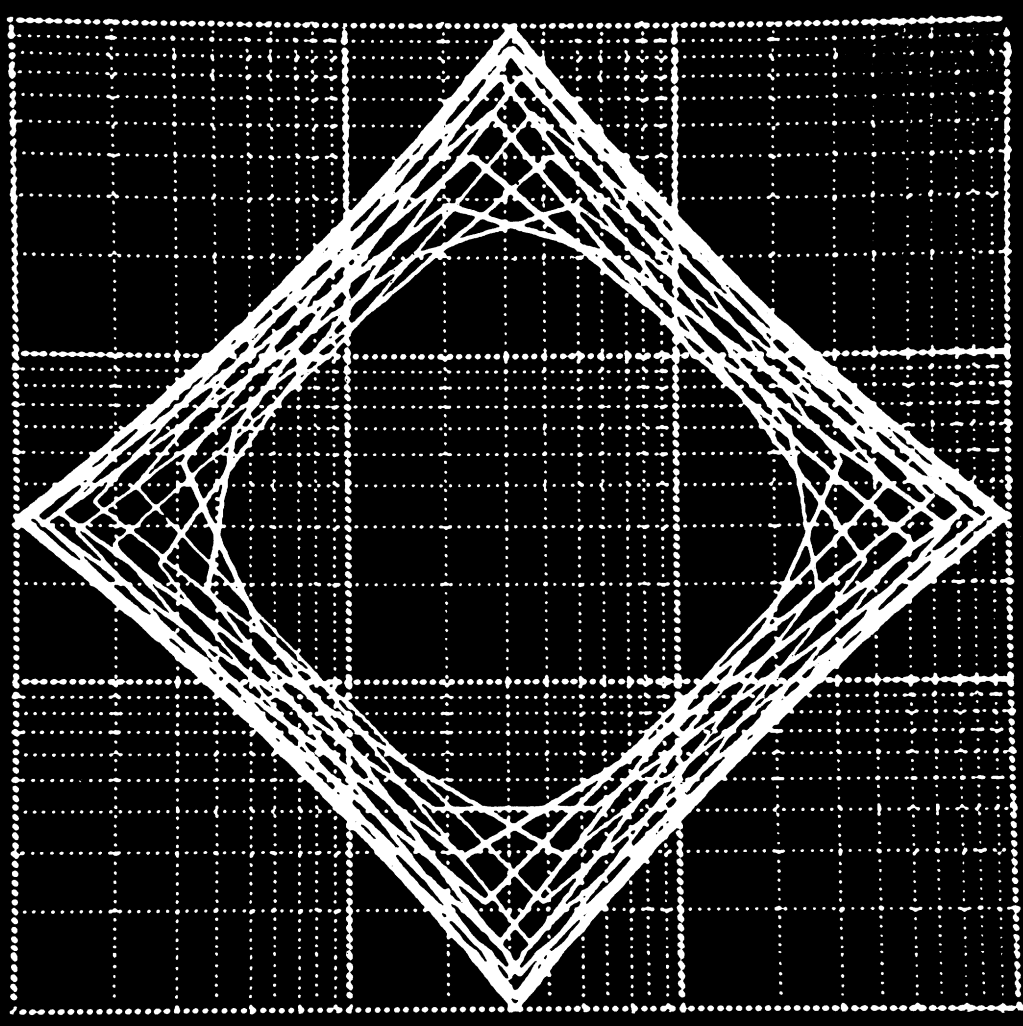
MAP

A System for On-Line Mathematical Analysis

R. Kaplow, S. Strong, J. Brackett

MAC-TR-24



This blank page was inserted to preserve pagination.

MAC-TR-24

MAP

A System for

On-Line Mathematical Analysis

Description of the Language
and
Instruction Manual

Roy Kaplow, Stephen Strong
and John Brackett

Department of Metallurgy
Massachusetts Institute of Technology

January 1966

Copyright © 1966 by
Massachusetts Institute of Technology, Cambridge, Massachusetts

This empty page was substituted for a blank page in the original document.

Foreword

This report relates to a project within the M.I.T. Metallurgy Department, under the supervision of Professors B.L. Averbach and Roy Kaplow. The effort has been greatly assisted by Project MAC, under the direction of Professor Robert Fano, which contributed the major financial support, computer facilities and frequent advice. Mr. Stephen Strong, presently Research Assistant in the Department of Metallurgy , and Dr. John Brackett, Research Associate in the Department of Metallurgy, are responsible for the programming and adaptation of existing programs in the development of the system to its present state.

New developments in any field are usually heavily dependent on previous work. This is especially true in this instance; we have taken advantage of existing programs as well as the entire Time-Sharing System. The present work would not have been possible without that background, which is due to the staffs of the M.I.T Computation Center and Project MAC. In addition, the graphical output commands in MAP rely heavily upon the work of Professor Thomas G. Stockham of the M.I.T. Electrical Engineering Department.

This empty page was substituted for a blank page in the original document.

Table of Contents

Forward..... i

Introduction..... 1

Section A. Scope of the System..... 2

Section B. Language of the System..... 3

Section C. Numbers,Constants and Functions..... 4

Section D. Arithmetic Equations..... 6

Section E. Operational Functions..... 15

Section F. Complex Procedures..... 16

F1. Integration..... 17

F2. Convolution..... 21

F3. Least Square Analysis..... 22

F4. Basis Change (Interpolation)..... 25

F5. Fourier Transform..... 30

F6. Change of Range of a Function..... 32

F7. Manipulation of a Portion of a Function..... 34

Section G. Command Sequences..... 37

Section H. Data Input, Output, Listing and Erasing..... 45

H1. Data Input..... 45

H2. Printed Output..... 46

H3. Graphical Output..... 49

H4. Data Handling Requests..... 60

H5. Correction of Errors..... 63

Section I. The Execute Command and the Use of Auxiliary Programs..... 65

Appendix I Summary of Elements in System MAP..... 69

Appendix II Formats and Off-Line Data Input..... 76

Appendix III Editor for Input Data and Command Sequences..... 78

Appendix IV Programs for the Execute Command..... 83

This empty page was substituted for a blank page in the original document.

Introduction

This manual describes a computer system suitable for use on the time sharing facility at the M.I.T. Computation Center or at Project MAC. Designed for direct computer access through a remote console, the system replaces the normal procedures of programming with a question and answer interchange between the user (hereinafter called U) and the computer (hereinafter called C). The system is intended for the solution of mathematical problems. It should be usable by a person with no knowledge of computers or programming and little knowledge of numerical analysis. Within its range of capabilities, it should be as efficient as are the normal means of computer access for the more sophisticated user.

The system establishes a "conversation" between U and C with an electric typewriter as the means of communication. U can give information to C and can ask it certain questions. C can answer those questions if it is given enough information. C can also ask questions and can therefore request any missing information. In addition, C can explain procedures to U in order to help the latter transmit the required information in a proper form. U, therefore, only needs to know a few basic rules, such as how to phrase his questions and how to name and tabulate his data.

A. The Scope of the System

An ideal computer could understand and answer any question. We must keep in mind, however, that qualities attributed to a computer or a computer system are, in fact, attributes of a total agglomerate of hardware and software. The hardware consists primarily of the electrical, electronic and mechanical devices of the basic computer and various input and output accessories. The software is composed primarily of programs of three types: those which oversee the entire operation; those designed to interpret questions or statements which actually pose questions; and those which "know", or can look up, or can determine how to calculate the answer. Each program consists of a properly planned sequence of simple instructions to be executed in turn by the hardware. In other words, computers must be given procedures for getting answers; accordingly, the ideal must be limited to that class of questions to which at least one person knows either the answer or a procedure for determining the answer. Considerations of economy of manpower and equipment further limit any computer system to types of questions which will prove to be of general usefulness.

The foregoing limits still leave a diversity of classes; of these we consider only those questions the answers to which require primarily mathematical computation. For this class of problems fifty or more general procedures might be required in order to handle 90% of the questions commonly occurring. Far fewer than fifty procedures are now available in the system. The system, however, is open-ended, and additional capabilities can and will be added.

B. The Language of the System

For a majority of users, communication between U and C must pass through a standard IBM 1050 console or type 35 teletypewriter. The console typewriter may be actuated manually by U and "read" electronically by C, or actuated electronically by C and read visually by U. On the 1050 console U's input will appear in lower case type, and C's responses in upper case.

The language used is a combination of English and arithmetic operation symbols. In choosing a language for user-machine communication, it is necessary to strike a balance between a terminology commonly understood and an efficient but cryptic code system. At one extreme is the possibility of using English prose throughout, and at the other the possibility of using a two or three-letter code just complex enough to eliminate ambiguities. The former choice requires a great deal of rather useless translation and elimination of redundancies by C, and a significant amount of typing by both U and C. On the other hand, codes are a nuisance for U to memorize or decipher.

The major requirement, of course, is that the demands on the user should be minimized, in terms of both typing and deciphering effort. We have therefore chosen neither of the extremes, but a combination of modifications of both. U "talks" in one or two-word phrases or in arithmetic equations, while C uses a passable form of English. Conversation between U and C consists primarily of questions and answers, with statements (for informational purposes) and outright commands occasionally admixed. Questions, which, in the present context, are any requests for information, are often phrased in the normal form of commands, because of the simpler grammatical form of the latter.

C. Numbers, Constants and Functions

Mathematical problems are usually stated in terms of variables as in the equation

$$g(x) = a\ h(x)$$

Such an equation normally means the following: at each and every value of the independent variable x, the value of the variable g, which is a function of x, is equal to the value of the constant variable a times the value of the variable h, which is also a function of x. "a" is the name of a constant, the value of which is unspecified. Hence, to distinguish it from a simple number which might appear explicitly in an equation, we call it a "constant variable". For simplicity, however, in the remainder of the text we shall call both such variables and explicit numerical values, "constants". In using the system, U may employ variable names in exactly the same sense as illustrated above. For example, he may refer to the value of a constant by the name "a". Constants, in fact, may be named almost any combination of one to six letters, except those ending in f. Functions are given two names, with the second enclosed in parentheses, just as in normal usage: e.g. y(x). Each of the two names are subject to the same restrictions that apply to names of constants.*

Numbers themselves often occur in the statement of a mathematical problem and are required in any case in order to specify the values of variables. Numbers can be used in three forms: 1) Integers, such as 22; 2) Decimals, such as 19.385; and 3) Exponentials, such as 2.156×10^{-11}.

* At present the only additional restrictions are that the words "minmax" and "input" can not be used as first names for a variable and "loop" and "data" cannot be used as second names for a variable.

Digital computers are discrete state machines, and hence cannot operate on continuous functions. Rather, the specific values of any function, such as y(x), must be specified or calculated at specific values of x, adequately covering the range of interest. If y(x) refers to the values of y, the corresponding values of x may be referred to by another name, such as x(x) , if they will be required for calculations. In many instances, the values of y(x) will be available (or calculable) at equal intervals in x, a form which is advantageous for most numerical procedures. It is then unnecessary to tabulate all the values of x; only the first, the last and the interval between values of x are required. Because of the general simplicity of operation when equal intervals in the independent parameter are used, the system is set up to handle such tabulation automatically, but the more general form is not excluded.

When the values of a function are defined, they will be stored as an array of numbers and identified by the name of the function. Each value in the array will also be associated with a subscript. When U defines a function, he will also be asked to define three additional numbers, which C calls MIN, MAX and DEL. MIN, if given in integer form, specifies the subscript which will be assigned to the first value in the array and MAX the subscript of the last value. Intermediate values will be assigned subscripts MIN+1, MIN+2,...,MAX-1, MAX so the total number of values in the array should be (MAX-MIN+1). In operations which involve more than one function, C will assume that values in the respective arrays which have been assigned identical subscripts have a meaningful correspondence. For example, in an operation which involves the two functions, g(x) and h(x), C will assume that the value in the g(x) array which has been assigned subscript 23 corresponds to the same value of x as the value in the h(x) array with subscript 23.

When U defines an array for which the values correspond to equal intervals in the independent variable, he may specify the interval with DEL. When the values of the independent variable are required, C will then assume that they are the subscripts multiplied by DEL. In such instances, however, U need not concern himself with subscripts at all. In response to the request for MIN and MAX he may type the actual values of x, as <u>decimal numbers</u>, which correspond to the first and last values of the function. The only restriction is that both MIN/DEL and MAX/DEL be integers (within the allowable round-off error of 0.001*DEL). C will automatically calculate appropriate subscripts, such that subscript*DEL = x and will maintain the proper correspondence between different functions.

D. Arithmetic Equations

Since the major purpose of the system is to facilitate the handling of computational problems, it is necessary that a simple method be provided for indicating any necessary arithmetic. Here it is reasonable to deviate from the use of English words, because of the simplicity of arithmetic operation symbols. The following symbols may be used:

+	plus
-	minus
*	multiply by
/	divide by
**	raise to the power
(...)	grouping specification

Using these symbols, U may ask arithmetic questions in the form of equations, with the unknown to the left of the equal sign. Consider the following example, which appears just as it would on the paper of the 1050 console.

Example D.1

```
(v=2.7183**(a*2.*10**(-4)))
COMMAND PLEASE
```

U has asked, "What is the value of 2.7183 raised to the power, 2×10^{-4} a?" He must give the answer a name, since he may want to refer to it later, so he tells C to call the answer "v". The left-hand side of the equation is therefore simply the name of the answer, that is, the name to be given to the value of the expression to the right of the equal sign. The arithmetic expression on the right may include any of the arithmetic operation symbols, the names of any variables (whether defined previously or not) and any of a large number of special operation functions (which will be described later in this section and in Section E). The only restriction is that the expression be mathematically meaningful. The first and last parentheses, which enclose the equation, are required. They indicate to C that the enclosed statement is an arithmetic equation. If the value of the constant a has been defined previously, C can proceed to evaluate the expression and will define v with the resulting value. C will then be finished with the question and will ask for another one by saying "COMMAND PLEASE." Note that C does not print out the answer automatically. The system assumes that the question is just an intermediate step in a larger problem and that U will not be interested in intermediate results which might fill yards of paper (and use up considerable time in being typed). U can

easily obtain printed output, however, as will be seen in Section H.2.

If the value of a had not been defined previously, C could not have answered the question, and the conversation would have proceeded as in the next example.

Example D.2

```
(v=2.7183**(a*2.*10**(-4)))
DECIMAL VALUE OF CONSTANT A PLEASE  3.152e+4
COMMAND PLEASE
```

Whenever C finds that undefined variables have been specified, it simply asks for the values of the variables. In the above example, only one value is missing, that of the constant a, but if many were missing C would request them all, one by one. U, in this instance, has replied "3.152e+4", meaning that the value which should be used for the variable a in this or any subsequent question is 3.152×10^{-4} Notice that in answer to specific questions, U can use "e" as an abbreviation for power of 10 exponentiation. On the other hand an integer may not be used in reply to a request for a decimal value. As soon as U types his reply and gives a carriage return, C has all the required information, finishes the question and asks for another.

The number being exponentiated in the above example is equal to the base of natural logarithms, e. Powers of e are often required in computations, so a special operation function is provided, expf(E), which represents $e^{(E)}$, for any arithmetic expression, E, and may be used as in the following example.

Example D.3

```
(v = expf (a*2*10**(-4)))
COMMAND PLEASE
```

As far as U is concerned, the above sequence is identical to the first example and not much simpler, since the exponentiating operation is also available as an arithmetic symbol. Many more complex functional procedures are also commonly required, however, so a variety of operational functions, like expf, have been made available. Included are those for the sine, cosine, arc sine, absolute value, square root, logarithm and others, which are listed in Section E. These make it unnecessary for U to worry about numerical methods for evaluating common functions and also allow him to write his arithmetic questions in much the same way as he would an analytical equation. The next example further illustrates the use of operational functions.

Example D.4

```
(h = g*sinf(2.*pi*v)/sqrtf(g*v)-logf(absf(cosf(v**2-g))))
DECIMAL VALUE OF CONSTANT PI PLEASE      3.14159
DECIMAL VALUE OF CONSTANT G PLEASE       0.3182
COMMAND PLEASE
```

U has asked C to evaluate the expression:

$$g\sin(2\pi v)/\sqrt{gv}-\ln(|\cos(v^2-g)|).$$

It will be noted that all of the operational functions end in f, which is the reason why "f" must be excluded as a possible final character in the name of a variable. Note the importance of the parentheses, particularly the four which terminate the expression. The first of these indicates the end of the argument of the cosine function, v^2-g; the second the argument of the absolute value function, $\cos(v^2-g)$; the third the argument of the logarithm function, $|\cos(v^2-g)|$; and the fourth mates with the left parenthesis before the h to enclose the entire question. The total number of left parentheses must always equal the number of right parentheses, and C will tell U if they do not. Since the question in this example is rather long, we should point out here that U can only use 72 spaces on a given line of type. If the equation is too long to be specified on one line, U can give a carriage return at any point, other than in the midst of a name. C will then note that the number of left and right parentheses do not correspond and will give U the option of continuing the expression or, if he has made a mistake, of retyping it. C's questions in example D.4 indicate that the constant variables pi and g had not been defined previously, but that v had been, perhaps with a question like that in example D.2. Note once again that U terminates each of his questions and answers with a carriage return.

Thus far we have not shown any examples of arithmetic equations which involve multi-valued variables, or -as we have refered to them earlier- functions. Suppose that y(x) represents an experimental variable which had been measured at equal intervals in an independent variable, x. Say the values of the latter were -3.5, -3.0, -2.5 and so on up to +1.5. Suppose also that z(x) represents a second experimental variable, measured at the same values of x. The following example indicates a computation which involves the two functions, y(x) and z(x), neither of which have been defined previously.

Example D.5

```
(g(x) = y(x)*sinf(z(x)))
Z(X) IS NOT DEFINED.  IF IT HAS A DIFFERENT NAME,
TYPE THAT NAME.  IF YOU WANT TO TYPE IN NUMERICAL
VALUES NOW, TYPE THE WORD INPUT.  OTHERWISE, GIVE
A CARRIAGE RETURN AND DEFINE THE FUNCTION BEFORE
USING THE NAME AGAIN.
input

PLEASE PRINT ON THE NEXT LINE MIN, MAX, AND DEL FOR
THE VARIABLE X.
-3.5  1.5  .5
MIN = -7   MAX = 3
TYPE IN DATA IN ARBITRARY FORMAT, EACH DATA POINT SEPARATED
BY A SPACE.  THE INPUT DATA CAN BE EDITED BY USING
THE CONVENTIONS GIVEN IN THE MANUAL.  WHEN ALL DATA POINTS
HAVE BEEN ENTERED, GIVE TWO CARRIAGE RETURNS.  COMPLETE
EDITING IF NECESSARY, AND GIVE COMMAND 'FILE INPUT DATA'.

INPUT:
5.136  12.175  11.321  15.391  16.784
11.623  11.591
17.211  19.002  13.5672  17.69
EDIT:
file input data

Y(X) IS NOT DEFINED.  IF IT HAS A DIFFERENT NAME, TYPE
THAT NAME.  IF YOU WANT TO TYPE IN NUMERICAL
VALUES NOW, TYPE THE WORD INPUT. OTHERWISE,
GIVE  A CARRIAGE RETURN AND DEFINE THE FUNCTION BEFORE USING
THE NAME AGAIN.
input no
PREVIOUS INPUT DATA IS STILL AVAILABLE.
DO YOU WANT TO EDIT THOSE VALUES.    no
PLEASE PRINT  ON THE NEXT LINE MIN, MAX, AND DEL FOR THE
VARIABLE  X.
-3.5  1.5  .5
MIN = -7   MAX = 3
INPUT:
1.00  1.1  1.29  1.327
1.9  2.73  4.16
5.15
7.936  9.223  14.325
EDIT:
file input data
COMMAND PLEASE
```

In the example above we note, first of all, that whenever a required function has not been defined previously, C will ask U to type in numerical values. Thus a specific input command is unnecessary and is not provided. If, for some reason, the desired values had previously been given a different name, the initial part of C's response would be pertinent and U is thereby given the opportunity to pick up those values while simultaneously changing the name to z(x). The latter facility is of some value, particularly when blocks of data are input through off-line equipment (using a procedure described in Appendix II) in which case they initially carry rather awkward names. The data input procedure should be reasonably obvious from the example, except for the editing facilities which are not illustrated. After U gives the two successive carriage returns to indicate the termination of a data input, C always types "EDIT:". U may then correct any previously unnoticed typing errors, insert omitted values or make virtually any other change in the data, by using a few editing requests which are described in detail in Appendix III.

