRAN.

In our experience, implied array arithmetic has been a success. No particular drawbacks or penalties have been experienced.

C. Symbolic (List) Processing

AMTRAN provides three types of variables: data, string, and text. Data variables are self-explanatory, text or literal variables are lists of characters, while string variables are used to store and manipulate executable strings of code. String variables differ from text variables in that, with string variables,

labels or reserved words are represented by numbers rather than by the actual multicharacter labels themselves. This permits the programmer to deal with a reserved word as a single entity or symbol, and it also bypasses the time-consuming conversion of labels to computer code when recompiling a symbolic variable. Functions and procedures may be regarded as string variables.

D. The Expandable Instruction Set

In most high-level languages, the user has the ability to write his own higher-level procedures or subroutines in order to extend, in effect, the basic instruction set of the system to fit his particular needs. However, in certain languages such as LISP, it is possible to incorporate user-defined procedures *into the compiler* so that there is no clear-cut dividing line between procedures and the "*intrinsic*" instructions of the system. This makes it possible to "bootstrap" the compiler, and to provide larger (as well as smaller) building blocks for the programmer.

This was a key principle in the development of AMTRAN. In order to facilitate the process, several extensions have been made to the rules for defining procedures:

a. Binary procedures may be defined in which one operand precedes and one operand follows the operator. For example, the vector cross product can be a user-defined procedure and can be used in the usual A × B format. The 1620 system does not provide for establishment of levels of hierarchy among binary operators but we hope to provide for this in the B-5500 and IBM 1130 systems.

b. When calling a procedure from the keyboard, execution may begin *before all of the parameters have been passed to the procedure.* Furthermore, the number of input parameters may vary widely. We have called such procedures "context-sensitive" since the first few expressions which are passed to them determine the number and kind of additional parameters which they will request from the user. For example, consider the SOLVE operator. This procedure is written in AMTRAN. To use it, the scientist enters SOLVE followed by his equation or equations. These equations are analyzed and classified within the SOLVE procedure, which then requests from the user whatever additional inputs it may need to solve the problem (anywhere from 1 to 100). The essential point is that the classification process and the request for inputs is a part of the SOLVE procedure which must be executed before the computer knows what inputs to request from the user. Furthermore, a variable number of inputs may be required. For the 1620 interpreter, this poses no problems, but for a line-at-a-time compiler it is difficult to accomplish. To illustrate the point, suppose that the SOLVE operator had been written in FORTRAN. In that case, a parameter string of some certain length would have to be designated when the subroutine was programmed. To

use the subroutine later on, a parameter string of *that fixed length* must be entered by the user *in its entirety* before he enters the end-of-line character and execution begins. This means that "context-sensitivity" is difficult to achieve. *Yet "context-sensitive" interactive procedures are very important if an interactive system is to respond flexibly to a user's input!* This is a problem we encountered which does not appear to arise in a batch-processing environment but which is apparently unique to the conversational mode.

c. Code strings can be passed to procedures. This has been discussed in a previous section.

The modifiable command feature makes it possible for AMTRAN to simulate other languages. For example, the CONCATENATE operator, as well as REFLECT, FORWARD, SUMMATION, INTEGRAL, and MINIMAX are written in AMTRAN but are intrinsic to the 1620 interpreter. Simple new operators can be added to the system in minutes, which makes it difficult to draw a hard-and-fast line between the basic instruction set and those which are added by the user.

The authors should point out that the ability to expand or modify the AMTRAN instruction set can be a two-edged sword. It is of value in developing the system and can be used to good advantage by individual users. However, there must be some basic system which does not change form from one day to the next. Our present philosophy is to provide the user with a basic system, together with a disk file "public library" of "instructions" (extended procedures) which he can utilize but cannot modify without special dispensation. In addition, the user can define his own procedures and, within limits, can modify the AMTRAN system to suit his needs, but the master system remains in the background and he can always return to it.

Space does not permit much more than the foregoing sample of features specific to AMTRAN. There are commands unique to interactive computing, such as BACKSPACE, DELETE, EDIT, RESET, etc. There are standard format input-output operators such as TYPE, PRINT, and SET, which permit the user to "converse" with the computer without necessarily writing format statements.