The values of MIN and MAX are immediately and automatically used to compute subscripts for the function tabulation, as has been previously mentioned. Though U need not concern himself with the subscripts, the first and last subscripts for the array are nontheless printed out and provide a convenient check on the total number of values which C expects, (MAX-MIN+1). The numerical values of the functions may be given in decimal form (as used in this example) or as integers or as exponentiated numbers using the "e" abbreviation.

Two further points about the input procedure: note (after the request for values of y(x)) that by appending the word "no" to the response "input" U informs C that he no longer needs the five lines of instructions which follow the MIN, MAX and DEL input. In addition C gives U one additional opportunity to edit the input data for z(x) before replacing the temporary input data file with the values for y(x).

In example D.5, the two functions involved in the calculation are defined over the same range and with the same interval in x. The answer, g(x), will of course be assigned the same values for MIN, MAX and DEL and will therefore also correspond to x values of -3.5, -3.0, -2.5,... up to +1.5. That is, there is an exact one-to-one correspondence between all the tabulated values. Though this circumstance is expected for most real problems, it is neither assumed nor required by C. C asks for MIN, MAX and DEL values

whenever the values for a function are being provided by U. Thus U may, if he chooses, define different ranges and intervals for different functions, whether or not they are actually functions of the same variable. If an arithmetic equation includes functions which are not defined over the same range of the independent variable then the result will be computed only for that range which is common to all the functions in the arithmetic expression. Therefore, if an equation involves functions of various ranges, the answer function will have a range extending from the largest MIN in the set to the smallest MAX.

If the DEL values of the functions do not agree, C will interpolate as required to use whichever DEL U chooses. Since the grouping of functions with mixed intervals is more likely an indication of error than intent, C will ask U to pick a DEL before doing any calculations. In addition, it should be noted that if U knows that a function will require such an interpolation in more than one arithmetic equation, it will be far more efficient to interpolate and re-tabulate once-and-for-all using the "basis" request which is described later.

The MIN, MAX and DEL values are by themselves sufficient to establish proper correspondence between functions. C therefore does not demand that all functions in an equation have identical second names; such a convention is most likely to be a convenient one for U to use, but he may, in fact, use any name convention which he desires. On the other hand, the concept that the second name is the independent variable is not without meaning. If U writes an arithmetic equation in which the second name of the answer function appears as a variable in the arithmetic expression, that variable will take on the successive values indicated by the values of MIN, MAX and DEL. U need not, therefore, tabulate the values of the independent variable when equal intervals are used. Consider the next example:

Example D.6

```
(tv(x) = x*g(x))
COMMAND PLEASE
```

U has asked, "What are the values of x g(x)?", where we may assume that g(x) has been defined, perhaps as in example D.5. Since x, which appears as a variable in the expression, is the second name of the answer function, tv(x), C will treat it as a true independent variable having values prescribed by the MIN,MAX and DEL values of the function g(x). Referring back to example D.5, we note that g(x) is associated with x values of -3.5, -3.0, -2.5.... to +1.5. Those values are therefore used for x along with the corresponding values

of g(x) in calculating tv(x).

If more than one array appears in the expression, an explicit independent variable will still take on the values prescribed by the ultimate MIN, MAX and DEL of the answer array.

If no functions appear in the expression, the values of the independent variable would be undefined and C would have to request them, as in the next example.

Example D.7

```
(wa(x) = x**3.5)
PLEASE PRINT ON THE NEXT LINE MIN, MAX AND DEL FOR THE VARIABLE
 X.
-12.   12.   1.0
MIN=-12   MAX=12
COMMAND PLEASE
```

U has defined the values of x to be -12., -11., -10., etc., to 10.,11.,12. Whenever C requests MIN, MAX and DEL for the independent variable associated with a specific function, such as y(x) in example D.5, those values refer only to the specific function in question. Other functions with x for the second name may have different ranges and intervals, as has been discussed. In example D.7, the specification is associated specifically with the variable x and will be used as the definition of the values of x only in any similar question where they would otherwise be undefined, that is whenever no arrays appear in the expression. (See examples F5.1 and F5.2, also.)

To summarize: In order to allow U the greatest flexibility, all functions of a given independent variable may be defined over different ranges and with different intervals, and in addition, the independent variable itself may have a different range and interval. In arithmetic equations, the answers will be computed for the region of overlap among those functions which appear in the arithmetic expression even if a different range has been defined separately for the independent variable. The latter definition is used only if functions are not included in the arithmetic expression (or if an entirely new independent variable must be specified as occurs, for example, in a Fourier transform).

Under certain circumstances, the assumption that C makes about equal intervals in the independent variable will not be particularly useful, but neither is it restrictive. Suppose for example, that the x's corresponding to the y(x) and z(x) functions in example D.5 were just 11 arbitrary values, not corresponding to equal intervals, or for that matter that those two sets of numbers could not be

associated with an independent variable at all. It may be seen that example D.5 would nonetheless be acceptable in its exact form, with no changes. MIN, MAX and DEL would have no meaning so far as x were concerned, and it would be more convenient to use the integers 1 and 11 for MIN and MAX, and to leave DEL blank, but the original values would nonetheless give the proper correspondence between y(x) and z(x). Example D.6, however, would have to be different. Since 11 arbitrary values of x have to be specified, U must define an appropriate array, say x(x). The corresponding question would be (tv(x) = g(x)*x(x)), and if x(x) had not been defined previously C would request the values in the manner which has been illustrated. As far as C is concerned, the x(x) array will be just like any other, so that U only needs to be certain that he maintains the correct order and a consistent MIN, MAX specification.

In certain instances the use of integers for MIN and MAX may be convenient even though equal intervals are available. Suppose U desires to compute n(x) = x*m(x) with m(x) available only at x = -1.6, -1.1, -0.6, -0.1, +0.4, +0.9, +1.4. Though the spacing between each point is 0.5 he is restricted by the fact that neither 1.6/.5 nor 1.4/.5 are integers. U may readily obviate the difficulty without actually typing in the separate values of x, as follows:

Example D.8

```
(x(i) = i*0.5-1.6)
PLEASE PRINT ON THE NEXT LINE, MIN MAX AND DEL FOR THE
VARIABLE I.
0   6   1.0

COMMAND PLEASE
(v(i) = x(i) * m(i))
M(I) IS NOT DEFINED, IF IT HAS A DIFFERENT NAME,
TYPE THE NAME.  IF YOU WANT TO TYPE IN NUMERICAL VALUES NOW,
TYPE THE WORD INPUT.  OTHERWISE, GIVE A CARRIAGE RETURN
AND DEFINE THE FUNCTION BEFORE USING THE NAME AGAIN.
input  no

PLEASE PRINT ON THE NEXT LINE MIN, MAX, AND DEL FOR THE
VARIABLE  I.
0  6  1.
INPUT:
1.3   2.45   3.72   4.6   5.3
6.71   7.15
EDIT:
file input data
COMMAND PLEASE
```

Generally speaking, equations which involve only constants in the expression will have a constant as the answer, and those which contain functions will have a function as the answer. These are not rules, however, and exceptions are allowable and sometimes required. In particular, the summation operation yields a single-valued result, so the equation

```
(con = sumf(m(c)))
```

is perfectly legitimate. On the other hand,

```
(con = g(x))
```

is probably not meaningful, but C will define con with the first tabulated value of g(x) (or the first resulting value if the expression is more complicated). The equation

```
(b(x) = con)
```

is meaningful and C will set all values of b(x) equal to the value of con, first requesting if necessary, MIN,MAX and DEL values for x.

Since the arithmetic equations are a "define the values to be..." operation rather than a true statement of equality, equations such as

```
(g(x) = g(x) * y(x))
```

are also meaningful. In the present example the equation means "replace the values of g(x) with the values of the product g(x)*y(x)."

E. Operational Functions

The operational functions available for use in arithmetic equations can be divided into two classes: those which operate upon a single value at a time and those which operate simultaneously on all or part of a stored function.

1. The argument, E, of the following operational functions can be any arithmetic expression.

sinf(E)	sine (radian argument)
cosf(E)	cosine (radian argument)
tanf(E)	tangent (radian argument)
cotf(E)	cotangent (radian argument)
asinf(E)	arc sine (radian result)
acosf(E)	arc cosine (radian result)
atanf(E)	arc tangent (radian result)
sinhf(E)	hyperbolic sine
coshf(E)	hyperbolic cosine
tanhf(E)	hyperbolic tangent
expf(E)	exponential (power of e)
logf(E)	logarithm (base e)
absf(E)	absolute value
sqrtf(E)	square root (positive root)

2. The argument of the following operational functions must be an arithmetic expression which includes at least one function.

sumf(E(x))	summation over all values of E(x), $\sum_{MIN}^{MAX} E(x)$.
intf(E(x))	definite integral over the entire range of E(x).
derif(E(x))	derivative of E(x).

Intf(E(x)) evaluates the integral $\int_{MIN}^{MAX} E(x)dx$, using a third order numerical integration (Simpson's rule). The statement

(g(x)=derif(a(x)))

will obtain the derivatives of a(x) at the same values of x for which a(x) is tabulated and will set g(x) equal to the answers. A five point differentiation is used, except at the end points. Both intf and derif assume that the function being operated upon is tabulated at equal intervals in the independent variable.

F. Complex Procedures

More complex procedures have not been included among the operations possible in an arithmetic expression because a significant amount of subsidiary information is often required. Provision has therefore been made for such procedures in the form of separate questions. The following operations on linear arrays are available.

Integrate	(between fixed or variable limits)
Basis	(F(x) → F(any function of x))
Transform	(sine, cosine, or sine and cosine)
Convolute	(folding of two functions)
Least Square	(least square analysis)
Minmax	(changes the range of definition of a function)
Select	(manipulation of a portion of a function)

The integrate, basis, transform, and convolute procedures are designed to operate on arrays, such as y(x), which are tabulated at equal intervals in x and therefore use the associated values of MIN, MAX and DEL to specify the values of x.

All the procedures are initiated by simply typing the desired procedure name, followed by a carriage return. All require the names and values of a number of parameters, but C will request those whenever necessary. However, U will be able to avoid interrogation by C if he learns which and, in what order, parameter names and/or values should be specified, and specifies them on the same line as the procedure name.

In the following section the individual procedures are described in some detail. In each case an example is shown in which U does not specify any of the required parameters, so that C is required to request that information. Examples are also given which indicate how U may avoid all or part of C's explanations and interrogations by giving some or all of the information when he specifies the procedure.

1. Integration

Example F1.1

```
integrate
INTEGRATE HAS THREE OPTIONS ON THE LIMITS
     1.   BOTH LIMITS FIXED
     2.   LOWER LIMIT FIXED, UPPER LIMIT VARIABLE
     3.   SYMMETRIC INTEGRATION BETWEEN VARIABLE LIMITS
PLEASE INDICATE WHICH OPTION BY TYPING 1,2,OR 3
1
WHAT WOULD YOU LIKE TO INTEGRATE    y(x)
NAME OF ANSWER PLEASE   integ
DECIMAL VALUE OF LOWER LIMIT PLEASE =  1.
DECIMAL VALUE OF UPPER LIMIT PLEASE = 12.
COMMAND PLEASE
```

As C has explicitly stated, there are three options to an integrate question. In all three options the integrals are obtained with a third order numerical integration (Simpson's rule). The first option, illustrated above, is the $\int_a^b y(x)dx$ with a and b both constants. The result will be a single value, so the answer must be given a constant variable name, such as integ in the present illustration. The limits, a and b, must lie within the range of x for which y(x) is (or will be) defined. This condition is apparently satisfied in example F1.1, for C would otherwise have rejected the question, with the comment "THE LIMITS ARE NOT WITHIN THE RANGE FOR WHICH THE FUNCTION IS DEFINED." Notice once again that C does not automatically print out the answers. The "COMMAND PLEASE", however, will indicate that the answer has been obtained.

If the y(x) array had not been defined previously, C would request the values in the same manner that undefined variables are requested for arithmetic questions. If U had intended to define y(x) in an arithmetic statement, he should give a carriage return in response to the question and proceed to define y(x) after the subsequent "COMMAND PLEASE". After defining y(x) in that way, U must again give the command "integrate". The following example illustrates a conversation when U wished to supply numerical values for y(x).

Example F1.2

```
integrate
INTEGRATE HAS THREE OPTIONS ON THE LIMITS
     1.  BOTH LIMITS FIXED
     2.  LOWER LIMIT FIXED, UPPER LIMIT VARIABLE
     3.  SYMMETRIC INTEGRATION BETWEEN VARIABLE LIMITS
PLEASE INDICATE WHICH OPTION BY TYPING 1,2 OR 3
1
WHAT WOULD YOU LIKE TO INTEGRATE  y(x)
NAME OF ANSWER PLEASE   integ
DECIMAL VALUE OF LOWER LIMIT PLEASE=   1.
DECIMAL VALUE OF UPPER LIMIT PLEASE=  12.

Y(X) IS NOT DEFINED.  IF IT HAS A DIFFERENT NAME, TYPE
THE NAME.  IF YOU WANT TO TYPE IN NUMERICAL VALUES NOW,
TYPE THE WORD INPUT.  OTHERWISE, GIVE A CARRIAGE RETURN
AND DEFINE THE FUNCTION BEFORE USING THE NAME AGAIN.
input  no

PLEASE PRINT ON THE NEXT LINE MIN, MAX, AND DEL FOR THE
VARIABLE   X.
1.  12.  .5
MIN = 2   MAX = 24

INPUT:
6. 6.5 7.6 8.2 9.1 10.8 11.5 12.4
15.6  17.8  15.9  14.3  13.2
12.1  13.2  14.5  17.8  19.7  20.2
21.2  23.4  25.6  27.8
EDIT:
file input data
COMMAND PLEASE
```

Once the y(x) values and the MIN, MAX and DEL have been filed, C proceeds to find the integral exactly as in example F1.1. Values for arrays which have not been defined prior to their being specified in any of the complex procedures will always be requested by C in a similar manner, but no further examples of this procedure will be given.

Example F1.3

```
integrate
INTEGRATE HAS THREE OPTIONS ON THE LIMITS
     1.   BOTH LIMITS FIXED
     2.   LOWER LIMIT FIXED, UPPER LIMIT VARIABLE
     3.   SYMMETRIC INTEGRATION BETWEEN VARIABLE LIMITS
PLEASE INDICATE WHICH OPTION BY TYPING 1,2, OR 3
2
WHAT WOULD YOU LIKE TO INTEGRATE   y(x)
NAME OF ANSWER PLEASE    integ(x)
DECIMAL VALUE OF LOWER LIMIT PLEASE =  1.
COMMAND PLEASE
```

The second option is the $\int_a^x y(x)dx$. Integrals will be obtained for values of the upper limit equal to all of those values of x at which y(x) is defined and which are greater than a. The result will therefore be an array and must be given an array variable name, such as integ(x). The DEL and MAX values for the answer array will be the same as that for y(x), but the MIN value will depend on the lower limit, a.

Option 3 is illustrated below.

Example F1.4

```
integrate
INTEGRATE HAS THREE OPTIONS ON THE LIMITS
     1.   BOTH LIMITS FIXED
     2.   LOWER LIMIT FIXED, UPPER LIMIT VARIABLE
     3.   SYMMETRIC INTEGRATION BETWEEN VARIABLE LIMITS
PLEASE INDICATE WHICH OPTION BY TYPING 1,2, OR 3
3
WHAT WOULD YOU LIKE TO INTEGRATE   y(x)
NAME OF ANSWER PLEASE   integ(x)
COMMAND PLEASE
```

The third option is the $\int_{-x}^{x} y(x)dx$, which is meaningful only for arrays which are defined for both positive and negative values of x. Integrals will be obtained for values of the limit equal to all of those values of x for which both y(x) and y(-x) are defined. Therefore, the result will be an array and must be given an array name, such as integ(x). The DEL value for the answer array will be the same as that of the y(x) array. The MIN value will always be zero, and the MAX will be the smaller of the |MIN| and |MAX| of y(x). As an example, if y(x) is tabulated for values of x in the range $-5 \leq x \leq 10$ with an interval of 0.5, the answer array will

contain eleven values, corresponding to values of x in the range 0 to 5 and the same interval, 0.5.

U may specify the option and give all or part of the parameter name information immediately, and thereby avoid all or part of C's responses. Examples F1.5 through F1.8 illustrate various possibilities.

Example F1.5

```
integrate 1 y(x) integ
DECIMAL VALUE OF LOWER LIMIT PLEASE  = 0.0
DECIMAL VALUE OF UPPER LIMIT PLEASE  =10.
COMMAND PLEASE
```

Whenever U gives any information immediately, C will skip the information about the options and ask for the remaining parameters.

Example F1.6

```
integrate 2 y(x) integ(x)
DECIMAL VALUE OF LOWER LIMIT PLEASE = 0.
COMMAND PLEASE
```

Example F1.7

```
integrate 3 y(x) integ(x)
COMMAND PLEASE
```

Example F1.8

```
integrate 1 y(x) integ  0.0  10.0
COMMAND PLEASE
```

2. Convolution

The convolution of two functions is defined as

$$\int_{-\infty}^{+\infty} y\ (\varepsilon)\ q(x-\varepsilon)\,d\varepsilon$$

The example below illustrates a request for such an operation.

Example F2.1

```
convolute
THIS COMMAND PERFORMS CONVOLUTION OF THE EXPRESSION
     A(X)*B(R-X)*DX.
WHAT IS THE NAME OF THE FUNCTION OF THE TYPE A(X)    y(x)
WHAT IS THE NAME OF THE KERNEL FUNCTION   sin(r)
NAME OF ANSWER PLEASE    conv(x)
COMMAND PLEASE
```

The DEL values for the two arrays, y(x) and sin(r), should be the same, though of course there is no need for correspondence in the ranges. However, if the DEL's are not identical, C will ask U which of the two DEL's should be used and will perform the required interpolation. Note that the fact that y(x) and sin(r) have different second names is of no consequence whatsoever.

Once again U may provide all the necessary information immediately, as shown below.

Example F2.2

```
convolute y(x) q(x) conv(x)
COMMAND PLEASE
```

The answer array, conv(x), resulting from the convolution, will have a DEL value identical to that of the two convoluted arrays (or the chosen one, if the DEL's were not equal), but the range in x will depend on the separate ranges of y(x) and q(x) and will always be different from either of those two.

3. LEAST SQUARE ANALYSIS

The least square analysis allows the fitting of data with equations of the form

$$g(x) = z_1 f_1(x) + z_2 f_2(x) + z_3 f_3(x) + z_4 f_4(x) + z_5 f_5(x).$$

Since the data are tabulated at discrete values of x in the computer, for simplicity one may use the alternate notation

$$g_i = \sum_{j=1}^{M} z_j f_{ij},$$

where M is the number of fitting functions, whose values are the f_{ij} and the unknown coefficients are the z_j.

As the data will not generally be perfectly consistent with the assumed fitting functions, the mathematical problem is to determine an approximate set of z_j (z_j') such that

$$\sum_{i=1}^{N} \left| g_i - \sum_{j=1}^{M} z_j' f_{ij} \right|^2 = \text{minimum},$$

where N is the number of points at which the data are defined.

Example F3.1

```
least square
I CAN FIT EQUATIONS OF THE FORM
V(Y) = XA*FA(Y) + XB*FB(Y) + XC*FC(Y) + XD*FD(Y) + XE*FE(Y)
WITH A MAXIMUM OF 5 UNKNOWNS, XA, XB, ETC., AND 100 DATA POINTS.

WHAT IS THE NAME OF THE VARIABLE COMPARABLE TO V(Y).  data(x)
HOW MANY FUNCTIONS,  FA(Y), FB(Y), ETC., WILL BE REQUIRED TO FIT THE
DATA.    4
PLEASE PRINT ON THE NEXT LINE THE NAMES OF THE 4 FUNCTIONS REQUIRED.
a(x) b(x) c(x) d(x)
```

If a(x), b(x), c(x) and d(x) had not been previously defined, numerical values for each of the functions would have been requested. v(y) cannot be defined over a range greater than the smallest of the ranges defining the fitting functions. It is important to note that neither the data nor the fitting functions need be tabulated at equal intervals in the independent variable. However, the values of v(y), fa(y), fb(y), etc. with the same subscripts must have a meaningful correspondence. If no errors are detected in U's responses, the set of simultaneous equations resulting from the least square analysis will be printed, followed by the results of the procedure. A double precision matrix inversion procedure is used to solve the simultaneous equations.