There is automatic English formatting of code strings when the computer lists programs on the scope or punches out source programs on cards. Appendix I presents a list of the principal operators in the 1620 system. Two typical 1620 AMTRAN programs are presented in Figs. 6 and 7.

One pertinent question that might be mentioned at this time is: how does AMTRAN relate to the principal scientific languages of the day, such as FORTRAN, ALGOL and PL/1?

At the present time, AMTRAN programs cannot be converted to FORTRAN, ALGOL, or PL/1 because of certain AMTRAN features that come into play during execution.

```
1.  X, Y.
2.  ERASE, X1 = MINIMUM X, X2 = MAXIMUM X, X = .666(X - X1)/(X2 - X1) + .333.
3.  Y1 = MINIMUM Y, Y2 = MAXIMUM Y, Y = .5(Y - Y1)/(Y2 - Y1) + .25.
4.  WRITE.SCOPE X, Y.
5.  C = LEAST.SQUARES X, Y.
6.  X1 = MINIMUM X, X2 = MAXIMUM X, Y1 = MINIMUM Y, Y2 = MAXIMUM Y.
7.  X = ARRAY X1, X2, 50.
8.  Y = C SUB 0 X SQ + C SUB 1 X + C SUB 2.
9.  WRITE.SCOPE X, Y.
10. NAME.THIS DISPLAY.DATA 117.
```

FIG. 6. AMTRAN subroutine DISPLAY.DATA.

(This is the DISPLAY.DATA subroutine. The two parameters X and Y which are to be picked up by the subroutine are listed on the first line. ERASE causes the erasure of the scope. MINIMUM and MAXIMUM locate the minimum and maximum tabulated values in the X array. Statement 5 causes the polynomial coefficients for the LEAST.SQUARES curve fit to be stored in an array called C.)

```
1.  N, A1, X1, X2, Y, ENTRY, H = (X2 - X1)/10, X = X1, Y2 = Y, I = 0.
2.  J = 0, REPEAT N, Y3 SUB I = Y SUB J and I = I + 1 AND J = J + 1.
3.  IF X GT = X2, THEN GO TO 19.
4.  Y1 = Y.
5.  RUN, A = H Z, X = X + H/2, Y = Y1 + A/2, F = A.
6.  RUN, B = H Z, Y=Y1 + B/2.
7.  RUN, C = H Z, X = X + H/2, Y = Y1 + C.
8.  RUN, D = H Z, Y2 = Y = Y1 + (A + 2 B + 2 C + D)/6.
9.  RUN, A = H Z, X = X + H/2, Y = Y2 + A/2 + A/2, G = A
10. RUN, B = H Z, Y=Y + Y2 + B/2.
11. RUN, C = H Z, X = X + H/2, Y = Y2 + C.
12. RUN, D = H Z, Y = Y2 + (A + 2B + 2C + D)/6, RUN, D = H Z.
13. E = ABS(Y - Y1 - (F + 4 G + D)/3)/(.000001 ABS Y + A1), F = LOG MAGNITUDE K + 1.
14. IF E GT 1, THEN X = X - 2H AND Y = Y1 AND IF E GT 2, H = H/(EXP(F LN 1.5848931))
14.C AND GO TO 3, OTHERWISE H = H((.5/E)** .2) AND GO TO 3.
15. IF E LT = .016, THEN H = 2 H, OTHERWISE H = H((.5/E) ** .2).
16. J = 0, REPEAT N, Y3 SUB I = Y2 SUB J AND I = I + 1 AND J = J + 1.
17. J = 0, REPEAT N, Y3 SUB I = Y SUB J AND I = I + 1 AND J = J + 1
18. GO TO 3.
19. Y3.
20. NAME. THIS RUNGE1 104.
```

FIG. 7. Schlesinger–Sashkin adaptive Runge–Kutta subroutine for integrating simultaneous first-order differential equations.

[This is the adaptive Runge–Kutta–Simpson's rule integration subroutine developed by Schlesinger and Sashkin at Aerospace Corporation. In this routine, the ENTRY which appears in the first line causes the computer to pick up a code string (namely, the differential equation), while the word RUN causes the accepted code string to be evaluated repeatedly.]