THE EQUATIONS RESULTING FROM THE LEAST SQUARE ANALYSIS ARE

.3133E03 = .9455E02 *XA+.8074E01 *XB+.1806E02 *XC+.3137E02 *XD

.5072E02 = .8074E01 *XA+.5505E01 *XB+.6545E01 *XC+.2993E01 *XD

.9911E02 = .1806E02 *XA+.6545E01 *XB+.1231E02 *XC+.7587E01 *XD

.1123E03 = .3137E02 *XA+.2993E01 +XB+.7587E01 *XC+.1231E02 *XD

LEAST SQUARE SOLUTION. THE FOLLOWING ARE THE 4 UNKNOWNS CORRESPONDING TO XA,XB,ETC.

.2162E 01 .1309E 01 .3399E 01 .1273E 01

ESTIMATES OF THE ERROR IN THESE VALUES ARE

.1426E-04 .1091E-03 .6910E-04 .1344E-03

DO YOU WANT THE VALUES OF THE FITTED CURVE PRINTED. THESE VALUES ARE AVAILABLE FOR FURTHER CALCULATIONS AS THE FUNCTION 'FITTED (X)'. TYPE YES OR NO AND GIVE A CARRIAGE RETURN. yes
THE FOLLOWING ARE THE VALUES OF THE FITTED CURVE AT VALUES OF X FOR WHICH DATA (X) IS TABULATED.

.2583E01 .2947E01 .3346E01 .3772E01 .4212E01 .4656E01
.5091E01 .5505E01 .5885E01 .6223E01 .6511E01 .6745E01
.6926E01 .7054E01 .7134E01 .7173E01 .7179E01 .7160E01
.7123E01 .7076E01 .7024E01 .6974E01 .6929E01 .6984E01
.6870E01 .6861E01 .6868E01 .6893E01 .6937E01 .6999E01

DO YOU WANT THE DIFFERENCES BETWEEN THE FITTED AND ORIGINAL CURVES PRINTED. yes

-.2498E-01 -.6710E-02 .6881E-02 .1526E-01 .1840E-01 .1683E-01
.1164E-01 .4287E-02 -.3814E-02 .3177E-01 .9490E-01 .1398E-01
-.1469E-01 -.1018E-01 -.3814E-02 .3177E-01 .9490E-01 .1398E-01
.1585E-01 .1478E-01 .1096E-01 .5070E-01 -.1876E-02 -.8650E-02
-.1400E-01 -.1682E-01 -.1625E-01 -.1178E-01 .3273E-02 .9052E-02

COMMAND PLEASE

The error estimates, Δz_j, are calculated from

$$\Delta z_j = Q_{jj} \sum_{i=1}^{N} \Delta_i^2 / (N - M),$$

where Q_{jj} is the appropriate diagonal element in the inverse of the coefficient matrix of the simultaneous equations, and the Δ_i are the deviations between the data points and the fit values,

$$\sum_{j=1}^{M} z_j' f_{ij}$$

The error estimates provide only a rough indication of the errors in the calculated coefficients. However, when their values become appreciable in comparison to the calculated coefficients, the validity of the results is questionable.

In this example, data (x) was generated from the equation

$$data(x) = 2.1x + 1.5e^{-.9x} + 3.2e^{-1.1(x-1)^2} + 1.5e^{-.89(x-2)^2}$$

while the fitting functions used were

$$a(x) = x$$

$$b(x) = e^{-x}$$

$$c(x) = e^{-(x-1)^2}$$

$$d(x) = e^{-(x-2)^2}$$

It may be seen that one can readily obtain an excellent fit although the assumed fitting functions are not in fact equivalent to those upon which the data is actually based. In general one must exercise caution in drawing conclusions from multiparameter fits to experimental data.

As usual, U may give all or part of the necessary parameter information immediately. In the following example all of the required information is supplied.

Example F3.2

```
least  data(x)  4  a(x)  b(x)  c(x)  d(x)
```

The name "least" can be used as an abbreviation for the "least square" command. If only a portion of the required information is supplied, the remainder will be requested.

4. Basis Change

It will often occur that an experimental function is readily measured at equal intervals in one independent variable, while the analysis requires equal intervals in a second independent variable which is itself a function of the first. For example, one may measure g(angle) at equal increments in the angle but the analysis of the data may require the transform,

$$g(r) = \int_{k_{MIN}}^{k_{MAX}} i(k)\sin(kr)\,dk$$

where k = A*sin(angle).

Since the transform procedure, which will be discussed in the next section, will assume that the data are tabulated at equal intervals in k, the original tabulation, i(angle), will not do. A new array must be created from the data, by interpolation, using the known relationship between angle and k, i.e. $angle_m = \arcsin(k_m/A)$, where the $angle_m$ are the values of the angle which correspond to equally spaced values of k, k_m. To illustrate, we assume a) that i(angle) contains only 5 points, corresponding to 0°, 22.5° , 45° , 67.5° and 90°; b) that the value of the constant A is 16.8; c) that five values of i(k) are desired, the first at k=0 and the last at k=16.8. The following tables and figures 1 and 2 show the relationship between the angles at which i(angle) is available ($angle_T$) and the corresponding values of k, and the relationship between the desired values of k(k_m) and the corresponding values of the angle.

Independent Variable in i(angle)

$angle_T$	sin(angle)	k_T=16.8sin(angle)
0°	0	0
22.5°	.383	6.43
45°	.707	11.89
67.5°	.924	15.51
90°	1.000	16.80

Independent Variable in i(k)

k_m	k/16.8	$angle_m$=arcsin(k/16.8)
0	0	0
4.2	0.25	14.48°
8.4	0.50	30.00°
12.6	0.75	48.59°
16.8	1.00	90.00°

The procedure for obtaining the required interpolated function, i(k), begins with the definition of the change-of-basis function, angle(k):

Example F4.1

```
angle(k)=asinf (k/16.8)
PLEASE TYPE ON THE NEXT LINE MIN, MAX, AND DEL FOR THE VARIABLE
 K.
0.0   16.8   4.2
MIN=0  MAX=4
COMMAND PLEASE
```

The interpolation is actually accomplished by the basis procedure:

```
basis
THIS COMMAND EVALUATES AN EXPRESSION OF THE FORM
F(R)=G(X(R)), WHERE X(R) MAY BE SPECIFIED BY A
FUNCTION OR BY A NUMBER WHICH REPRESENTS A NEW VALUE
OF DEL TO BE USED IN RETABULATING THE SPECIFIED FUNCTION.
IN WHAT FUNCTION WOULD YOU LIKE TO MAKE A CHANGE OF BASIS.
i(angle)
NAME OF ANSWER PLEASE   i(k)
PLEASE GIVE THE NAME OF THE CHANGE-OF-BASIS FUNCTION OR THE NEW
VALUE OF DEL.     angle(k)
COMMAND PLEASE
```

As is always the case, there is a short form available, so that U may avoid the conversation if he remembers the required order for the parameters.

Figure 1

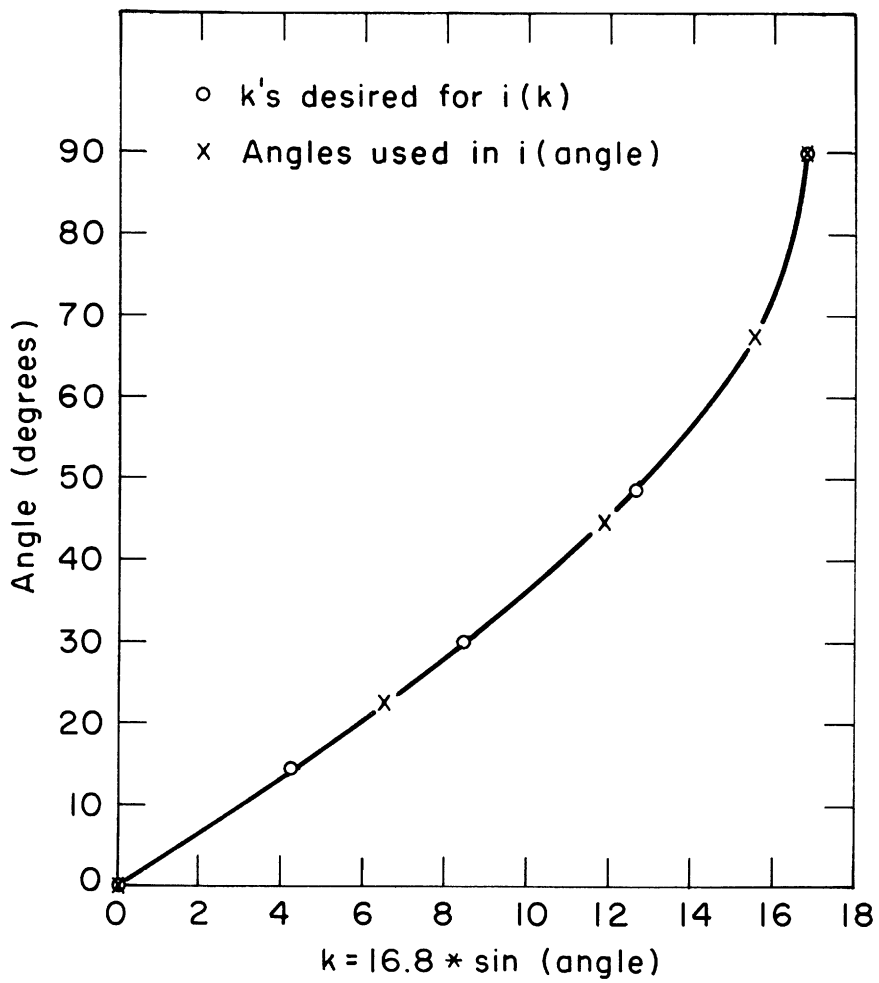
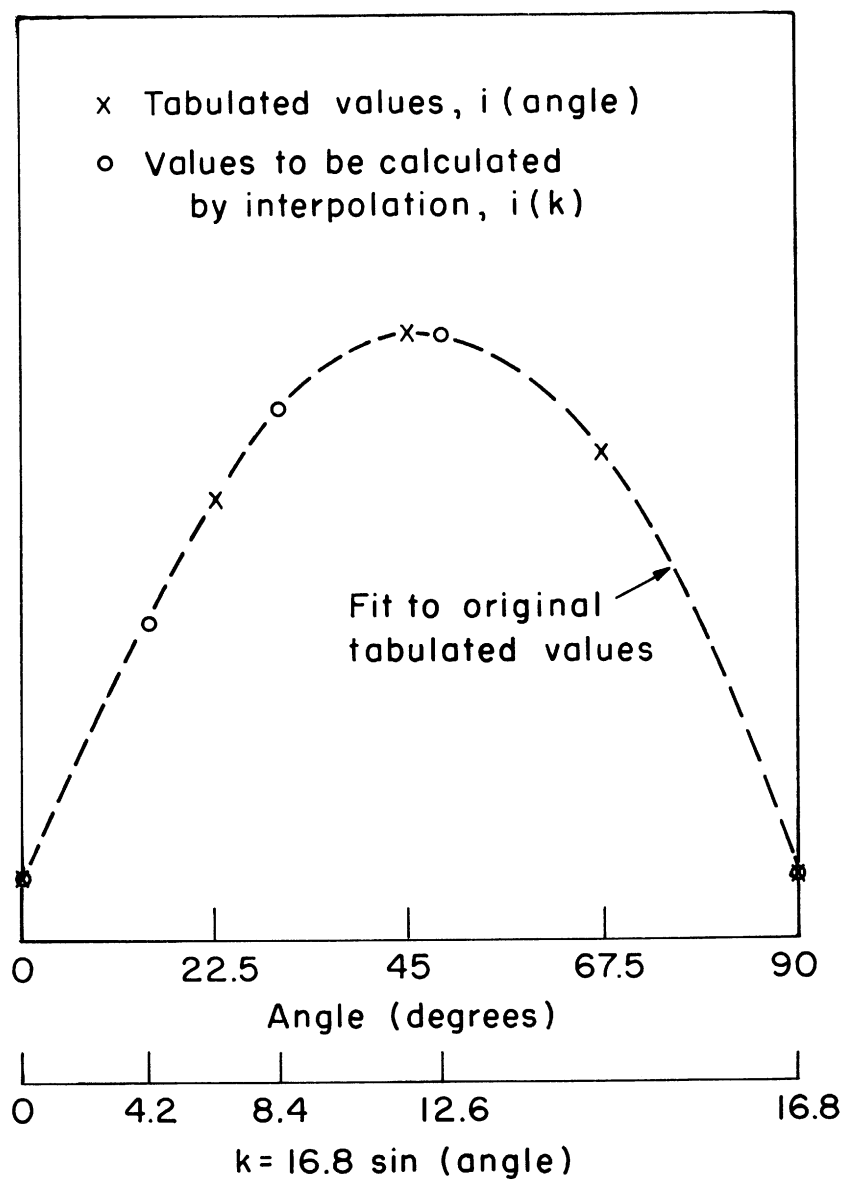

o k's desired for i(k)
x Angles used in i(angle)
Angle (degrees)
0
10
20
30
40
50
60
70
80
90
0
2
4
6
8
10
12
14
16
18
k = 16.8 * sin (angle)

Figure 2


x Tabulated values, i (angle)
o Values to be calculated by interpolation, i (k)
Fit to original tabulated values
0
22.5
45
67.5
90
Angle (degrees)
0
4.2
8.4
12.6
16.8
k = 16.8 sin (angle)

Example F4.2

```
basis  i(angle)  i(k)  angle(k)
COMMAND PLEASE
```

Basis will always use a four point interpolation and will calculate all the values of the new function which lie within the range available in the old function. If values for k had been specified, by its MIN, MAX and DEL, which were outside the range available in i(angle) the corresponding values of i(k) would have been set equal to zero.

It is also important to note that there are no general restrictions on the change-of-basis function; in particular it need not consist of steadily ascending values, but may oscillate or vary in any other way that is meaningful to U, providing in such instances that it does not contain points outside the range of the original function.

The basis procedure can also be used to simply change the interval of tabulation by interpolation. In this case U should type just the new value of DEL instead of a change-of-basis function. That is, to create a new function, newg(x), with DEL = 0.5, from an available one, g(x), which may have been tabulated with any DEL, the request would be:

Example F4.3

```
basis  g(x)  newg(x)  0.5
```

5. Fourier Transformation

The next step in the problem discussed at the beginning of the preceding section is to obtain the Fourier sine transform of i(k). U may obtain his result in the manner illustrated in the following example:

Example F5.1

```
transform
TRANSFORM TAKES SINE AND COSINE TRANSFORMS WITH CONSTANT = 1.0.
WHAT WOULD YOU LIKE TRANSFORMED  i(k)
IF YOU DO NOT WANT ONE OF THE TRANSFORMS, GIVE ONLY A SIMPLE
CARRIAGE RETURN AS THE ANSWER TO REQUESTED NAME.

WHAT WOULD YOU LIKE TO CALL THE SINE TRANSFORM   g(r)
WHAT WOULD YOU LIKE TO CALL THE COSINE TRANSFORM
PLEASE PRINT ON THE NEXT LINE MIN, MAX AND DEL FOR VARIABLE R
0.0   20.0   0.1
MIN = 0   MAX = 200
COMMAND PLEASE
```

U has asked C to evaluate the integral

$$g(r) = \int_{k_{MIN}}^{k_{MAX}} i(k)\sin(kr)\,dk$$

for values of r separated by 0.1, ranging from 0.0 to 20.0.

If both the sine and cosine integrals are requested, the same MIN, MAX and DEL will apply to both answer arrays. While no explicit provision is made in this procedure for complex number notation, the two integrals combined are equivalent to the general complex exponential transform, with the sine integrals associated with $\sqrt{-1}$. The integrals in each case will include the entire range of k for which i(k) is defined. By defining the range and interval in r correctly, the transform procedure may also be used to calculate the integrals required in obtaining the coefficients of a Fourier Series.

As usual, U may give all or part of the necessary parameter information immediately. In the following example the name of the array to be transformed and the name of the answer arrays are specified.

Example F5.2

```
transform y(x)  sin(s)   cos(s)
PLEASE  PRINT ON NEXT LINE MIN, MAX, AND DEL FOR THE VARIABLE S
0.0   10.0    1.0
MIN = 0     MAX = 10
COMMAND PLEASE
```

Note that it is not possible to define the MIN, MAX and DEL for the variable s on the original question line. These are "data" values, which are requested only because they had not been previously defined. The MIN, MAX and DEL specifications for r and s in the preceeding examples are associated specifically with the variables r and s, just as the specifications for x in example D.6 were associated specifically with the variable x. It should be noted that such specification does not preclude the subsequent definition of array variables which have those second names, but which have different ranges and intervals. On the other hand, C will not have to request MIN, MAX, and DEL for r or s when those appear as the second name for the answer arrays in subsequent transform questions, or when they appear by themselves in arithmetic questions. The values of an independent variable may also be defined by a separate procedure, "minmax".

6. Minmax

The "minmax" request may be used to redefine the range over which a function is defined or to define MIN, MAX, and DEL for an independent variable.

Example F6.1

```
minmax
FOR WHAT FUNCTION OR INDEPENDENT VARIABLE WOULD YOU LIKE TO
CHANGE THE RANGE.   r
PLEASE PRINT ON THE NEXT LINE MIN, MAX AND DEL FOR THE
VARIABLE   R.
0.0   10.0   .5
MIN = 0   MAX = 20
COMMAND PLEASE
```

The MIN, MAX and DEL for the independent variable r is now defined, and will be used whenever equations of the type g(r)=r or g(r)=const are to be evaluated or whenever r is the independent variable of transform space in a Fourier transform. The use of the above minmax request will not alter the MIN, MAX, and DEL for any previously defined function of r.

If U desires to change the MIN and MAX for a previously defined function, such as g(x), he could proceed in the following manner, using the abbreviated form of the command:

Example F6.2

```
minmax   g(x)
PLEASE PRINT ON THE NEXT LINE MIN, MAX AND DEL FOR THE
VARIABLE X.
-2.   3.   .5
MIN = -4   MAX = 6
COMMAND PLEASE
```

The request will change the MIN and MAX subscripts of g(x) to -4 and +6 respectively. If MIN is less than the former MIN, or MAX is greater than the former MAX, the additional values created will be equal to zero. If the new MIN and MAX specify a smaller range for the function, the values of the function outside of the new range will be destroyed. The request to change the range of g(x) will have no effect on any other previously defined function, nor will the command change or set the range to be used for x when it appears alone.

U can not use the minmax request to interpolate a function by providing a DEL different from that used in originally defining the function; the basis request should be used to obtain such an interpolation. If a different value of DEL is provided to the minmax request, the value of DEL stored as part of the function will be modified, but the values of the function will not be altered.

Example F5.2

```
transform y(x)  sin(s)   cos(s)
PLEASE  PRINT ON NEXT LINE MIN, MAX, AND DEL FOR THE VARIABLE S
0.0   10.0    1.0
MIN = 0     MAX = 10
COMMAND PLEASE
```

Note that it is not possible to define the MIN, MAX and DEL for the variable s on the original question line. These are "data" values, which are requested only because they had not been previously defined. The MIN, MAX and DEL specifications for r and s in the preceeding examples are associated specifically with the variables r and s, just as the specifications for x in example D.6 were associated specifically with the variable x. It should be noted that such specification does not preclude the subsequent definition of array variables which have those second names, but which have different ranges and intervals. On the other hand, C will not have to request MIN, MAX, and DEL for r or s when those appear as the second name for the answer arrays in subsequent transform questions, or when they appear by themselves in arithmetic questions. The values of an independent variable may also be defined by a separate procedure, "minmax".

7. Manipulation of a Portion of a Function

The request "select" allows U to manipulate a portion of a function or to create a constant equal to a single value of a function.