There is no provision in the 1620 system for the incorporation of FORTRAN or ALGOL subroutines or procedures. However, the B-5500 AMTRAN system is written in ALGOL and ALGOL procedures can readily be attached to it. It is planned that both FORTRAN and ALGOL subroutines will be attachable to the B-5500 and IBM 1130 systems. It is hoped that the B-5500 and IBM 1130 systems can provide full conversational ALGOL when they are complete.

We are now ready to return to one of the questions posed at the beginning of this section: how much can AMTRAN reduce the labor involved in applications programming? It will be apparent at this point that we have sought to achieve such reductions by (a) reducing the amount of bookkeeping with which the programmer must cope, and (b) by expanding (and allowing the programmer to expand) the instruction repertoire of the system.

We do not yet have a clear and reliable answer to this question. However, in the eight benchmark tests which have been conducted on the 1620, improvements in programming, keypunching, and checkout times have been measured which range from 2 : 1 to 20 : 1 over on-line FORTRAN II.

One problem that has become apparent is that the addition of new high-level operators to the system frequently requires modification of the basic interpreter. The SOLVE operator is a good example of this, since it required modifications of the system to handle "context-sensitive" procedures and symbolic variables. This experience has led us to our third objective: that of writing the interpreter or "pseudo-compiler" in such a form that it may readily be reprogrammed.

IV. The Macro Compiler

The 1620 AMTRAN interpreter is a complicated, interwoven unit. Every time a modification is made to some part of it, other parts must also be modified. This has inspired us to look for a better way. We hope to construct the 1130 AMTRAN system in a completely modular way, including the time-sharing monitor and the disk-core overlay scheduler. The system would be "driven" by control lists of core addresses which would cause the computer to branch from one macro module to another. Of course, "bootstrapping" of compilers is not a new idea. However, a further objective of this effort would be to define a higher-level machine language. This could then be microprogrammed into a read-only memory in order to augment, with hardware, the efficiency of the interpretive process.

V. Low-Cost Graphics Terminals

We begin this section by reemphasizing that AMTRAN is not dependent upon a special keyboard terminal. AMTRAN can employ any standard I/O arrangement (e.g., punched cards or typewriter); however, our experience has indicated that the special terminals facilitate the AMTRAN conversational mode. We undertook the development of these terminals because, at the time the AMTRAN effort began (in 1964), low-cost ($10,000–15,000) graphics terminals with vector plotting capability were not commercially available. Like some others, we felt that storage scopes could be the key to low-cost graphics by eliminating the need for a small computer at the terminal to provide a continuously refreshed display. Since that time, Bolt, Beranek, and Newman has marketed a low-cost graphics display using a double keyboard and a 5-inch Tektronix storage scope tied to a voice-grade telephone line. Recently, M.I.T. announced plans to build a number of low-cost graphics terminals using the recently developed, high-resolution 11-inch Tektronix storage scopes tied to a voice-grade telephone line [4]. Their target price for the terminal, having local character generation, a keyboard, a graphic input device, and interfaced with a 201 data set, is $5,000. If present plans materialize, high-quality graphics terminals in the $5,000–15,000 price range should be available from several manufacturers in the near future.

It should be noted that these terminals achieve part of their economies by depending upon the computer for all display preparation and modification.

Our experience with the 1620 AMTRAN terminal has been favorable. For many of our applications, a principle advantage of the scopes over the typewriter has been the ability to print messages rapidly on the scope. This effectively provides a low-cost, on-line printer at the terminal. However, the graphics aspect has been quite valuable too.

The electric typewriter, which provides hard-copy output, has slowed down our operations. It types out the user's commands to the computer as the computer receives them, as well as typing out specific data values and error messages. It is too slow for bulk data output, such as the listing of programs or the printing of the instruction manual. We hope that low-cost photostatic printers will be able to replace the typewriter within the next year or two.