Example F7.1

```
print  g(x)
MIN = -10   MAX = 7   DEL = .50000

  .2500E02   .2025E02   .1600E02   .1225E02   .9000E01   .6250E01
  .4000E01   .2250E01   .1000E01   .2500E00   .0000E00   .2500E00
  .1000E01   .2250E01   .4000E01   .6250E01   .9000E01   .1225E02
COMMAND PLEASE

select
SELECT WILL CREATE A NEW FUNCTION WHICH WILL BE EQUAL TO A
PREVIOUSLY DEFINED FUNCTION OVER A SPECIFIC RANGE OF THE INDEPENDENT
VARIABLE, OR IT WILL CREATE A CONSTANT EQUAL TO THE VALUE OF A
FUNCTION AT A PARTICULAR VALUE OF THE INDEPENDENT VARIABLE.

FROM WHICH FUNCTION WOULD YOU LIKE TO SELECT THE VALUE(S).  g(x)
WHAT WOULD YOU LIKE TO CALL THE SELECTED  VALUE(S).       p(x)
PLEASE DEFINE THE RANGE OF  X TO BE USED IN CREATING    p(x)    FROM
g(x).   -2.   1.

COMMAND PLEASE
print p(x)
MIN = -10   MAX = 7   DEL = .50000
  .0000E00   .0000E00   .0000E00   .0000E00   .0000E00   .0000E00
  .4000E01   .2250E01   .1000E01   .2500E00   .0000E00   .2500E00
  .1000E01   .0000E00   .0000E00   .0000E00   .0000E00   .0000E00
COMMAND PLEASE
```

In the example, U has requested that the values of p(x) be replaced by the values of g(x) over the specified range in x, -2. to +1. With p(x) previously undefined, as was true in this example, the overall range of p(x) is arbitrarily set equal to the full range of g(x), with zeroes filled into the region outside the range of selection, and g(x)'s DEL is also transferred. If p(x) had been previously defined, the request would be rejected unless both functions had the same DEL; values of p(x) lying outside the range of selection would not be altered. The range of p(x) would be extended, if necessary, to include the range of selection.

As with the other requests, providing all the required parameters on the command line eliminates C's explanatory notes.

Example F7.2

```
select  g(x) p(x)  -2. 1.
COMMAND PLEASE
```

It is possible, as with most of the commands, to give the derived function the same name as the primary function. Select treats such an occurance as a special case.

Example F7.3

```
select  w(x) w(x)  -1.0 1.5
COMMAND PLEASE
```

The result, in this instance, is simply that any previously defined values of w(x) which lie outside the range $-1.0 \leq x \leq 1.5$ are set to zero, but the range of the function is not changed.

An important use of the select command is to generate a composite function, the various sections of which are drawn from different functions. Suppose, for example, that: v(x) =y(x),

$|-5. \leq x \leq -2.5|$; v(x) = y(x) + g(x), $|-2. \leq x \leq 1.|$; and
v(x) = g(x), $|1.5 \leq x \leq 6.|$.

The following example will accomplish the formation of v(x), assuming that g(x) had previously been defined over the range $-2. \leq x \leq 6.$, y(x) at least over the range $-5. \leq x \leq 1.$ and that both functions have a DEL of .5. p(x) is used only as an intermediate function in the example, and is assumed not to have been defined previously.

Example F7.4

```
select     g(x)     p(x)     -2.0     1.0

COMMAND PLEASE
select     y(x)     v(x)     -5.0     1.0

COMMAND PLEASE
select     g(x)     v(x)      1.5     6.0

COMMAND PLEASE
minmax     p(x)
PLEASE TYPE ON THE NEXT LINE MIN, MAX, and DEL FOR THE VARIABLE X.
-5. 6. .5
MIN = -10   MAX = 12

COMMAND PLEASE
(v(x) = v(x) + p(x))

COMMAND PLEASE
```

In those instances in which U is working with tabulations of values which do not correspond to equal intervals in an independent variable, he will probably have used integers to define the MIN and MAX of the functions. The use of integers, in specifying the desired range for a select request, tells C to pick out those values carrying the prescribed subscripts. Though the values of DEL are still checked to ensure consistency, they are then not further utilized and can be equal to zero. In any case, whether values of the independent variable or subscripts are used to specify the range of selection, it is not necessary that the second names of the primary and derived functions be the same.

The following example illustrates how a constant, equal to a particular value of a function, may be created.

Example F7.5

```
select  g(x)
WHAT WOULD YOU LIKE TO CALL THE SELECTED VALUE(S).  a
TO WHAT VALUE OF X SHOULD A CORRESPOND.  2.3

THE VALUE OF X AT WHICH A IS TO BE DEFINED DOES NOT CORRESPOND TO  A
DATA POINT.  LINEAR INTERPOLATION WILL BE PERFORMED USING THE
ADJACENT DATA POINTS.

COMMAND PLEASE

print a
     A =     .5300E01
```

Notice that C will interpolate, if necessary, to estimate the value of the function at the desired value of the independent variable. Such an interpolation is meaningful only if the function is tabulated at equal intervals. Of course, if the chosen value of the independent variable corresponds precisely to a tabulated value or if an integer is used to specify a subscript, no interpolation will be necessary.

G. SPECIFICATION AND EXECUTION OF COMMAND SEQUENCES

For certain problems it may be necessary to repeat a particular sequence of MAP commands, perhaps varying certain of the parameters. In other instances U may find a certain sequence of commands to be particularly suited to his work and therefore recurring often in many sessions at the console. In order to relieve U of repetitive typing, a mechanism is provided for setting up and saving a sequence of commands, which may subsequently be executed as often as required.

The command "create" allows a sequence of 18 or fewer lines of MAP statements to be defined and stored. Whenever "create" is typed, C will reply with a message explaining the command and will then type "INPUT:". U may then type as many successive statements as are required for the sequence. The commands are <u>not</u> executed after each carriage return. When U gives two successive carriage returns, C will type "EDIT:". Using the procedures described in Appendix III, U may then correct or alter the sequence. When the command sequence is complete and edited, the request "file name" should be given, where "name" is a name of 6 or fewer characters used to designate that particular command sequence. Subsequent use of an associated request, "run" or "run name" will cause C to execute the designated sequence, each command being handled as if it had just then been typed by U in its precise form.

As an illustration consider the following problem. U would like to evaluate the integral

$$\int_u^v z(x)\,dx, \text{ with } \quad z(x) = [\exp(-a*y(x)) + q] \quad \text{and } y(x) = x^2$$

for a variety of values of the constants, a and q, and various limits, u and v, on the integral.

Example H.1

```
(y(x)=x**2)
PLEASE PRINT ON THE NEXT LINE MIN, MAX, AND DEL FOR THE
VARIABLE X.
0.  10.  1.
MIN =     0      MAX =  10

COMMAND PLEASE
create
TYPE IN COMMANDS, ONE PER LINE.  WHEN ALL COMMANDS HAVE BEEN
ENTERED, GIVE TWO CARRIAGE RETURNS, EDIT IF NECESSARY
AND GIVE COMMAND 'FILE XXXXXX' (WHERE XXXXXX IS A NAME
OF 6 OR FEWER CHARACTERS BY WHICH YOU CAN IDENTIFY YOUR
COMMAND SEQUENCE).  THE COMMAND SEQUENCE CAN BE EDITED
AND PRINTED BY USING THE CONVENTIONS GIVEN IN THE MANUAL.

INPUT:
(z(x) =expf(-a*y(x)) +q)
integrate 1 z(x) ann
print ans
delete q const a

EDIT:
top
verify
locate ann
INTEGRATE 1 Z(X) ANN
change /ann/ans/
INTEGRATE 1 Z(X) ANS
file param
COMMAND PLEASE
```

Notice that U has defined y(x) outside of the command sequence, so that C will not be required to recalculate it unnecessarily during each repetition of the sequence. In order that the limits on the integral, u and v, be treated as variable parameters of the sequence, U has not specified the integration limits in the definition of the command sequence so that C will request values for them at each repetition. The last command in the sequence is a request that C "forget" the values of the constants a and q so that it will request new values each time the sequence is executed (this command will be discussed along with the other data handling commands in Section H). The example also shows the use of the editing facility which is automatically available after U signals the end of his input with two successive carriage returns. The editing requests required to correct errors in the input file are discussed in detail in Appendix III. The last request given, "file

param", is a request to store the corrected command sequence under the name param. It will then be available for subsequent usage.

At this point, if U were ready to execute the sequence, he would give the command "run param". C will type the commands as they are executed and will request any missing information.

```
COMMAND PLEASE
run
WHAT LOOP WOULD YOU LIKE TO EXECUTE          param
THE COMMAND BEING EXECUTED IS
(Z(X) = EXPF(-A*Y(X))+Q)
DEC. VALUE OF CONSTANT Q PLEASE              10.
DEC. VALUE OF CONSTANT A PLEASE               1.

THE COMMAND BEING EXECUTED IS
INTEGRATE 1 Z(X) ANS
DECIMAL VALUE OF LOWER LIMIT PLEASE =         0.
DECIMAL VALUE OF UPPER LIMIT PLEASE =        10.

THE COMMAND BEING EXECUTED IS
PRINT ANS
   ANS =     .10084E 03

THE COMMAND BEING EXECUTED IS
DELETE Q CONST A

COMMAND PLEASE
run param

THE COMMAND BEING EXECUTED IS
(Z(X) = EXPF(-A*Y(X))+Q)
DEC. VALUE OF CONSTANT Q PLEASE              20.
DEC. VALUE OF CONSTANT A PLEASE               1.

THE COMMAND BEING EXECUTED IS
INTEGRATE 1 Z(X) ANS
DEC. VALUE OF LOWER LIMIT PLEASE =            0.
DEC. VALUE OF UPPER LIMIT PLEASE =           10.

THE COMMAND BEING EXECUTED IS
PRINT ANS
   ANS =      .20084E 03

THE COMMAND BEING EXECUTED IS
DELETE Q CONST  A
```

The example shows just two repetitions of the sequence though, of course, U could try as many variations of the parameters as he desired. In addition he might interpose whatever other calculations he desired between repetitions.

If the results obtained were not satisfactory to U, he could change the sequence by typing the MAP command "edit param". C would respond by typing "EDIT:" to indicate it's readiness to accept editing requests. U could then give requests to correct or to make alterations. After making all the necessary changes, U would again request "file param" and again could execute the sequence by using the "run" command in order to determine if his corrections had been sufficient. If the command sequence had involved many commands rather than only three, and had an error been found in the tenth line of the sequence, U might not need to repeat the entire command sequence, but might need only to repeat the sequence beginning with the first erroneous command. For example in this case he could use the command "run param 10" to start execution at the tenth line of the sequence. The "run" command will assume that the sequence is to be executed from the beginning unless a line number is provided after the name of the sequence.

The use of a command sequence is obviously useful to study the effects of variations of the parameters, but it also has applications in other areas, such as iterative solutions. For example, suppose U would like to solve the integral equation,

$$U(r) = \phi(r) + c \int_0^r U(r)dr.$$

From the physical nature of the problem he has concluded that ϕ (r) would be a good initial approximation to U(r). One procedure for solving the equation iteratively is illustrated in the following example. Before defining the command sequence U printed ϕ (r) simply to record the initial approximation to U(r). Notice also that by following the "create" request with the word "no", U avoids the descriptive message.

Example G.2

```
     print phi(r)
     MIN = 0  MAX = 29  DEL = .50000

  .0000E 00  .9996E 00  .1997E 01  .2989E 01  .3975E 01  .4948E 01
  .5910E 01  .6857E 01  .7787E 01  .8696E 01  .9583E 01  .1045E 02
  .1128E 02  .1208E 02  .1286E 02  .1359E 02  .1429E 02  .1495E 02
  .1557E 02  .1614E 02  .1667E 02  .1714E 02  .1756E 02  .1793E 02
  .1824E 02  .1849E 02  .1868E 02  .1880E 02  .1885E 02  .1884E 02
     COMMAND PLEASE
     (u(r) = phi(r))

     COMMAND PLEASE
     create    no

     INPUT:
     (q(r) =c*r*u(r))
     integrate 2 q(r) ans(r)  0.
     (u(r) =phi(r) +ans(r))
     print u(r)

     EDIT:
     file intequ

     COMMAND PLEASE
```

```
run intequ

THE COMMAND BEING EXECUTED IS
(Q(R)=C*R*U(R))
DEC. VALUE OF CONSTANT      C PLEASE    .005

THE COMMAND BEING EXECUTED IS
INTEGRATE 2 Q(R) ANS(R)  0.

THE COMMAND BEING EXECUTED IS
(U(R)=PHI(R)+ANS(R))

THE COMMAND BEING EXECUTED IS
PRINT U(R)
MIN = 0   MAX = 29   DEL = .50000

.0000E00  .1000E01  .2000E01  .3000E01  .4000E01  .5000E01
.5999E01  .6998E01  .7997E01  .8994E01  .9990E01  .1098E02
.1197E02  .1296E02  .1394E02  .1492E02  .1589E02  .1685E02
.1780E02  .1874E02  .1967E01  .2075E02  .2146E02  .2233E02
.2317E02  .2398E02  .2476E02  .2551E02  .2621E02  .2686E02

COMMAND PLEASE
```

```
run intequ

THE COMMAND BEING EXECUTED IS
(Q(R) =C*R*U(R))

THE COMMAND BEING EXECUTED IS
INTEGRATE 2 Q(R) ANS(R)  0.

THE COMMAND BEING EXECUTED IS
(U(R)=PHI(R)+ANS(R))

THE COMMAND BEING EXECUTED IS
PRINT U(R)
MIN = 0   MAX = 29   DEL = .50000

.0000E 00  .1000E 01  .2000E 01  .3000E 01  .4000E 01  .5000E 01
.6000E 01  .7000E 01  .8000E 01  .9000E 01 1.0000E 01  .1100E 02
.1200E 02  .1300E 02  .1400E 02  .1500E 02  .1600E 02  .1699E 02
.1799E 02  .1898E 02  .1998E 02  .2097E 02  .2195E 02  .2294E 02
.2391E 02  .2489E 02  .2585E 02  .2681E 02  .2775E 02  .2868E 02
COMMAND PLEASE


run intequ

THE COMMAND BEING EXECUTED IS
(Q(R) =C*R*U(R))

THE COMMAND BEING EXECUTED IS
INTEGRATE 2 Q(R) ANS(R)  0.

THE COMMAND BEING EXECUTED IS
(U(R)=PHI (R) +ANS(R))

THE COMMAND BEING EXECUTED IS
PRINT U(R)
MIN = 0   MAX = 29   DEL = .50000

.0000E 00  .1000E 01  .2000E 01  .3000E 01  .4000E 01  .5000E 01
.6000E 01  .7000E 01  .8000E 01  .9000E 01  .1000E 02  .1100E 02
.1200E 02  .1300E 02  .1400E 02  .1500E 02  .1600E 02  .1700E 02
.1800E 02  .1900E 02  .2000E 02  .2100E 02  .2200E 02  .2300E 02
.2399E 02  .2499E 02  .2599E 02  .2698E 02  .2797E 02  .2896E 02

COMMAND PLEASE
```

Each time the sequence "intequ" is executed U can compare the values of U(r) thus obtained with those obtained from the previous iterative cycles. After the third cycle the changes in the values are sufficiently small that he would probably not request further

iterations.

The "create" and "run" commands allow U to create a particular structure of MAP statements pertinent to his problem. Since command sequences can themselves contain "run" commands, an increasingly elaborate structure of commands may be referred to by a single name. The complexity of such structures is limited only by the restriction that sequences may not be nested more than three deep; that is, a sequence, loop1, may call another, loop2, which may call a third, loop3, but the third sequence may not contain a "run" command to call a fourth.

For problems which require more than three levels of sequences, or for operations not yet available as MAP commands or for sequences which require long or numerous arithmetic equations and which will be used often, simplified MAD programs which can be used in conjunction with the MAP command "execute", are recommended. The "execute" command is briefly discussed in section I and instructions are given in Appendix IV for writing MAD programs to be used by MAP. In addition to providing a more efficient mechanism for repetitive evaluation of arithmetic equations, this procedure allows MAP to be extended indefinitely in whatever directions are desired by a particular user.

H. Data Input, Output, Listing and Erasing

1. Data Input

Since C will specifically request values to be defined whenever an undefined parameter is used in a question, there is no need for a separate mechanism for data input, and none is provided within the language of the system. However, provision has been made for off-line loading of large blocks of data (from tapes or punched cards) directly into the system storage. Data which is entered in this way, however, is initially identified by a special name which includes the numerical form (i.e., format) of the data. It is this fact which gives rise to the "other name" question in C's response to finding an undefined array. It is expected that the off-line process of data input will not be commonly used, but the exact procedure to be followed is described in Appendix II.

2. Printed Output

In all of the examples of questions, it was noted that C does not automatically print out the results. However, U may obtain printed results at any time simply by using the procedure name "print".

Example H2.1

```
print
WHAT WOULD YOU LIKE PRINTED    a
A = 3.152
COMMAND PLEASE
```

Here, U has asked the question, "What is (are) the value (values) of .....?" C understands the question, but cannot proceed because it does not know the name of the parameters desired. It therefore asks for the missing information. Apparently the value of a has been previously defined, since C then responds by printing out the value, 3.152. If a had not been defined, C would, as usual, request the value. U may avoid C's interrogation by specifying the desired variable immediately.

Example H2.2

```
print integ
INTEG = 0.50000E 02
COMMAND PLEASE
```

The procedure is identical for array variables.

Example H2.3

```
print integ(x)
MIN=0    MAX=9    DEL=2.0000
0.2000E 01  0.8000E 01  0.1800E 02  0.3200E 02  0.5000E 02
0.7200E 02  0.9800E 02  0.1280E 03  0.1620E 03  0.2000E 02
COMMAND PLEASE
```

Notice that C will print the MIN and MAX (as subscripts) and DEL values for the array, as well as the values of the array itself. In this instance the MIN, MAX, DEL specification shows that there are ten values of integ(x) tabulated, corresponding to values of x equal to 0.0, 2.0, 4.0 and so on up to 18.0.

In response to a print request which refers to an array, C will normally use the format indicated in the example, that is, a four significant figure exponentiated number, six values to a line. If this format is not satisfactory to U, he may change the form by specifying an alternative format with the original "print". For example,

print y(x) 5F12.7

The format, 5F12.7, is a request for 5 decimal numbers to a line, each number allotted twelve spaces with seven significant figures to be given after the decimal point. Examples of other possible format specifications are given in Appendix II.

The "print" question is an excellent one to use for data input, since U may immediately compare his input data list with C's output to ensure that neither has made an error.

The print request also permits U to print a function over a limited range, rather than over the entire range for which the function is defined. The interval at which the function is to be printed can be different from the DEL at which the function is tabulated. If the desired interval for printing is not a multiple of DEL, interpolation will be performed. The additional features of the print command require that U give all of the necessary information on the same line as 'print'; if such information is not provided, C will assume the entire function is to be printed using the interval at which the function is defined. The following examples will illustrate the use of the additional features:

a. print y(x) .5

y(x) will be printed at intervals in x of .5, rather than with whatever interval for which y(x) may have been defined. Four point interpolation will be performed if necessary.

b. print y(x) 2. 4.

y(x) will be printed for values of x between x=2. and x=4. using the interval at which y(x) is tabulated. Integers may be used to specify the range to be printed; C will print the values of y(x) having subscripts between the two values given in the request. The restrictions upon the use of decimal values in defining the MIN and MAX of a function applies to this request; i.e. the limits of the range must be integers or differ from integers by .001*DEL or less. If this condition is not satisfied, C will not print the function over exactly the range expected. The range for which the function is to be printed must be within the range for which the function is defined.

c. print y(x) 5F12.7 2. 4. .05

y(x) will be printed for values of x between x=2. and x=4. at intervals of x=.05. As in b., integers may be used to specify the range of y(x) to be printed. If a format specification is included

in the print request, the format must immediately follow the name of the function to be printed.

3. Graphical Output

U may obtain a graphical presentation of his results by means of the "plot" or "compare" requests if he has access to the required display equipment. MAP can be used to generate graphs on either the ESL display console or on storage oscilloscopes. Unfortunately, limitations in the present data transmission facilities make it impossible to provide graphical output terminals at remote stations, so the requisite display units are available only at the Project MAC facilities at 545 Technology Square. Until graphical display facilities become available at the M.I.T. Computation Center, the "plot" and "compare" requests will not be recognized by the version of the MAP system at the Center.

The two types of available terminals differ in both the quality of the display and the amount of information that can be presented at one time. The ESL display console uses a cathode-ray tube on which the picture is redrawn several times a second, the rate depending upon the complexity of the picture being displayed. The more information to be presented, the less frequently each part of the picture can be redrawn. In many cases the picture will flicker or even fade away for an instant, before being restored. However, the display is both sharp and bright, and graphs of high quality can be obtained. The time sharing system limits the amount of information that can be displayed at one time on the ESL unit, since a portion of the memory of the computer is used to store the data being drawn. Currently U is permitted to plot graphs that include not more than a total of about 150-200 data points. In order to reduce the flickering and to allow more than one function to be plotted on a single graph, all functions should be plotted using the largest interval that will give a meaningful representation when the data points are connected by straight lines.

In contrast to the ESL console the picture on the storage oscilloscope need be drawn only once by the computer. The storage tube will maintain a good quality picture for at least 10 minutes, at which time the picture can be regenerated, if desired, with another "plot" request. The screen on this oscilloscope is less than one quarter the size of the ESL display (about the size of a postcard) and the amount of detail that can be resolved is correspondingly less. On the other hand, any amount of information can be presented on the screen without introducing the flicker which is characteristic of the ESL console.

In addition to providing on-the-spot display of data, both units can also be used to obtain photographic copies for reference or presentation at a later time. At present there is no provision in the time sharing system for easily obtaining hard copy graphs other than by photography.

MAP will use either type of display device to produce graphs consisting of scaled graph paper and the desired functions. C examines the range of both the dependent and independent variables and creates a graph paper with the appropriate scales. Either scale can be linear or logarithmic, and the axes will be labelled with convenient values. Unless a point plot is requested, the data points will be connected by straight lines, which will create a good visual approximation to the function if the data points are sufficiently closely spaced.

Example H3.1

```
plot
PLOT WILL CREATE A GRAPH OF THE DESIRED FUNCTION(S).

WHAT FUNCTIONS WOULD YOU LIKE TO PLOT.  g(x)  a(x)
SHOULD THE PLOT BE LINEAR, LOG-LOG, LINEAR-LOG, or LOG-LINEAR.
log-linear
DO YOU WANT A POINT OR LINE PLOT OF G(X).  point
DO YOU WANT A POINT OR LINE PLOT OF A(X).  line

IF YOU DO NOT WANT ALL OF THE POINTS OF THE FUNCTION(S)
PLOTTED, TYPE THE RANGE AND/OR INTERVAL IN X TO BE USED.
OTHERWISE JUST GIVE A CARRIAGE RETURN.  2.  100.  2.
COMMAND PLEASE
```

By typing only "plot", U has requested the use of the ESL display console. In order to use the storage oscilloscope, the equivalent command is "plot storage". He has requested that two functions be plotted, g(x) and a(x); a maximum of three functions may be called for simultaneously. U has indicated, by typing "log-linear", that he wants to plot the log of g(x) and the log of a(x) versus the independent variable, x. The "point" and "line" responses request that only the data points of g(x) should be displayed, but that the plot of a(x) should be constructed by connecting the points defining a(x) with straight lines. In order for the two functions to be plotted on the same scaled graph paper it is necessary that the DEL's of g(x) and a(x) be equal. Obviously, if the g(x) and a(x) tabulations do not correspond to equal intervals in the independent variable, the values of x which C generates from MIN, MAX and DEL are meaningless. In such instances, U must tabulate the values of the independent variable in a function such as x(x), and use the

command "compare", which is described in the following section, in order to generate a meaningful graph.

The values of g(x) and a(x) will be examined to determine the largest and smallest values which are required for the vertical axis, in order to properly generate and scale the graph paper. Since the logarithm of a negative number or zero is not defined, negative or zero values for g(x) and a(x) will cause an error comment followed by "COMMAND PLEASE" to be typed. U has also indicated a range and interval to be used in plotting the graph, which is presumably different, at least in part, from that used in the original definition of g(x) and/or a(x). In the case of the ESL console the interval should be as large as is consistent with the use of linear segments between the data points. The range can also be provided as integers, in which case C will take them to be the subscripts of the tabulated values.

On the ESL console there may not be sufficient buffer space available in the main computer memory area to store the information required to plot the graph paper and the desired functions. Therefore, if U does not indicate a sufficiently large interval, the data will be sampled automatically in order to plot the graph. If this sampled data produces a poor representation of the functions, U should not attempt to plot as many functions on the same graph, or should use the storage oscilloscope.

A short form of the plot command is available to allow U to avoid C's interrogation. C will make several assumptions about the type of graph desired unless U provides information to the contrary. The short form of any MAP command can not consist of more than 72 characters or 13 parameter names, a function name such as z(x) being counted as 2 names. Therefore if three functions are to be plotted, it is not possible to specify all of the possible options on one line as is required. In this regard it should be recalled that a change in the range and/or interval could be accomplished by using the minmax and/or basis commands prior to the plot request. Unless information is supplied to the contrary, C will make the following assumptions in generating the graph:

a. The ESL display console should be used.
b. Each axis should be linear.
c. A line plot should be made of each function.
d. The range of the graph should be sufficient to include the entire range of all the functions.
e. All the tabulated data should be used in the plot unless (with the ESL console) the available buffer space

is exceeded; in this case the data will be sampled as required.

Example H3.2 (See Figure 3)

```
plot h(x)
COMMAND PLEASE
```

C will plot h(x) on the ESL console on graph paper with linear scales. Only the origin of the graph is labelled with complete numerical values, and these labels are of the "E" format for decimal numbers. In this example, the value of the origin on the dependent axis is $-1. \times 10^{12}$ and the value on the independent axis is 0. The other labels on an axis omit the power of ten given at the origin for that axis. For example, the main divisions on the dependent axis correspond to: $-.8 \times 10^{12}$, $-.6 \times 10^{12}$,..., 1×10^{12}.

Example H3.3

```
plot  g(x)  h(x)  3.2   6.8
DUE TO THE LENGTH OF THE ESL DISPLAY BUFFER,
THE DATA HAVE BEEN SAMPLED IN ORDER TO
PRODUCE THE PLOT.
COMMAND PLEASE
```

U has requested that g(x) and h(x) be plotted simultaneously over the range x = 3.2 to x = 6.8. The graph is shown in Figure 4 and it can be noted that the axes have been rescaled. The use of fewer points in creating the plot is particularly evident in the regions of the curve where the slope is rapidly changing, since in generating the graph to be photographed the sampling procedure was simply to omit every other data point.

Example H3.4

```
plot storage g(x) h(x) 3.2 6.8 lines points
COMMAND PLEASE
```

C will plot the graph shown in figure 4, with two exceptions: only the points of h(x) will be displayed and the graph will be plotted on the storage oscilloscope. The first occurence of the word "lines" or "points" refers to the first function named to be plotted. In this case U did not intend g(x) to be point plotted so he had to use the form in this request. If g(x) was to have been a point plot and h(x) drawn with lines only, the single word "points" would have been required; "lines" would be assumed for h(x). The words "lines" and "points" may occur at any point in the sequence of parameter names.

Logarithmic scales can be used on either or both of the axes. Figure 5 illustrates the four possible ways of plotting a function

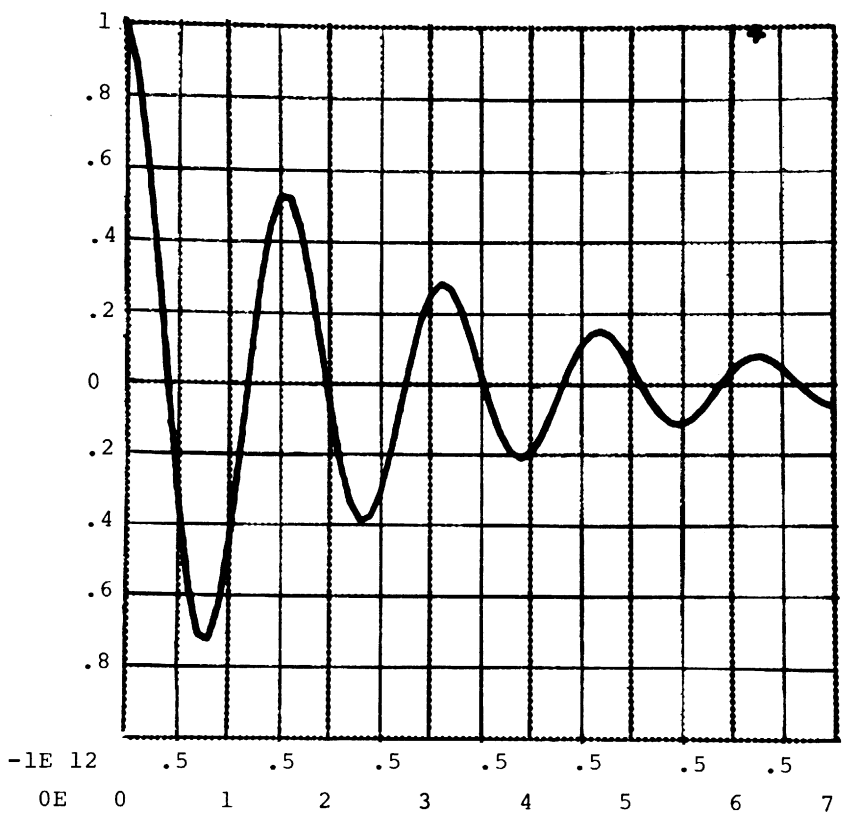


Figure 3
plot g(x)

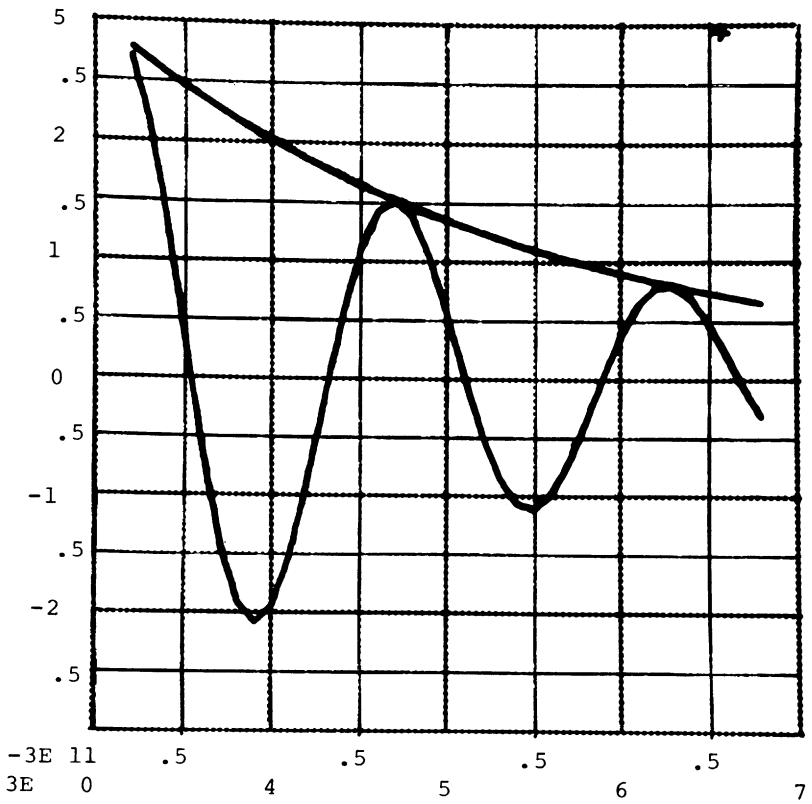


Figure 4
plot g(x) h(x) 3.2 6.8

g(x). Figure 5a is the representation of the function using linear axes. Figure 5b and 5c are the two possible semi-logarithmic plots and Figure 5d is the log-log plot. The command used to generate each figure is given below the figure. The words "log-linear", etc., can appear at any point in the sequence of parameter names. If these words are not spelled correctly, a graph with linear axes will be created.

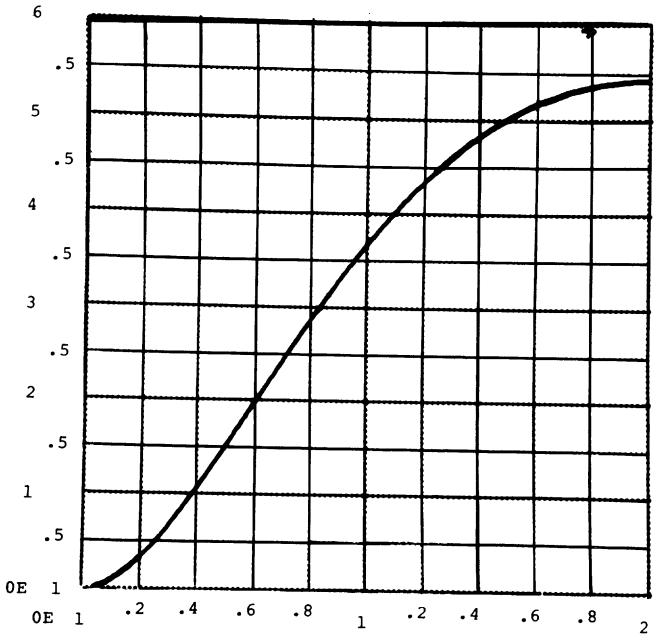
Many complex short forms of the plot command can be used, as seen by the following example.

Example H3.5

```
plot storage log-linear a(t) b(t) c(t) 1.5 lines points
COMMAND PLEASE
```

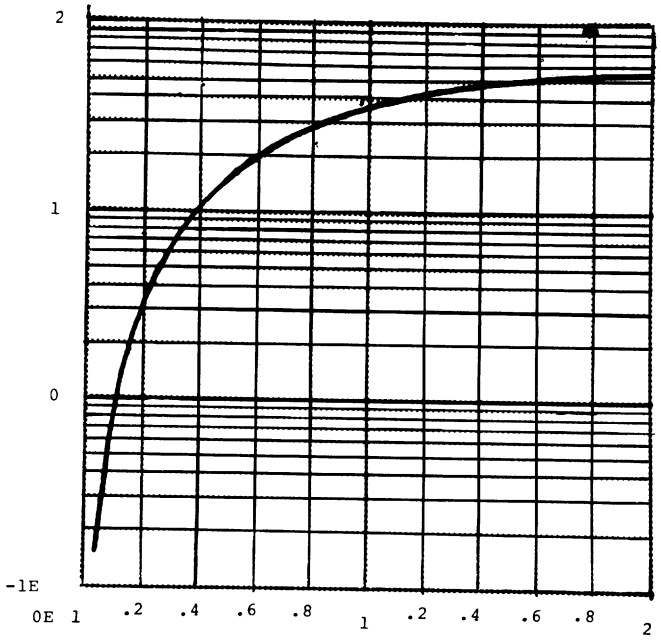
U has requested that a(t), b(t) and c(t) be plotted at intervals in t of 1.5 on the storage oscilloscope. The dependent axis will be logarithmic, and b(t) will be a point plot. Note that if only one decimal number is given it is assumed to specify the plot interval. If two numbers are given (decimal or integer) they are taken to be the range, while three, of course, completely specify MIN, MAX and DEL for the plot.

If U has tabulated a function such as a(t) at unequal intervals and the corresponding values of the independent variables are given by ind(t), the "compare" command should be used instead of the "plot" command.



5a

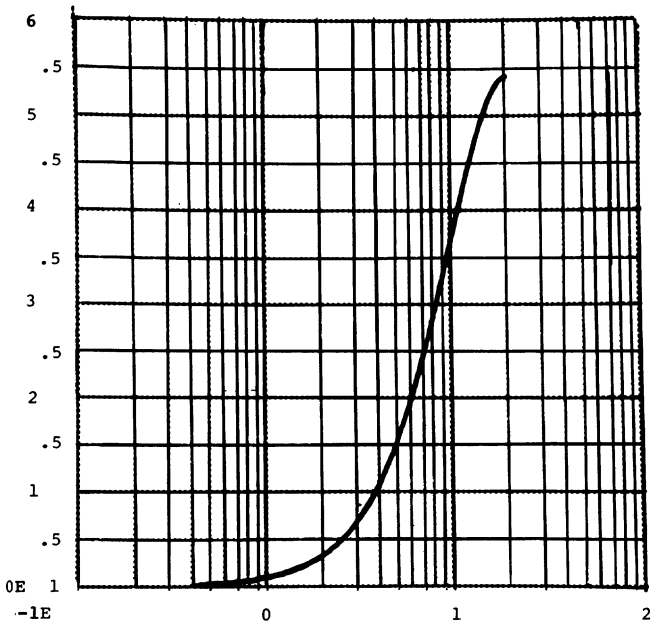
plot g(y)



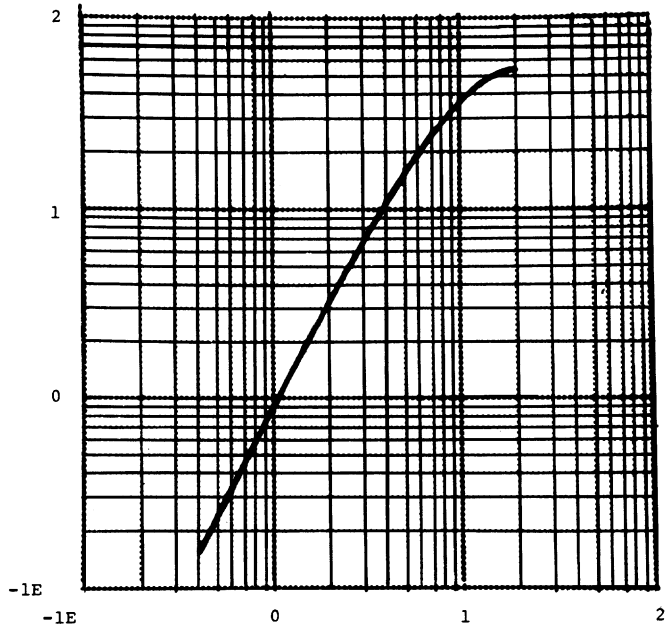
5b

plot g(y) log-linear

Figure 5



5c
plot g(y) linear-log



5d
plot g(y) log-log

Figure 5

Example H3.6

```
compare
COMPARE WILL CREATE A SCALED GRAPH BY PLOTTING THE DESIGNATED
FUNCTION(S) ON THE VERTICAL AXIS VERSUS THE VALUES
OF ANOTHER FUNCTION.

WHAT FUNCTION SHOULD BE USED TO SPECIFY THE HORIZONTAL AXIS.
ind(t)
WHAT FUNCTIONS SHOULD BE PLOTTED AS A FUNCTION OF IND(T).  a(t)
SHOULD THE PLOT BE LINEAR, LOG-LOG, LINEAR-LOG, OR LOG-LINEAR.
log-linear
DO YOU WANT A POINT OR LINE PLOT OF A(T).  line

IF YOU DO NOT WANT ALL OF THE POINTS OF THE FUNCTIONS
PLOTTED, TYPE THE INTEGERS SPECIFYING THE RANGE
OF A(T) TO BE USED.  OTHERWISE JUST GIVE A CARRIAGE RETURN.
10   30
COMMAND PLEASE
```

U has requested that log(a(t)) should be plotted against ind(t) on the ESL console, and that a line segment curve should be drawn. The values of a(t) having subscripts between 10 and 30 will be plotted against the values of ind(t) with the same subscripts. Only U himself can assure that the desired correspondence exists between those values.

In the short form of the command C makes the same assumptions that were stated in the description of the plot command, unless U provides information to the contrary. The possible short forms of the commands are identical to those for plot, with the exception of the extra function name required to specify the independent axis. For example:

Example H3.7

```
compare storage d(t) a(t) b(t) c(t) log-linear
COMMAND PLEASE
```

Y has requested that a(t), b(t), and c(t) be plotted on semilog paper as a function of d(t), not t. The storage oscilloscope will be used, and line plots for a(t), b(t) and c(t) will be generated over the entire range of d(t). C will always assume that the first function listed specifies the one to be used for the horizontal axis.