Our latest version of our keyboard is shown in Fig. 8. We have tried to ensure that each distinct input (e.g., +, SIN, DELETE, A, 1, etc.) should require a minimum of button pushes (ideally, only one). Of course, a typewriter must be employed for the entry of the less commonly used labels and mnemonics. However, the use of the typewriter is somewhat tedious and a source of error, particularly if caseshifts are involved. Also, the typing of mathematical expressions is more difficult than the typing of text, and a well-

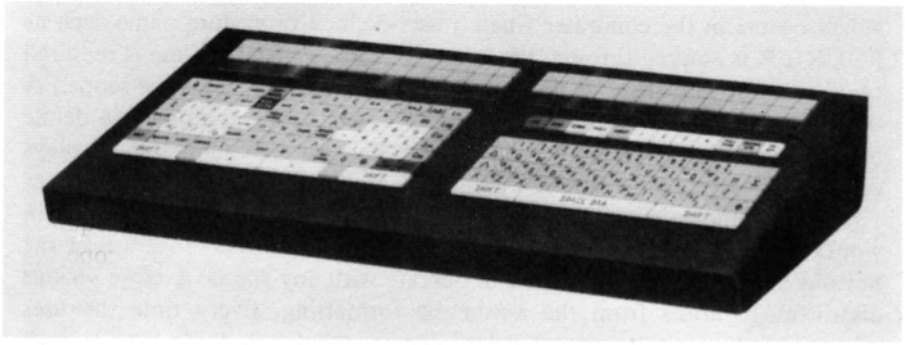

FIG. 8. New AMTRAN keyboard.

organized keyboard facilitates communication with the computer. Our keyboard contains the following:

1. There is a typewriter keyboard with the letters and a few punctuation marks organized in the standard manner. Case shift for uppercase letters has been utilized on the typewriter keyboard since we naturally expect them.
2. There is a number entry island consisting of the ten digits and decimal point, *and these are accessible without a case shift.*
3. Special symbols such as +, −, [, Δ, etc., are available *in lowercase.*
4. Special control operators such as backspace, delete, and the end-of-line character, also have separate buttons.
5. Several blank buttons have been provided for on-line programming,
6. The commonly used reserved words such as IF, TYPE, SIN, ∫, etc., have been assigned to separate buttons and organized for efficient use (e.g., all trigonometric buttons together).

An experimental keyboard which incorporates the above guidelines is part of the 1620 AMTRAN terminal shown in Fig. 1. In operation, each button sends a distinct 3-digit, BCD number to the computer. The computer then automatically formats and displays the label associated with the button. Thirty to fifty "blank" buttons are available to the user, to which he can assign his own labels *or symbols* attached to his more commonly used procedures and variables. (His less commonly used operators and operands are called through the typewriter keyboard in the usual teletype manner.)

This approach has its own advantages and disadvantages. On the positive side of the ledger, the automatic formatting facilitates the rapid and correct entry of mathematical expressions. In the future, it should be particularly useful in natural mathematical formatting of mathematical equations. Another advantage is that the direct transmission of a numerical code to the computer obviates the need for the time-consuming label recognition process

which occurs in the computer when a user-defined procedure name such as FOURIER is entered through the teletype. (Little computer time is required to convert the numerical code into a label and to display it on the scope.) A third important advantage of the large keyboard is that the user can define his own symbols and can attach them to the keys, or can manipulate displays on the typewriter or the scopes *without affecting those displays.*

A disadvantage of the large keyboard is that it takes longer to learn than would be the case with a smaller keyboard. A new user has to learn how the buttons are organized before he can operate with any speed. A more serious disadvantage arises from the automatic formatting. Every time the user presses a button, the computer must interrupt what it is doing long enough to find and format the label associated with the button, and send it down an output line to the scope. This "formatting" service is separate and distinct from the serious computing which occurs when the user enters the end-of-line character; at that time, there is no difference to the computer between the keyboard and the teletype. However, the "formatting" service for keyboard labels leads to about ten times the interrupt frequency that is required by a buffered teletype. This "formatting" service can probably be furnished by a large, time-sharing computer system at negligible cost in computing time, provided that it can be handled as a special service by the time-sharing monitor rather than as a full-scale transfer of control from one program to another. Otherwise, the economics of the situation requires a closer look.

Another promising approach to speeding up the entry of information to a computer is that of "menu selection," in which the user employs a light-pen to select a particular label or symbol on the scope. Here again, a service function must be performed by a computer after each selection which the user makes, and now, the service becomes sizeable. Instead of sending a single label down the line in response to a button push, the computer must send a whole set of new labels to the terminal after each label selection with the light-pen.

It appears that graphics terminals in the $5,000–15,000 price range will be available from one source or another within a year or so, using the new 11-inch Tektronix storage tube. Photostatic reproduction devices will probably be available at about the same time at a price of $2,000–3,000.