The second names of all the functions specified in either a "plot" or "compare" request need not be identical. In a "compare", C will not even check to see if the DEL's correspond, and only the correspondence of the subscripts between the functions is used in

creating the graph: that is, $a(t)_{10}$, $b(t)_{10}$, and $c(t)_{10}$ will all be plotted versus $d(t)_{10}$. A range of d(t) provided in decimal form will be regarded as meaningless and will cause an error message, followed by "COMMAND PLEASE" to be printed.

We have discussed the application of the "compare" request to the plotting of functions tabulated at unequal intervals. As the name implies, however, "compare" may be most useful for providing a visual display of the relationship between two functions. A comparison between an experimental and a theoretical function is readily obtained; for example, if the two functions are identical the plot will consist of a single diagonal line, sloping up to the right. Wherever the function named first in the "compare" request has larger values, the curve will dip below the ideal line, and conversely. The "compare" request is, in fact, exactly analogous to using an ordinary oscilloscope with both x- and y-axis inputs (except that MAP can have three simultaneous y-inputs). Thus one can also obtain the magnitude and phase shift information inherent in Lissajous-type figures with aperiodic and semi-periodic functions as well as those which are fully periodic.

4. Data Handling Requests

There are, so far as general usage is concerned, two levels at which data, command sequences and programs are stored in the MAP system; one for temporary or current purposes and the other for information required on a permanent basis. All variables and command sequences are stored in the temporary level from which they may be readily erased. If specifically requested, however, C can raise any stored information to the permanent level so that it will not be destroyed by the normal requests for erasure. Five requests, "data", "data restore", "delete", "data delete" and "data update", are available to U for the purposes of reviewing, erasing and saving data. A knowledge of the conventions used internally for data storage is required in order to use these commands. Any block of information, such as the function "g(x)", the constant "a", or one of the programs making up the MAP system is considered by C to be a file. Due to the overall structure of the time-sharing system all files must have two separate names. The following name conventions have been adopted in the MAP system:

1. All functions are stored with the name of the dependent variable as the second name. For example, phi(r) would have first name "phi" and second name "r"; the parentheses are not part of the name C uses to refer to the function.

2. All constants are automatically given the second name "const". For example, the constant "a" would be stored with first name "a" and second name "const".

3. All command sequences are automatically given the second name "loop". The first name is the one assigned by U to the command sequence.

4. The MIN, MAX, and DEL for an independent variable will be stored in a file with "MINMAX" as the first name and the name of the independent variable as the second name.

5. All programs comprising the MAP system have a second name "BSS" or "SAVED". These should never be erased.

data

After a prolonged session at the console, or when a number of persons have access to the MAP system on the same problem number (and hence the same disk storage space), U may need to be reminded of the names of all the variables, command sequences and programs in temporary storage. If he types

```
data
```

C will respond by typing out a list of the names of all files at the temporary storage level.

data restore

If U would like to obtain a completely clean slate, he may request

```
data restore
```

in which case C will type out the same list and then erase all the data on the list. No files at the permanent storage level will be erased by the request. U should generally finish his work with "data restore" so that the list will be empty for the next user.

delete

The "delete" command permits U to erase specific files from the disc storage. If U wanted C to forget the value of the constant a and the function g(x) in order that new values of these variables would be requested when they were next used, he could type:

```
delete
WHAT FILES WOULD YOU LIKE TO DELETE.
g(x)   a
COMMAND PLEASE
```

C will ignore the parentheses in the name g(x) and delete the file with first name g and second name x and the file with first name a and second name const. If a second name is not given for the last file in the string, it is assumed to be "const". In order to erase the command sequence param and the constants a and b, U should give the request:

```
delete param loop a const b
```

In this case U has given all the required information on the same line and has had to provide the second name "const" for the constant a in order to make his request clear to C. The request:

```
delete param loop a b
```

is identical to the request

```
delete param loop a(b)
```

since C ignores the parentheses in determining what files are to be deleted from the disc storage. Therefore, in case of possible ambiguity, both names of the files must be used in the data handling requests. If the delete request cannot locate the file to be deleted, it does not print a message and no action is taken.

data update

The "data update" command allows U to raise information to the permanent storage level. Assume that g(x), a, a command sequence called "param" and a MAD program called "root" had been defined since the last "data restore" request. If U desires to save the command sequence, the MAD program, and the translation of the MAD program, he could use the data update request:

Example H4.1

```
data update
THE FOLLOWING DATA ARE KNOWN TO THE SYSTEM
        PARAM          LOOP
            G             X
            A         CONST
         ROOT           MAD
         ROOT           BSS
PLEASE TYPE IN NAMES OF THE NEW FILES TO  BE  SAVED,  TERMINATE
WITH WORD END.
param   loop   root   mad   root   bss   end
                G             X
                A          CONST
COMMAND PEASE
```

After U types the names of those files to be given permanent status, C types back the remaining file names. The three files which were saved can subsequently be removed from the disc storage only by pushing the "reset line" button (on the 1050 console) twice in order to leave the MAP system, and giving a "delete" command identical to the all-on-one-line form of the delete command in the MAP system. In this case the parentheses must be omitted from function names.

data delete

The "data delete" request is equivalent to a "data update" and "data restore" request given in succession. When U requests "data delete" C will delete the files not designated for permanent retention.

While using the MAP system, U may receive a message indicating that he is trying to use more disc storage space than had been allotted. To restart the system if control has been taken from System MAP, U should type "resume map". The first statement after the "COMMAND PLEASE" must be a request to delete enough files so that U will not be using more than his allotment of disc storage.

5. Correction of Errors

Typing errors are easily corrected, since they only need to be cancelled and not erased. Two options are available for deleting mistakes. A quotation mark, ", deletes the previous character in a line of typing. Two quotation marks, "", delete the previous two characters, three delete three characters, and so on. Therefore, lines which read

```
(na=)"0.0)
```

or

```
(n=+3"""a=0.0)
```

are both interpreted as

```
(na=0.0)
```

If the mistake is so far back that a character by character deletion would be tedious, all of the characters on the line may be deleted by typing a question mark, ? . The line which reads

```
(n=3.0?(n=0.0)
```

is interpreted as

```
(n=0.0)
```

Neither erased characters nor the erasure marks (" or ?) count when considering the limitation of 72 characters per line.

If a mistake is not noticed until after a carriage return, it will be too late for U to make any corrections, unless he is defining a command sequence or typing input data, which may be edited using the requests described in Appendix III; however, C will reject most typing errors and request a new question. If, however, the error is acceptable to C, or if U changes his mind, C may be in the process of answering the question before U decides to change it. If C happens to pause, in order to ask for some information, U may cancel most questions by typing "quit". C will oblige by typing "COMMAND PLEASE". However, if C has already started on a long calculation which U would like to interrupt, he may stop it only by pushing a special key marked "reset line" on the IBM 1050; C will usually type "INT." to signify receipt of the interrupt. If U then gives a carriage return, C will type "COMMAND PLEASE". If the reset line key is pushed twice in succession, control of the computer will be taken from System MAP; normal operation can be resumed by typing "resume map". Typing "resume map" will also return control to System MAP if certain errors cause control to be transferred to the time-sharing system.

This empty page was substituted for a blank page in the original document.

I. The "Execute" Command and the Use of Auxiliary Programs

In section G it has been demonstrated that U may create personalized commands by combining separate statements already available as MAP requests. Another facility is also available to allow the creation of new commands, which is more general and likely to be more efficient in execution than command sequences which contain long arithmetic equations. The command "execute prog", with "prog" being the name of a suitable binary coded program, will cause that program to be executed. U may therefore write his own commands, in a language such as MAD, FAP or AED, and execute them together with the commands already available within MAP. A variety of subroutines are available to assist the user in achieving full compatibility with the MAP language, not only regarding required technicalities but also in less tangible aspects such as the level of communication between U and C. Thus all the MAP procedures, such as transform and convolute, are available as program subroutines and will execute in the usual manner. Other subroutines will allow U to retain the question and answer form of communication, to use the all-on-command-line short form of transmitting parameters, or to predefine the names of some or all of the parameters, even for those parts of his program which are entirely novel.

Such programming nonetheless requires a knowledge of a programming language as well as the particular requirements of programming for the execute command. These latter details, which are extensive, are described in Appendix IV.

This empty page was substituted for a blank page in the original document.

APPENDICES

This empty page was substituted for a blank page in the original document.

APPENDIX I
Summary of the Elements in System MAP

I. Numbers and Variables

A. Numbers

Constants may be specified in the form of integers (2245), fixed decimal point numbers (2245.0) and exponentiated numbers (2.245×10^3). The three forms may be used interchangeably with the following exceptions:

a. the exponential form required in equations is 2.245*10**3,but in all other places the abbreviation 2.245e3 is used.
b. integers may not be used in reply to a specific request for a decimal number.

B. Constant Variables

Constants may be referred to by a symbolic name, and are then called constant variables. A constant variable has a single name which may be almost any combination of 1 to 6 characters, except those ending in f. Corresponding to the name there will be a single number which C will use as the value of the constant.

C. Functions or Array Variables

An array has a name composed of two parts, each of which may be almost any combination of 1 to 6 characters, except those ending in f. The two parts are written together, with the second enclosed in parentheses, as name(x). Corresponding to this name there will be a block of numbers which C will use as the values of the function. Subscripts, which will be associated with each value in the array, are defined whenever the values themselves are defined, using quantities which C calls MIN and MAX or MIN, MAX and DEL. In operations involving more than one function, values which carry identical subscripts in their respective arrays will be assumed to have a meaningful correspondence.

When a function is tabulated at equal intervals of an independent parameter, C will then take the values of the independent parameter to be specified by the products of the subscripts and DEL. U may then consider the second part of the array name, such as x, to be the name of the independent parameter, and may, in fact, use it as such in arithmetic equations.

II. Arithmetic Equations

An arithmetic equation is phrased in the form:

(answer = arithmetic expression)

where "answer", in general, may be either a constant variable name or an array variable name. It is not necessary that all (or any) of the variables appearing in the expression be previously defined.

A. Arithmetic Expressions

The expression can be any mathematically meaningful combination of variable names, constants, arithmetic operation symbols and operational functions.

B. Arithmetic Operation Symbols

The following operation symbols are available:

+	plus
-	minus
*	times
/	divided by
**	raise to the power
()	group specification

The order in which these operations will be actuated will be effectively just the reverse of the above listing.

C. Operational Functions

The functions sine, cosine, arc sine, arc cosine, exponential(e^E) $\log_e$, absolute value, tangent, cotangent, arc tangent, hyperbolic sine, hyperbolic cosine, hyperbolic tangent, square root, sum, derivative and definite integral are presently available. These functions of an argument E may be specified in any arithmetic expression by using the function names, which are, respectively:

sinf(E), cosf(E), asinf(E), acosf(E), expf(E), logf(E), absf(E), tanf(E), cotf(E), atanf(E), sinhf(E), coshf(E), tanhf(E), sqrtf(E), sumf(E), derif(E) and intf(E).

The argument, E, may itself be any arithmetic expression, as defined above and in Section D of the manual.

III. Complex Procedures

Under this heading are included various mathematical procedures which are not included as possible operations within arithmetic expressions. The first group of procedures, transform, convolute, integrate and basis, are designed to operate only on arrays which are tabulated at equal intervals in the independent variable. In

every case for which the answer is an array, it will also be tabulated at equal intervals. The subtitles listed below are in the form of the actual commands which might be used during operation, assuming (except for item C) that the user gives names for all the required parameters on the command line.

A. transform g(x) sin(k) cos(k)

This command is a request for the values of:

$$\cos(k) = \int g(x) \cos(xk)\,dx$$

$$\text{and } \sin(k) = \int g(x) \sin(xk)\,dx$$

The limits on the integral will be determined by the range of g(x).

B. convolute g(x) h(x) ans(x)

This command is a request for the convolution:

$$\text{ans}(x) = \int g(\varepsilon)\, h(x-\varepsilon)\, d\varepsilon$$

The limits on the integral and the range of ans(x) will be determined by the combined ranges of g(x) and h(x).

C. integrate

This command has three options. U may obtain

$$\text{ans} = \int_a^b g(x)\,dx,$$ where a and b are constants within the tabulated range of g(x).

$$\text{or, ans}(y) = \int_a^y g(x)\,dx \quad \text{or, ans}(y) = \int_{-y}^{y} g(x)\,dx.$$

C will ask U which option is desired and for the names (and values) of the required parameters.

D. basis g(x) h(k) x(k)

This command has two options. The above is a request for an interpolation in the g(x) array, in order to obtain values tabulated at equal intervals in a new independent variable k, where the functional dependence between x and k is expressed by the array x(k). The new array will be called h(k). Alternatively,

basis g(x) h(k) 0.05

would be a request for an interpolation in the g(x) array, to obtain values tabulated at a presumably new interval in x, 0.05. The new array will be called h(k).

E. least square data(x) 4 a(x) b(x) c(x) d(x)

This command is a request for a least square fit of the function data(x) using the 4 fitting functions, a(x), b(x), c(x), and d(x). A maximum of 5 fitting functions may be specified.

F. minmax g(x)

This command is a request to redefine the range over which g(x) is defined. The command may also be used to define MIN, MAX, and DEL for an independent variable, such as x, by the statement

minmax x

G. select g(x) p(x) 2. 4.

This command has two options. The above is a request to define the function p(x) to have the values of g(x) over the range x=2. to x=4. Alternatively,

select g(x) a 2.3

would be a request to define the constant a to have the value of g(x) at x=2.3.

It should be stressed that it is not necessary for U to remember any of the details described above for these procedures. If he does not specify any parameter names with the command, C will describe the procedures and the options and will request all the required information.

IV. Input and Output Requests

A. Data Input

No specific procedure is provided for on-line data input, since all of the requests listed in Sections II, III and IV will automatically initiate specific requests for the values of previously undefined variables. The off-line loading of large blocks of data is described in Appendix II.

B. Printed Output

print cons

is a request for the value of the constant, cons.

print g(x)

is a request for the values of g(x) and its associated range and interval.

print g(x) 2. 4. .1

is one of several possible requests which will cause the printing of g(x) over a specified range with a selected interval; interpolation will be performed if necessary. In this case g(x) will be printed at intervals of x = .1 in the interval x = 2. to x = 4.

C. Graphical Output

plot g(x) log-log points 2. 4.

is a request for a graph of the log of g(x) as a function of the log of x over the range x=2. to x=4. Only the data points will be used in plotting the graph. A maximum of three functions may be plotted on the same graph. Each axis can be either logarithmic or linear, and the data points will be connected by straight line segments unless a point plot is requested.

compare a(x) g(x) log-linear 10 30

is a request for a graph of the log of g(x) as a function of a(x). The values of a(x) and g(x) having subscripts between 10 and 30 will be used in plotting the graph, which will have the data points connected by straight lines. A maximum of three functions can be plotted versus another function.

With the output commands as well as with the complex procedures (section III) the user need not present any parameter information on the command line. If, for example, only the word "compare" is typed, the system will explain the operation and will request the necessary information.

V. Data Handling Requests

A. Listing of Variable Names

data

is a request for a listing of all of the names of variables, command sequences and programs which have been defined since a previous "data restore", that is, of the temporary data.

B. Deletion of Temporary Data

data restore

is a request to erase the names and, in effect, the values of all variables, command sequences and programs not specifically raised to permanent status.

C. Raising Data to the Permanent Storage Level

data update

is a request to raise certain variables, command sequences or programs to the permanent storage level from which they can be deleted only by a CTSS "delete" request. A listing of the type provided by the "data" request is also printed.

D. Raising Data to Permanent Level and Deleting Temporary Data

data delete

is a request to raise certain variables, command sequences or programs to the permanent storage level and to delete all of the remaining temporary data. The request is equivalent to a "data update" followed by a "data restore" request.

E. Deletion of Specific Data

delete g(x) a

is a request to delete from temporary or permanent storage the function g(x) and the constant a. No other information in MAP storage is affected by the request.

VI. Defining, Editing, and Executing Command Sequences

A sequence of MAP commands can be defined,edited, and executed by use of the "create", "edit" and "run" requests. The "create" request allows the definition and initial editing of a sequence of MAP commands. The user will assign the sequence a name and the sequence will be stored by MAP until deleted by one of the data handling requests.

run value 4

is a request to execute the command sequence assigned the name "value" starting at line 4. The execution of the sequence will proceed exactly as if each statement had been individually typed on the console.

edit value

is a request to edit the command sequence with the name "value". Changes can be made at any time and the edit request may be followed immediately by another "run" request.

VII. Creation of New Commands and Use of Auxiliary Programs

execute prog a(x) b(x) c(x)

is a request for the execution of the program "prog", which must be a program available in BSS or SAVED form within the time-sharing system. If the program is written using the conventions described in Appendix IV, the user can create new commands or use his own programs within the MAP System by means of the execute request. The parameter names a(x), b(x), and c(x) are available for use by the program to be executed (up to 13 arbitrary parameter names and/or values may be supplied).

VIII. Typing Errors

A. A number of consecutive quotation marks (") cancel an equal number of immediately preceding characters on a line.

B. A question mark (?) cancels all of the preceding characters on a line.

C. A single depression of the "Reset Line" key on the 1050 IBM or the "break" key on the Model 35 teletype followed by a carriage return generally interrupts the computer and allows a new question to be phrased.

D. Typing the word "quit" in response to an interrogation by the computer will generally stop the operation in progress and allow the user to give another command.

E. The editing requests described in Appendix III can be used to edit command sequences and numerical values provided as input data.

APPENDIX II

I. Formats for Printed Output

The form which will be used for printing out an array variable in response to a print request will be four-significant-figure exponentiated numbers, six values to a line. If that form is not satisfactory, it may be altered by specifying an alternative format on the request line such as:

print y(x) 5F12.7

Two general types of format may be used:

A. nFm.p specifies n values to a line, each value to be typed as a fixed point decimal number with p digits retained after the decimal point, and m places allotted to the entire value (including the decimal point, the sign and any leading blank spaces desired).

B. nEm.p specifies n values to a line, each value to be typed as a normalized exponentiated number with p significant figures retained, and m places allocated to the entire value (including the decimal point, the preceding zero, the exponentiating symbol (E), the signs of the value and the exponent and any blank spaces desired).

A maximum of 72 characters, including blanks, may be specified for any one line.

No more than eight significant figures should be requested for any value. If an "F" format is used and the number is too large to be printed by that format, control of the printing will be taken from System MAP. The command "resume map" must then be given to restore normal operation.

II. Off-Line Input of Data

When a large amount of experimental data is to be input to the computer, it may be more convenient to submit the values on punched cards than to type them in on the console. A format for punching the cards should be chosen that allows all of the values of the data to be represented. This format must be used in all cards of a given set. Only the first 72 columns on each card should be used, and each value of the data must be positioned as far to the right in the field as possible when using an "E" format.

The data should be preceded by a card of the form:

INPUT PROB PROG NAME FORMAT

on which the fields are separated by one or more blanks.

PROB is the user's problem number.

PROG is the user's programmer number.

NAME is the name the user desires to specify for his input data. The name can not contain more than 6 characters.

FORMAT is the format used in punching the cards. The format specification can not consist of more than 6 characters.

The last card of the deck must have *EOF* beginning in column 8. The deck should be submitted for loading onto the disk at the appropriate location at either Project MAC or the Computation Center, depending on the computer being used.

When the data has been loaded and is to be used in the MAP System, U should refer to it by a new name, which he plans to use throughout his calculations. The new name is required since the data will be loaded with first name NAME and second name FORMAT, and FORMAT will contain numbers, which are illegal in a MAP variable name. When C asks if the values have been defined under a different name, U should type "NAME FORMAT" as the old name. The data on the cards will then be converted to the required binary form and will be available as a MAP function with the new name.

Appendix III

Editor for Input Data and Command Sequences*

When the numerical values of a function have been requested by C or whenever the command "create" has been given in order to define a command sequence, C will type "INPUT:". U may then type, as rapidly as he desires, as many lines of input as necessary. When U gives two successive carriage returns to indicate the end of the input, C will type "EDIT:". U may then alter the block of input, which we will refer to as a file, with various specific requests to the editor. These requests will be described below. In order to alter a previously defined command sequence, U may reach the editor directly by typing the MAP command "edit xxxxxx", where xxxxxx is the name of the stored sequence.

Since the requests refer to specific lines in the file, it will be useful to imagine a pointer which designates the line in question. When the editor is called automatically by the two carriage returns terminating the input, the pointer will be positioned at the last line of the file. When U uses the MAP command "edit", the pointer will be before the first line at the top of the file. If a request causes the pointer to move past the end of the file, C will type "END OF FILE REACHED BY.....", where .... will be the responsible request.

* The MAP editing facility utilizes the CTSS 'ED' command developed by R.C. Daley and C. Garman of the M.I.T. Computation Center. This appendix discusses only the editing features necessary for use within the MAP system; additional information about the 'ED' command is available in the CTSS Programmer's Guide.

EDIT REQUESTS

No response is made by C to a request unless the response is indicated in the description of the request.

REQUEST: FIND LINE
ABBREVIATION: F
ERRORS: END OF FILE

The FIND request is used to move the pointer down from its present position to the line specified by LINE. LINE is a sufficient portion of a line, starting from the left side, to designate that line uniquely. Matching is done only on the non-blank characters specified in LINE. For example, the request,

```
F (u(r)   =
```

might be used to find the line,

```
(u(r) = sinf(a(r))
```

REQUEST: LOCATE STRING
ABBREVIATION: L
ERRORS: END OF FILE

The LOCATE request is used to move the pointer down from its present position to the first line which contains the entire character string specifed by "STRING".

REQUEST: NEXT I
ABBREVIATION: N
ERRORS: END OF FILE

This request is used to move the pointer down from its present position in the file. "I" specifies the number of lines to be skipped over. If I is "0" or not specified, it is assumed to be "1" and the pointer will be moved to the next line in the file. If the NEXT request is given after the end of file has been reached, the pointer will reset to the beginning of the file and moved "I" lines from there.

REQUEST: DELETE I
ABBREVIATION: D
ERRORS: END OF FILE

The DELETE request will delete "I" lines from the file starting with the line at which the pointer is currently positioned. The pointer is left at the position vacated by the last line deleted by this request. If I is "0" or left unspecified, only the current line will be deleted.

REQUEST: PRINT I
ABBREVIATION: P
RESPONSE: printed lines
ERRORS: END OF FILE

The PRINT request will print "I" lines from the file starting with the line at which the pointer is currently positioned. Upon completion of this request, the pointer will be left pointing to the last line printed. If I is "0" or left unspecified, one line will be printed.

REQUEST: RETYPE LINE
ABBREVIATION: R
ERRORS: none

This request will cause the line at which the pointer is currently positioned to be replaced by LINE. The pointer <u>is</u> <u>not</u> moved by this request. The first blank following "RETYPE" is part of the request and, therefore, is not part of the new line.

REQUEST: TOP
ABBREVIATION: T
ERRORS: none

This request will cause the pointer to be reset and positioned just above the first line in the file.

REQUEST: BOTTOM
ABBREVIATION: B
RESPONSE: INPUT:
ERRORS: none

This request will cause the pointer to be positioned after the last line in the file. All subsequent typing will be treated as input and added to the file.

REQUEST: INSERT or carriage return
ABBREVIATION: I
RESPONSE: INPUT:
ERRORS: NONE

All subsequent typing will be treated as input and inserted <u>after</u> the line at which the pointer is currently positioned. If the INSERT request is given immediately following a TOP request, the inserted lines will be placed at the beginning of the file.

REQUEST: INSERT LINE
ABBREVIATION: I
ERRORS: none

The INSERT request may also be used to insert a single line immediately after the current position of the pointer. The first blank following "insert" is part of the request and, therefore, is not part of the new line.

REQUEST: CHANGE /string1/string2/ J G
ABBREVIATION: C
ERRORS: END OF FILE

This request will examine "J" lines starting at the line at which the pointer is currently positioned. If the character "G" is present, every occurrence of string1 will be replaced by string2 in the lines examined. If "G" is not present, only the first occurrence of string1 will be replaced by string2 in each line. Upon completion, the pointer will be left positioned at the last line examined by this request. If J is "0" or left unspecified, it is assumed to be "1" and only the current line will be examined. If the character "/" appears in string1 or string2, another character not appearing in string1 or string2 must be used to delineate the two strings. "string1" and "string2" may be of different lengths.

EXAMPLES:

```
line:        ALPHA = ALPHA + ALPHA
request:     C /ALPHA/BETA/
new line:          BETA = ALPHA+ALPHA
request:     C /ALPHA/DELTA/ 1 G
new line:          BETA = DELTA+DELTA
request:     C /DELTA//
new line:          BETA = +DELTA
```

REQUEST: VERIFY
ABBREVIATION: VE
ERRORS: none

The VERIFY request sets the verify mode. In the verify mode the lines selected by the requests FIND, LOCATE and NEXT and all lines altered by a CHANGE request will be printed. It is recommended that VERIFY be the first request given to the editor in order to avoid errors.

REQUEST: BRIEF
ABBREVIATION: BR
ERRORS: none

The BRIEF request turns off the verify mode.

REQUEST: FILE NAME

ABBREVIATION: FL
RESPONSE: COMMAND PLEASE (or another MAP response)
ERRORS: NO FILE NAME GIVEN

This request is used to terminate the editing process; the new file will be written onto the disk. In the case of data input, NAME must be "input data". When defining a command sequence NAME must be a name of 6 or fewer characters used to designate the sequence.

The data should be preceded by a card of the form:

INPUT PROB PROG NAME FORMAT

on which the fields are separated by one or more blanks.

PROB is the user's problem number.

PROG is the user's programmer number.

NAME is the name the user desires to specify for his input data. The name can not contain more than 6 characters.

FORMAT is the format used in punching the cards. The format specification can not consist of more than 6 characters.

The last card of the deck must have *EOF* beginning in column 8. The deck should be submitted for loading onto the disk at the appropriate location at either Project MAC or the Computation Center, depending on the computer being used.

When the data has been loaded and is to be used in the MAP System, U should refer to it by a new name, which he plans to use throughout his calculations. The new name is required since the data will be loaded with first name NAME and second name FORMAT, and FORMAT will contain numbers, which are illegal in a MAP variable name. When C asks if the values have been defined under a different name, U should type "NAME FORMAT" as the old name. The data on the cards will then be converted to the required binary form and will be available as a MAP function with the new name.

Appendix IV
Programs for the Execute Command

The "execute" request facilites the expansion and personalization of the MAP System, and also allows the possibility of improved efficiency in the event that certain command sequences (particularly those with long arithmetic equations) are to be used frequently. Its use, however, presupposes the existence of suitable programs. Even though many subroutines have been written to simplify the process, the writing, translating and error correction of the programs may require a significant amount of time and a technical compentence on the part of U beyond that required to use MAP.

Before we discuss the mechanics of programming for "execute", it will be useful to recall the following basic attributes of the MAP system, which are, in the main, responsible for the special requirements of such programming.

1. All variables are referred to by name; in particular, tabular data (or functions) are referred to in standard mathematical form, such as "y(x)".

2. None of the available procedures have pre-defined parameter names. That is, at execution time U can request the procedure to operate on any functions rather than ones of specific names.

3. If the parameter names are not supplied on the command line, C will request them.

4. If specified parameters (i.e. functions or constants) have not been defined previously, C will request values for them.

For "execute", U must keep the first of these in mind in order to transfer data to and from other MAP procedures. The other points may be relaxed, as suits U's needs.

In the following discussion it will be assumed that U has a basic knowledge of CTSS (the time sharing system) and the MAD programming language. Actually, programs can be written in any language which produces a compilation acceptable to the BSS loader. If any MAP System subroutines are used, the language must be able to call subroutines written in MAD, but it need not produce subroutines that can be called by MAD.

Consider the following command sequence:

```
(g(x)=x*sinf(a(x)+d))
transform g(x)  sint(u)   cost(u)
integrate 1 sint(u) ans
```

which might represent a sequence that U would want to use many times in analyzing experimental data. When the command sequence was executed C would ask for any additional required information (such as the values of the integration limits). We will assume that U would like to write a program for the "execute" command, which will simulate this sequence. The following MAD program will serve that purpose, and will be discussed in detail to illustrate how the facilities within the MAP system can be used.

Example A.1

```
      INTEGER  MIN, MAX, I
      DIMENSION AA(1000), G(1000)
      EXECUTE IN.($A(X)*$,AA, MIN, MAX, DEL)
      C=VALUE.($D*$)
      THROUGH LOOP, FOR I=MIN, 1, I.G. MAX
LOOP  G(I-MIN)=I*DEL*SIN.(AA(I-MIN)+C)
      EXECUTE OUT.($G(X)*$,G, MIN, MAX, DEL)
      VECTOR VALUES TRANSF=$G(X) SINT(U) COST(U)*$
      EXECUTE TRANS1.(TRANSF)
      VECTOR VALUES INT=$1 SINT(U) ANS*$
      EXECUTE INTEG1.(INT)
      EXECUTE CHNCOM.
      END OF PROGRAM
```

The first two statements, like all the rest in such programs, are standard MAD statements. These serve to specify that the MAD variables min, max and i are integers and to allocate 1001 storage units each to the arrays aa and g. Notice that the program must be written as a "main program" rather than as subroutine. It must be given a name, which is assigned using the standard CTSS procedure (not shown here). The transfer of variables between this program and the disc storage where they are stored by MAP, will be described below. The dimension of 1000, for aa and g, is the most general one for variables to be transfered since no MAP function may contain more than 1000 points. If U were certain that the program would never use that large an array he could use any safe smaller number. The subroutine "IN." loads the function a(x) from the disc storage into the array "aa" and will set the MAD variables min, max and del equal to the MIN and MAX subscripts and DEL for a(x). If no values are found for a(x), values will be requested in the normal MAP

manner. The function name must be provided to the subroutine in Hollerith (BCD) form; hence the MAD notation involving the $'s. The "*" indicates the end of the BCD information, much as the carriage return signifies the end of a line of console input.

The use of the "VALUE." subroutine in line 4 illustrates how the value of the constant d is obtained; if d had not been previously defined, a value will be requested when the program is executed.

In lines 5 and 6 the values of g(x) are calculated using a "through" statement which includes all the values between MIN and MAX. A convention of the MAP system is that "IN." will transfer a function so that the value with subscript MIN is stored in the first storage unit allotted to the corresponding program array. For example, the element of a(x) with subscript MIN is loaded into that storage element which is called aa(0) by MAD. This convention results in a large saving in storage space when MIN is a large number. In refering to the elements of the array, therefore, U must always account for the shift; in this instance the value of a(x) corresponding to x=I*DEL will be stored with subscript (I-MIN). The OUT. subroutine, as is probably obvious, is just the reverse of IN. and, as used here, will create a MAP function, "g(x)", having the values of the MAD array "g".

All the procedures, such as transform, are called by giving the first five letters of the procedure name followed by a "1.". For example, TRANS1. calls the transform procedure. Notice that the entire argument of all of the MAP procedure subroutines must always be Hollerith (BCD), which is the form in which the arguments would normally be supplied via the typewriter. Only if the required BCD argument contains fewer than six characters (including the terminating asterisk) may it be provided within the argument parentheses (as has been done with both "d" and "a(x)" in the calls to "IN." and "VALUE."). In this example, the arguments of both "TRANS1." and "INTEG1.", which simulate the short form of the command, are too long to be provided directly. They are therefore preset using "vector values" definitions of the arays "transf" and "int", which are in turn provided as the arguments in the calling sequences. Any additional required information (such as the integration limits) will be requested by C each time the program is executed.

The call to the subroutine CHNCOM. automatically returns control to the MAP System. This statement, which is analogous to the MAD "function return" statement in a subroutine, must be used whenever the next response from C is expected to be "COMMAND

PLEASE". All of the MAP system subroutines, such as "IN.", "OUT.", etc. will be automatically loaded when the "execute" command is given. In a subsequent portion of the appendix all of the MAP subroutines available for use by U in writing programs are described in detail.

Once U had successfully translated the MAD program he could execute it at any time during normal use of the MAP System. Assuming the name given to it had been "test", the MAP request would be given in the following manner:

Example A.2

```
COMMAND PLEASE
execute test
DEC. VALUE OF CONSTANT   D PLEASE  3.4

TRANSFORM BEING PERFORMED
PLEASE PRINT ON THE NEXT LINE MIN,MAX, AND DEL FOR THE
VARIABLE U.
-6.   6.   .2
MIN = -30   MAX = 30

INTEGRATION BEING PERFORMED
DECIMAL VALUE OF LOWER LIMIT PLEASE =  -3.
DECIMAL VALUE OF UPPER LIMIT PLEASE =    3.

COMMAND PLEASE
```

Evidently U had previously defined a(x); otherwise, numerical values would have been requested in the usual manner. The MIN, MAX, and DEL for the independent variable u and the value of the constant d, being "data", would not be requested again unless the values given here were subsequently erased. On the other hand the integration limits will be requested each time since they are simply parameters of a specific execution which, in this case, are not supplied "on the command line."

Although this MAD program might satisfy U's immediate requirements, it is doubtful that he would always name his data a(x) and he also might want to name the results of the transform differently whenever the program was executed. In that case U could write the MAD program with greater generality so that the names of the important functions could be supplied at the time the "execute" statement was given. In the following example, the program has been written such that U can use arbitrary names in place of the specific names a(x), g(x), sint(u), cost(u) and ans which were used in the previous example.

Example A.3

```
     INTEGER MIN, MAX, I, ARG, NAMES
     DIMENSION AA(1000), ARG(6), NAMES(8), G(1000)
     EXECUTE SETUP.(NAMES,9)
     ARG(0) = NAMES(0)
     ARG(1) = NAMES(1)
     ARG(2) = $*$
     EXECUTE IN.(ARG, AA, MIN, MAX, DEL)
     C=VALUE.($D*$)
     THROUGH LOOP, FOR I=MIN, 1,I.G.MAX
LOOP G(I-MIN)=I*DEL*SIN.(AA(I-MIN)+C)
     ARG(0)=NAMES(2)
     ARG(1)=NAMES(3)
     ARG(2)=$*$
     EXECUTE OUT.(ARG, G, MIN, MAX, DEL)
     ARG(2)=NAMES(4)
     ARG(3)=NAMES(5)
     ARG(4)=NAMES(6)
     ARG(5)=NAMES(7)
     ARG(6)=$*$
     EXECUTE TRANS1.(ARG)
     ARG(0)=$1$
     ARG(1)=NAMES(4)
     ARG(2)=NAMES(5)
     ARG(3)=NAMES(8)
     ARG(4)=$*$
     EXECUTE INTEG1.(ARG)
     EXECUTE CHNCOM.
     END OF PROGRAM
```

In order to use the same names as were predefined in the previous example the program would be executed in the following manner:

```
execute test a(x)  g(x)  sint(u)  cost(u)  ans
```

To write a program in which the names of the important parameters are substitutable, U must decide which parameters he will want to supply each time the program is executed. In the first example the program was written in such a way that none of the function names could be changed at the time the program was executed; changes in the names could be made only by altering one or more of the statements in the program and then recompiling the entire program. In the second example U wrote the program in such a manner that the name of the argument of the sine, the name of the function to be transformed, the name of the sine and cosine transforms, and the name of the result of the integration are to be

provided as part of the execute command. Obviously the user of the program must be aware of the order in which the author expects the parameter names to be provided, just as U must know how to use the short form of the MAP commands.

At execution time the parameter names must be obtained by the subroutine SETUP. This subroutine is called with two arguments: the first is an array which will ultimately contain the information typed on the command line after the word "execute" and the name of the MAD program. Note that this array must be specified to be of integer mode. The second argument is the number of parameters expected to be supplied when the execute command is given. This number should be calculated assuming that a quantity within a parentheses is a separate parameter, that is, a(x) is two parameters. If the correct number of parameters is not supplied as part of the "execute" command, C will give U an opportunity to retype the portion of the line following the name of the MAD prgram. The array "names" will contain in successive computer words the parameters typed on the line with the execute command: for example, names(4), in the calling sequence given above, will be the word created by MAD from $ sint$. Since SETUP obtains all the parameter names provided on the command line, it is necessary to break up the list into the separate groups of names required for each of the subroutines which may be involved and to provide the asterisks at the end of each group. This has been accomplished by repetitively setting the "arg" vector equal to the appropriate elements of the vector "names", which was obtained by SETUP. In all other respects this procedure is equivalent to the use of a vector values definition since all of the subroutines for calling MAP procedures will replace, if necessary, any parentheses in the "vector values" definition of the parameter names by blanks and will separate all the parameters into separate computer words by the presence of the blanks, just as "SETUP." does.

The subroutines stored in the MAP system library and available for use in programming can be divided into two categories: those necessary for utilizing the present capabilities of the MAP system and those which will be useful in attaining an equivalent sophistication in the novel parts of programs written for the "execute" command.

This empty page was substituted for a blank page in the original document.

A. Subroutines for Access to MAP System Facilities

1. Subroutines for Calling MAP Procedures

In order to use "basis" for example, the calling sequence is:

```
execute basis1. (arg)
```

where "arg" may have one of three different forms, depending upon the amount of information U desires to provide in the call to the subroutine.

a. The simplest way to call BASIS1. is equivalent to typing just "basis" on the console; all of the required information will be requested at execution time. In this case arg is simply $*$, which indicates that no parameter names are predefined and that none will be provided with the MAP "execute" request. The calling sequence in a MAD program would be:

```
execute basis1.($*$)
```

b. The second type of call is one in which U predefines some or all of the necessary parameter names and expects any unspecified ones to be requested at execution time, as in (a). Such calls will be equivalent to such MAP commands as:

```
basis   g(x)   a(k)   x(k)
```

or

```
basis   g(x)   a(k)
```

For both simulations arg should be defined in the MAD program with a vector values statement, such as

```
vector values arg=$g(x)   a(k)   x(k)*$
```

for the first case, and

```
vector values arg=$g(x)   a(k)*$
```

for the second case.

U must follow the convention common to all MAP procedures: the names provided must be supplied in the same order that they would be requested by C, and no intermediate name in the expected sequence can be omitted.

If the information provided within the $'s of the vector values statement contains 6 or fewer characters, a simplified calling sequence can be used. In order to call BASIS1. in a manner equivalent to the MAP command

```
basis g(x)
```

the "vector values" statement may be avoided and the argument provided directly, as

```
execute basis1.($g(x)*$)
```

Of course, no other name can be substituted for g(x) without changing this statement and recompiling the entire program.

c. In general U will want to provide different parameter names each time he runs the program containing the call to BASIS1. and yet may not desire to recieve the individual specific requests by C. Then, as in example A.3, the program must expect to find the parameter names typed on the line with the execute command, and these names will be used in the call to BASIS1.. Of course the user of the program must know which parameter names the author expected him to provide with the execute command, which names C is expected to request and which names, if any, are predefined. The program must use the subroutine SETUP. to obtain parameter names from the execute command line and some (or all) of these names will be used in the call to BASIS1. The pertinant names must be assembled by the program into an array, such as "arg", the last word of which contains an asterisk, which indicates the end of the list of parameter names for BASIS1. The calling sequence is

```
execute basis1.(arg)
```

This is exactly the same procedure as was used in example A.3. The array arg must be included in an integer statement.

All of the MAP prodecures are available as subroutines. The following subroutines call the corresponding MAP procedure:

Procedure	Subroutine
Compare	COMPAR1.
Convolution	CONVO1.
Differentiation	DIFFE1.
Integration	INTEG1.
Least Square Analysis	LEAST1.
Minmax	MINMA1.
Plot	PLOT1.
Print	PRINT1.
Select	SELEC1.
Transform	TRANS1.

A subroutine for performing differentiation is included in the list although this operation is provided in MAP as an operational function for equations, rather than as a separate procedure. The

procedure expects two function names as parameters; the first is the name of the function to be differentiated and the second is the name of the answer. If both names are not provided, they will be requested.

Whenever an additional procedure is included within the MAP system, the appropriate subroutine will be added to the system library with a name consisting of the name of the procedure (or of the first five letters) followed by "1.".

2. Subroutines for Manipulating MAP Functions and Constants

The following subroutines facilitate the use of MAP-created functions, such as g(x), and MAP constants within a MAD program. In all cases the array "arg" specifying a MAP name must be of integer mode.

a. IN. loads a MAP function into a designed array and provides the values of MIN, MAX, and DEL for the function.