In a long-term sense, the storage scopes are probably not the answer to computer graphics requirements. Eventually, we might envision a terminal unit cheap enough to be placed in the home, driving the home TV set.

As digital logic costs drop, it would seem that mass-produced terminal units will eventually drop in price below the then-current price of storage scope terminals. However, if that day is to come, it lies some distance in the future. In the meantime, storage-scope terminals may fill the gap and may spur the search for lower-cost TV monitor displays.

VI. Conclusions

The reader may be interested in some of the other systems which resemble AMTRAN in some respects and which differ from it in other ways.

AMTRAN has been influenced by several other systems, particularly by the Culler–Fried, JOSS [5], and BASIC [6] systems. It has been more strongly influenced by JOSS and BASIC (as well as FORTRAN and ALGOL) and less strongly influenced by the Culler–Fried system than might be apparent from a cursory look.

The Culler–Fried system utilizes special keyboards and the 5-inch Tektronix storage scopes. Its language is based upon an operator–operand concept. The Rand Corporation's highly polished JOSS system employs typewriters and is intended to be easy to learn. Darmouth's BASIC is also designed as an easy-to-learn language; and, like JOSS, is being used relatively widely. Columbia's Klerer–May [7] effort is broad, including a programming language, natural mathematical formatting, and automatic numerical analysis. The Lincoln Laboratory's RECKONER [8] was originally designed for matrix routines. It is implemented in a modular and readily expandable way. Harvard's TACT project is vigorously underway but, to the writer's knowledge, is not yet operational [9]. MIT's MAP system is an on-line problem-oriented language for mathematics—i.e., a general-purpose problem-solving system. IBM's APL system is an implementation of a subset of the Iverson programming language [10].

Purdue's NAPSS system is an extensive numerical analytical problem-solving system. MIT's OPS-5 is a multilevel programming language [11]. Aerospace Corporation's POSE system is an outgrowth of the EASL simulation language and is a general-purpose mathematical problem-solving system [12].

Obviously, it is impossible to do justice to these languages in one sentence. There are many differences, as well as overlapping areas among them. Also, the features which we have emphasized may not be their most significant attributes. Finally, the direction and complexion of these development efforts is changing rapidly with time. Perhaps the papers presented at this conference will provide up-to-date references regarding these systems.

ACKNOWLEDGMENTS

The authors wish to acknowledge gratefully the contributions to the 1620 AMTRAN system made by J. Reinfelds and P. L. Clem, Jr., and more recently, by L. A. Flenker, H. M. Glanzer, A. P. Collier, and J. P. Harmon.

APPENDIX

Control operators specific to the interactive mode

1. SUPPRESS/EXECUTE
2. DELETE
3. BACKSPACE
4. EDIT
5. RESET
6. CLEAR
7. LIST
8. LABELS
9. CORE
10. TRACE
11. MOVE PROGRAM
12. FULL TYPEOUT
13. HALT
14. GENERAL INSTRUCTIONS
15. SPECIAL INSTRUCTIONS
16. TURN PAGE

Mathematical operators

1. +
2. −
3. ×
4. /
5. =
6. →
7. $\sqrt{\ }$
8. SQ
9. * (exponentiation)
10. ABS
11. SIN
12. COS
13. ARCTAN
14. TAN
15. EXP
16. LN
17. LOG
18. Σ
19. Δf
20. Δb
21. $\int$
22. d/dx
23. Array
24. Left (Shift)
25. Right (Shift)
26. MINIMUM
27. MAXIMUM
28. MINIMAX
29. MAGNITUDE
30. INTERPOLATE
31. REFLECT
32. ZEROES
33. INVERT
34. LET
35. STEP.FCT
36. CUBIC
37. SUM.OF.SERIES
38. ERF
39. LAPLACE
40. SOLVE
41. AVERAGE
42. SIGMA
43. MOMENTS
44. REGRESSION
45. CORRELATION
46. LEAST. SQUARES
47. (All the TRIG and hyperbolic functions.)
48. REPRESENT

Input-output operators

1. TYPE
2. PRINT
3. PUNCH
4. PLOT.ON.SCOPE
5. TYPE.OUT
6. PRINT.OUT
7. PUNCH.PROGRAM
8. TYPE.ON.SCOPE
9. WRITE.SCOPE
10. ERASE
11. PUNCH.PROGRAM
12. READ.CARDS
13. *
14. @
15. SET
16. X.LOG
17. Y.LOG
18. POINT.PLOT
19. VECTOR
20. HACHURE
21. GRID
22. POLAR
23. DISPLAY.DATA
24. RANGE

Programming and logical operators

1. IF
2. <
3. >
4. THEN
5. OTHERWISE/ELSE
6. AND
7. IF.INDEX
8. SWITCH
9. DIV.ZERO
10. EXIT
11. GO
12. TO
13. REPEAT
14. INPUT
15. ENTRY
16. RUN
17. TRANSFER
18. ACC
19. CALL
20. NAME.THIS
21. =

APPENDIX continued

Special variables and data operators

1. α	8. ψ	15. REG
2. β	9. ξ	16. ′ (prime)
3. γ	10. η	17. INSERT
4. δ	11. X.MIN	18. CONCATENATE
5. ρ	12. X.MAX	19. SORT
6. φ	13. INTERVALS	20. ORDER
7. θ	14. ROW	21. SUB

Basic graphics operators

1. LINE	7. AUTOSCALE	13. YZ.CURVE
2. ARC	8. SYMBOL	14. XZ.CURVE
3. ROTATE	9. THREE.D.MATRIX	15. XY.CIRCLE
4. TRANSLATE	10. THREE.D.AXES	16. YZ.CIRCLE
5. MAGNIFY	11. XYZ.CURVE	17. XZ.CIRCLE
6. AXES	12. XY.CURVE	

Graphic circuit elements

1. RESISTOR	4. GROUND	6. NODE
2. CAPACITOR	5. BATTERY	7. FILTER
3. INDUCTOR		

Special operators

1. PARALLEL	4. DELTA.WYE	6. QUADRATIC
2. SERIES	5. DETERMINANT	7. SIMPSON
3. WYE.DELTA		

Complex routines

1. TIMES	5. COS	9. CONJUGATE
2. OVER	6. ARCTAN	10. POLAR.CONV
3. POWER	7. EXP	
4. SIN	8. LN	

REFERENCES

1. CULLER, C. J., and FRIED, B. D., The TRW Two Station, On-Line Scientific Computer: General Description, *in* "Computer Augmentation of Human Reasoning" (M. A. Sass and W. D. Wilkinson, eds). Spartan Books, Washington, D.C., 1965.
2. RICE, J., and ROSEN, S., NAPSS—A Numerical Analysis Problem Solving System, *Proc. ACM 21st Natl. Conf., 1966*, p. 51. Thompson Press, Washington, D.C., 1966.
3. SASHKIN, A. and SCHLESINGER, S., A Runge–Kutta Integration Procedure with Automatic Interval Control. Rep. No. ATN-64 (59990-1), Aerospace Corp., San Bernardino, California, July 1964.

4. STOTZ, R. H., and CHEEK, T. B., A Low Cost Graphic Display for a Computer Time-Sharing Console, Information Display Symp. Conf. Rep. *Datamation* (July 1967).
5. BAKER, C. L., JOSS: Introduction to a Helpful Assistant, Memorandum RM-5058-PR, The Rand Corp., Santa Monica, California, July 1966.
6. KEMENY, J. G., and KURTZ, T. E., BASIC Users Manual, Dartmouth College Computation Center, Hanover, New Hampshire, January 1966.
7. KLERER, M., and MAY, J., A User Oriented Programming Language. *Comput. J.* **8**, No. 2. (1965).
8. RUYLE, A., BRACKETT, J., and KAPLOW, R., The Status of Systems for On-Line Mathematical Assistance, *20th Anniv. Conf. ACM*, Washington, D.C., August 1967.
9. TACT, Private Communication with Adrian Ruyle, Aiken Computation Laboratory, Harvard Univ., Boston, Massachusetts.
10. IVERSON, K. E., "A Programming Language," Wiley, New York, 1962.
11. GREENBERGER, M., and JONES, M. M., On-Line Simulation in the OPS System, *Proc. ACM 21st Natl. Conf. 1966*. p. 131. Thompson Press, Washington, D.C., 1966.
12. SCHLESINGER, S., and SASHKIN, L., POSE: A Language for Posing Problems to a Computer, *Comm. ACM* **10**, No. 5 (1967).