Calling sequence from MAD:

```
execute in.(arg, array, a,b,c)
```

arg	The name of a MAP function in one of the forms described in the proceding section,
array	An array into which the values of the function having the specified name will be loaded,
a,b,c	Variables that will be set equal to the MIN, MAX, and DEL, respectively, of the function.

If the function is not defined, numerical values will be requested in the usual manner. If arg contains more than the two names necessary to specify a function, or if arg does not contain the terminating "*", an error message will be printed, followed by "COMMAND PLEASE". arg, a, and b must be defined to be of integer mode.

b. OUT. creates a MAP function and stores it on the disc.

Calling sequence:

```
execute out.(arg, array, a,b,c)
```

The function with the name specified by arg will be created from the values stored in array and will have MIN=a, MAX=b and DEL=c; a, b, and arg must be integers. The number of points in the array must be (b-a+1). The error returns are identical to those for IN.

c. VALUE. will load the value of a MAP constant.

Calling sequence:

```
d=value.(arg)
```

d will be set equal to the value of the MAP constant specified by arg. The constant is always a floating point number. If the constant is not defined, the value will be requested in the usual manner. If arg contains more than a single variable name, an error message will be printed, followed by "COMMAND PLEASE".

d. CONST. will create a MAP constant and write its value on the disc.

Calling sequence:

```
execute const.(arg,val)
```

A MAP constant with the name specified by arg and with the value of val will be created. Val must be a floating point number.

e. RANGE. will obtain the MIN,MAX and DEL for an independent variable from the disc whenever a file with the first name MINMAX and a second name equal to the name of the variable exists. If this file does not exist, the values of MIN, MAX, and DEL will be requested from the console. U may then supply either decimal or integer values for MIN and MAX; decimal values will be converted to the appropriate integers by using the value of DEL provided. The appropriate MINMAX file will also be created.

Calling sequence:

```
execute range. (arg,min,max,del)
```

where arg is the name of the variable of interest. min, max, and del will be set equal to the appropriate values; min, max and arg must be of integer mode.

In general RANGE. will not need to be used whenever a function is being obtained by either IN. or FGET. (which will be described in Section B of the Appendix). When values of MIN, MAX or DEL are required, they will be requested by these routines.

f. BCDBIN. will convert a number provided as one of the parameters of a MAP execute statement into a floating point number.

Calling sequence:

```
a = bcdbin.(arg)
```

a will be set equal to the floating point value of the BCD number which had been obtained from the execute command line by using

subroutine SETUP. The initial number must consist of 6 or fewer characters including the decimal point and the "e" (if one is used to indicate exponentiation).

g. SETUP. will obtain the parameters provided as part of the MAP "execute" command for use within a MAD program. SETUP. must be used if any of the parameter names within a program are to be substitutable.

Calling sequence:

```
execute setup.(names,num)
or  a = setup.(names, num)
```

num can be either 0 or the number of parameters expected to be provided as part of the execute command and must be designated to be an integer in the program. names is the array in which the parameter names given with the MAP execute command will be stored and likewise must be of integer mode. All parentheses are replaced by blanks, and the blanks are used to separate the parameters. Each parameter is stored in a separate computer word in the array "names", and is right justified with leading blanks. If the second calling sequence is used, a will contain the number of parameters obtained from the execute command and both a and SETUP. must be included in an integer statement.

If num is specified as 0, SETUP. will assume that the first parameter in the execute command is the number of additional parameters to be provided. In this case the array "name" will contain only the additional parameters and will not include the first parameter specified. If that number of parameters is not found, the value of a will be the number of additional parameters actually found. If the proper number of parameters are not found, C will ask U if he wishes to retype the required parameters. If he answers "no" arg will contain only the parameters found, and a will be the number found.

B. Subroutines Useful in Adding Commands to the MAP System

The subroutines described in part A should enable U to write a private command for use with the MAP execute command if he has no need to communicate with the program during its execution (other than is implied by the use of MAP subroutines) and intends to include only minimal error checking. The following groups of subroutines will aid U in writing more sophisticated private commands.

1. Subroutines Assisting Error Checking

a. DOI2. The subroutine will check 2 successive computer words containing BCD information for the existence of a decimal point.

Calling sequence:

```
a = doi2.(name(0))
```

a will equal 1 if name(0) or name(1) contains a decimal point; if no decimal point is located, a will equal 0. If only one word is to be examined, the entry DOI1. should be used. a, DOI2. or (DOI1.) and name must be of integer mode.

b. ERROR. This subroutine provides, with a single call, the printing of a message and an exit from the program. The status of the program being executed will be destroyed. The calling sequence in a MAD program is

```
execute error.(mess)
```

where mess is the name of a vector which contains the error message. The message can be defined in a "vector values" statement, such as

```
vector values mess=$ any error message $,777777777777K
```

The message will be printed on the console, 72 characters per line, and followed by "COMMAND PLEASE". (The subroutine PRNTP., described in the CTSS Programmer's Guide, section AG.1.03, can be used to print a message at any point in a program, without leaving the program. The subroutine CHNCOM., illustrated in example A.1 and described in section AG.8.03 of the CTSS Programmer's Guide, can be used to leave a program at any point, without printing a message.)

c. EXIT. A subroutine of this name is included in the MAP system library to avoid transfers out of the control of MAP whenever this subroutine name is used inadvertently in a MAD program written for the "execute" request or called as an error exit by an existing CTSS subroutine. In its MAP form the EXIT. subroutine is identical to a

call to CHNCOM. except that it will erase any core images which have been saved on the disc by the MAP system, whereas CHNCOM. will not. Therefore EXIT. provides a method for immediately obtaining a "COMMAND PLEASE", even if the program in question has been called with an "execute" request in the middle of a command sequence.

d. NUM. The subroutine will determine whether a word of alphabetic (BCD) information contains any number and/or decimal points, and will provide the total number of such characters. NUM. is particularly useful in determining if U has provided a possible MAP variable name since MAP names can not contain numerical characters.

Calling Sequence:

```
a = num.(b)
```

a will be equal to the number of digits and decimal points in the BCD word b. a,b, and NUM. must all be declared to be of integer mode.

2. Subroutines Facilitating Program Communication with the Console

The most important feature of the time-sharing system is the variety of ways in which it allows U and C to communicate during the solution of a problem. In order to write a MAP command it is necessary to have available subroutines for printing messages on the console, and for converting the console input into a form that can be used by a program.

a. Printing messages on the console.

MAP uses the subroutine ERROR. or the CTSS system subroutine PRNTP. to print text messages on the console. The PRINT FORMAT feature of MAD is used in the print command.

b. Reading information from the console.

All information read from the console is obtained in BCD form and must be converted to binary form if it is to be used for numerical calculations. Manipulations upon the BCD information before processing or printing are frequently necessary. The available subroutines are:

GET. obtains an input line from the console, deletes leading blanks, replaces any parentheses by blanks, and divides groups of characters separated by blanks into individual computer words. If a group of characters contains more than 6 characters, only the first 6 characters are retained. The resulting character groups are right

justified with leading blanks.

Calling sequence:

```
execute get.(arg,m)
or  k = get.(arg,m)
```

The array "arg" wil contain k words of information derived from the input line by replacing all parentheses by blanks, and storing each group of characters in one word. The number of words, m, to be obtained must be specified in the call. m and arg must be of integer mode as must k and GET. if the second calling sequence is used. If one of the character groups is the word "quit", EXIT. will be called. For example, if the input line were

```
alpha a(x)  g(gamma)
```

arg would contain:

Word	BCD contents
arg(0)	_alpha
arg(1)	_____a
arg(2)	_____x
arg(3)	_____g
arg(4)	_gamma

(where _ is used to represent a blank)

and k would have the value 5. Due to the way MAD stores arrays, arg(4) will be in the numerically lowest storage location, a fact which will be important to U only if he plans to write part of his command in a language other than MAD.

GET2. performs much the same function as GET. with two execptions: groups of characters can contain up to 12 characters and are allotted 2 computer words, and the resulting character groups are left justified with trailing blanks. If a group of characters contains 6 or fewer characters, the second of the two words will only contain blanks. The calling sequence for GET2. is the same as for GET.. If an input line were:

```
3.1415926  7.5e2 a(x),
```

and GET2. were called by the statement

```
k = get2.(arg,4)
```

the array would contain:

word	BCD contents
arg(0)	3.1415
arg(1)	926___
arg(2)	7.5e2_
arg(3)	_____
arg(4)	a_____
arg(5)	_____
arg(6)	x_____
arg(7)	_____

k would have the value 4, the number of separate groups of characters found. k, GET2., arg, and m must all be of integer mode. As was the case for GET., the word "quit" among the input will result in a call to EXIT. GET2. can also be called by an "execute" statement if the value of k is not required.

In general GET. is used when the input line is expected to contain MAP names, which by convention are expected to be right justified. GET2. is used primarily for numerical input.

The conversion routines IOHF. and IOHI. assume that all numbers to be converted from BCD to binary are in the form produced by GET2., left justified with trailing blanks.

IOHF. will convert two consecutive BCD words of the form produced by GET2. to a floating point number. The "E" notation for power of 10 exponentiation is acceptable, so short forms such as 7.5e2 can be used to indicate 7.5×10^2.

Calling Sequence:

```
num = iohf.(arg(2))
```

num will be set equal to the binary floating point values of the BCD number stored in arg(2) and arg(3).

IOHI. will convert two consecutive BCD words of the form produced by GET2. into a binary integer. IOHI. will regard both a decimal point and an "E" as illegal characters and will generate the error message "ILLEGAL CHARACTER IN NUMBER." The calling sequence is the same as for IOHF., but num and IOHI. must be of integer mode.

PARENS. will examine two successive computer words containing the two names of a function and will insert parentheses around the second so that, when printed, the full name will have the standard MAP form.

Calling Sequence:

execute parens.(name)

where name(0) and name(1) contain the function name. Each name should be right justified with leading blanks, and the array name should be of integer mode. If the two names consist of a total of more than 10 characters, no action will be taken.

VERIFY. will examine a string of BCD characters, replace all parentheses by blanks, and store all groups of non-blank characters terminated by a blank in separate computer words. Scanning will terminate whenever a specified number of character groups are found, an "*" is encountered, or 76 consecutive blanks occur.

As was the case with GET2., each group of characters may not consist of more than 12 characters and the group will always be stored in two consecutive computer words and left justified. If a group consists of 6 or fewer characters, the second of the two words will contain only blanks.

Calling sequence:

execute verify.(h,arg,k)

The array h should contain the line to be analyzed, and arg will contain the character groups found in the line, each character group occupying 2 successive words. Groups are left justified with the beginning of each character group located in the word of the pair with the lowest value of the MAD subscript. When VERIFY. is called, k should be the number of character groups to be located. After the return from the subroutine, k will be equal to the number of groups actually found. h, arg, and k must all be of integer mode.

As an example, consider h to have been created by

vector values h=$a(xx) 2.5 3.1415926*$

If VERIFY. were called with k $\leq$ 4 arg would then contain:

Word	BCD contents
arg(0)	a____
arg(1)	xx____
arg(2)	2.5___
arg(3)	______
arg(4)	3.1415
arg(5)	926___

The values of k after execution would be 4.

VERIFY. is used as the basis of the system subroutines that require input from the console. The subroutine is called by GET. and

GET2. and by all of the subroutines for access to the MAP procedures, such as BASIS1., in order to analyze the input arguments.

3. Additional MAP System Subroutines

a. CARGS. will obtain the parameters of a CTSS "resume" command when called within the program that was "resumed". Each parameter must contain 6 or fewer characters.

Calling Sequence:

```
m = cargs.(arg)
```

arg will contain the parameters of the most recent "resume" command in the form previously given for GET. m will be equal to the number of parameters found, and both m and CARGS. must be of integer mode. The use of CARGS. is basic to transmitting the information supplied in the short form of a MAP command to the appropriate program when the subroutine CHAIN. is used.

b. CHAIN. allows one program to begin execution of a second program (not a subroutine), and upon completion of that program, to restore control to the first program at the point where execution had been temporarily interrupted.

Calling Sequence:

```
execute chain.(arg)
```

where arg is the name of an integer array containing a CTSS command, generally a "resume", followed by a fence of 7's. For example, to resume the program "print" in order to print g(x) between x = 2. and x = 4., arg would be generated with:

```
vector values arg=$resume print  g(x)    2.    4.$.7...7k
```

where 7...7k indicates a fence of 7's, i.e., 777777777777k. The array created by the vector values statement must be composed of computer words containing 6 characters, and each parameter must be right justified in a word. Therefore, arg, would be the array:

Word	BCD Contents
arg(0)	resume
arg(1)	_print
arg(2)	_____g
arg(3)	_____x
arg(4)	____2.

arg(5) ____4.

The following discussion of the operation of CHAIN. need not be completely understood in order to use the subroutine, but it is the basic mechanism for the operation of the MAP system.

When CHAIN. is called with the preceding arguments, for example, the subroutine will save the present core image and machine status in a file of temporary mode with name "CHAIN* SAVED", where * is the number of core images already saved by previous calls to CHAIN. The subroutine will then determine, in the case of the preceding calling sequence, if the file "print saved" exists; if it does, that program will then be resumed. The system command lists will be modified so that the most recently saved core image will be resumed when "print" terminates, i.e. control will return to the program which called "print". If "print saved" is not available, the system command lists are modified as in the previous case, and the following CTSS commands are given automatically:

```
vload    print    (libe)    sub
save     print
resume   print
```

where "sub" is the MAP System subroutine library. The created saved file will be available at the MAP temporary storage level until a MAP data restore command is given, or until the program is raised to the MAP permanent level by the use of the data update command. (The names of saved programs are not printed by the MAP data handling requests.)

It should be noted that only the MAP library and the usual CTSS library are searched as part of the loading procedure. Therefore, if display routines, AED subroutines, etc., are required, the program must exist in saved form or must be combined with the appropriate subroutines before CHAIN. is called. If a called program is not found in saved form or can not be loaded using the MAP and CTSS libraries and the user's files, the usual error comments will be printed by the loader. If an error during execution of the called program causes an error message to be printed, followed by a "COMMAND PLEASE", all of the saved files created by CHAIN. will be deleted.

c. FGET. will obtain from a MAP function the values having subscripts between a given set of integers and will load those values into a specified array.

Calling Sequence:

```
execute fget.(arg,f, min, max, del)
```

arg is the name of an array containing in the first computer word the first name of the function and in the second computer word the second name of the function. Both names must be right justified with leading blanks; f is the array into which the values of the function will be loaded. The value with subscript min will be loaded into f(0) and the value with subscript max into f(max-min); del will be set equal to the DEL of the function. arg, min and max must be of integer mode.

d. CHECK. This subroutine is useful in programs that request a range in the independent variable to be used in operating upon a function. The routine will determine if the range, as given in BCD form, consists of decimal or integer numbers, and if it is decimal, will determine whether the limits of the range are compatible with the DEL of the function. If "limit" is the value of one of the specified limits, the MAP criterion for compatibility is that the absolute value of [INTEGER (limit/DEL)*DEL - limit] is less than .01, where INTEGER ( ) indicates the integral part of the quotient. The routine returns, as binary numbers, the integer subscripts necessary to specify the desired range of the function.

Calling Sequence:

```
error=check.(arg,a,del,rmin, rmax)
```

The arg array should contain the range to be checked in one of two BCD forms: if a=0 it will be assumed that the BCD numbers are in the form produced by GET. while if a≠0 the form produced by GET2. will be expected. del, which must be supplied by the calling program, should be the DEL of the function for which the range is being provided; this value will not be used and can be 0 if the range is provided as BCD integers. rmin and rmax are the integer subscripts generated by the subroutine and specify the range of the function. error will equal 0 if the routine was successfully executed; 1 otherwise. CHECK does not determine whether the range is within the interval for which the function is defined. error, rmin, rmax, arg, a and CHECK must be of integer mode.

C. Procedures for Making Private Commands Public MAP Commands

In some cases a procedure written as a private command may be of sufficient interest to warrant making it a MAP command and thereby available to all MAP users. To be acceptable as a MAP command the program must satisfy five conventions:

a. If U does not provide all the necessary information on the command line, the program must either request the necessary parameters or assume values (or options), preferably those that would be used most often. No command can be included in MAP that fails to operate intelligently if U correctly answers the questions presented to him, and it should provide reasonable error messages if he answers the questions incorrectly.

b. All commands must have a short form so that U need not receive extensive informative print-outs if he wishes to avoid them.

c. All MAP functions must be considered to be of dimension 1000.

d. The program for the command must have the name of the command (or the first six characters of the name) as its first name. The second name should be BSS or SAVED, as appropriate. A BSS program for a command will be loaded in the MAP system by the CHAIN. subroutine, which uses the system command

vload name (libe) sub

where name is the name of the command and sub is the MAP System library.

e. The program should be as short as possible. In particular the author should consider breaking the program into separate parts if it is necessarily long and "chaining" from one part to the next. U should be expected to print or plot any numerical results after completion of the operation.

If a program satisfies all of these conventions, and appears to be of general utility, the author should furnish J. Brackett (13-5118, x6919) or R. Kaplow (13-5106, x3322) with the documentation regarding his command. If the command is considered to be a suitable addition to the MAP System, it will be added to the list of commands and will be made available to all users. The linkable file "MAP NEWS" will contain information about all new commands as they are added to the system.

This empty page was substituted for a blank page in the original document.

CS-TR Scanning Project
Document Control Form

Date : 12/11/95

Report # LCS-TR-24

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 110 (117-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☒ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:
☒ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): FOLLOW TITLE PAGE, i, iii, 63, 65, APPENDICES TITLE PAGE, 88, 103

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:

IMAGE MAP: (1-110) UNNUMBERED TITLE & BLANK PAGES, i, UNNUMBERED BLK, iii, UNNUMBERED BLK, UNNUMBERED INTRODUCTION, 2-63, UNNUMBERED BLK, 65, UNNUMBERED BLK, UNNUMBERED APPENDICES TITLE PAGE, UNNUMBERED BLK, 69-88, UNNUMBERED BLK, 90-103, UNNUMBERED BLANK. (111-117) SCANCONTROL, COVER, FUNDING AGENT, DOD, TRGT'S(3)

Scanning Agent Signoff:
Date Received: 12/11/95 Date Scanned: 1/4/96 Date Returned: 1/11/96

Scanning Agent Signature: Michael J. Cook

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP

3. REPORT TITLE
MAP: A System for On-Line Mathematical Analysis

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Description of the Language and User Manual

5. AUTHOR(S) *(Last name, first name, initial)*
Kaplow, Roy, Stephen L. Strong, and John W. Brackett

6. REPORT DATE
January 1966

7a. TOTAL NO. OF PAGES
104

7b. NO. OF REFS
0

8a. CONTRACT OR GRANT NO.
Office of Naval Research, Nonr-4102(01)

b. PROJECT NO.
Nr-048-189

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
MAC-TR-24

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES
Qualified requesters may obtain copies of this report from DDC.

11. SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
Advanced Research Projects Agency
3D-200 Pentegon
Washington, D. C.

13. ABSTRACT

A system for on-line mathematical analysis, called MAP, has been developed for use within the M.I.T. Compatible Time Sharing System. Taking advantage of the varied user-machine interactions which are possible, MAP provides a facility for handling complex analyses, data input and presentation of results without requiring any computer programming by the user. This report is a description of the language and a self-teaching user manual; it does not describe the techniques used to implement the system.

When given incomplete requests, the system will provide instructions regarding the use of its procedures and will ask for all the parameters, values and option decisions which may be required. If the requests are correct and sufficiently detailed, the computer will proceed directly to the calculations and, on command, present the results in graphical or typewritten form. Provisions have also been included to allow the expansion and personalization of the system in whatever manner is desired by individual users.

14. KEY WORDS

Computer
Graphical Input-Output
Man-Machine Interaction
Mathematical Analysis
Multiple-access Computers
Numerical Analysis
On-Line computer systems
Time-sharing
Time-shared computer systems

DD FORM 1 JAN 64 1473 (M.I.T.)

UNCLASSIFIED

Security Classification

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 1/4/1996

M.I.T. Libraries
Document Services