Office of Naval Research

Contract N00014-67-A-0298-0006 NR-372-012

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Grant NGR 22-007-068

THE MATRIX ALGEBRA PROGRAM A CONVERSATIONAL LANGUAGE FOR NUMERICAL MATRIX OPERATIONS -

PART I: USER'S MANUAL

VE RI TAS

By

P. M. Newbold

May 1968

Technical Report No. 561

Division of Engineering and Applied Physics
Harvard University • Cambridge, Massachusetts

Office of Naval Research

Contract N00014-67-A-0298-0006

NR - 372 - 012

National Aeronautics and Space Administration

Grant NGR-22-007-068

THE MATRIX ALGEBRA PROGRAM A CONVERSATIONAL LANGUAGE FOR NUMERICAL MATRIX OPERATIONS - PART I: USER'S MANUAL

By

P. M. Newbold

Technical Report No. 561

May 1968

The research reported in this document was made possible through support extended the Division of Engineering and Applied Physics, Harvard University by the U. S. Army Research Office, the U. S. Air Force Office of Scientific Research and the U. S. Office of Naval Research under the Joint Services Electronics Program by Contracts N00014-67-A-0298-0006, 0005, and 0008 and by the National Aeronautics and Space Administration under Grant NGR 22-007-068.

Division of Engineering and Applied Physics

Harvard University Cambridge, Massachusetts

THE

MATRIX ALGEBRA PROGRAM

A

CONVERSATIONAL LANGUAGE FOR

NUMERICAL MATRIX OPERATIONS –

PART I: USER'S MANUAL

By

P. M. Newbold

Division of Engineering and Applied Physics

Harvard University, Cambridge, Massachusetts

May 1968

CONTENTS

1. INTRODUCTION

THE MATRIX ALGEBRA PROGRAM, abbreviated to MAP, is an ARPAS program written for the SDS 940 direct access time-sharing system, to operate independently. The purpose of MAP is to carry out numerical operations on matrices. The language is designed to be extremely simple to use: matrices can be operated on with the same ease as scalar variables. No knowledge of Fortran or any other language is required.

The program is based on an earlier one, MM, developed recently by the author [1]; but is considerably more flexible. A completely new facility is the ability to create stored sequences of matrix operations for execution at a later time.

In the following sections of this manual several conventions are used. Characters that MAP types are underlined, while those that the user types are not. A carriage return is denoted by the symbol ® and a bell by the symbol Ⓑ . In the text any words or phrases printed in capitals are to be considered as definitive terms relating to MAP language.

The manual assumes that the user has a basic familiarity with the operation of the SDS 940 system such as may be gained from the standard manuals for the system. The user of MAP will find, however , a section devoted to the system operation as it relates to him, in the Appendix.

2. STRUCTURE OF THE LANGUAGE

MAP LANGUAGE CONSISTS of sequences of STATEMENTS typed in by the user. The typing of a STATEMENT will in general give rise to one or more of the following responses from MAP:

(a) a request for the next STATEMENT;

(b) computation of a matrix operation;

(c) a request for numerical data or additional information for that same STATEMENT;

(d) output of information or numerical data;

(e) generation of an error message (in the event of the user typing an illegal STATEMENT).

Statements

MAP LANGUAGE STATEMENTS are of two types:

DIRECT STATEMENTS - these are decoded and executed by MAP at the time that the user types them in. MAP requests new DIRECT STATEMENTS from the user by typing < ® and waiting.

INDIRECT STATEMENTS - these are decoded and stored but not executed at the time of typing in. Thus the user can create his own MAP PROGRAM. INDIRECT STATEMENTS are distinguished by the presence of a STATEMENT LABEL; and can only be typed by the user when MAP is in EDIT MODE.

Each STATEMENT consists of a COMMAND, denoting the operation to be executed; up to two ARGUMENTS, which are either VARIABLE names on which the operation is executed, or LABELS referring to other STATEMENTS; and, if the STATEMENT is INDIRECT, a LABEL.

The format of a STATEMENT is in general:

{n} : {a} , {b} , {command} ®

where {n} is the LABEL;
{a} , {b} are the ARGUMENTS;
{command} is the COMMAND.

Labels

A STATEMENT LABEL consists of any two-digit number between 01 and 63 inclusive, and is invariably followed by a colon. If MAP is not to confuse two INDIRECT STATEMENTS all LABELS must be distinguishable. It follows, therefore, that it is possible only to create a PROGRAM up to 63 STATEMENTS long.

There is no relation whatever between a STATEMENT LABEL and the position of that STATEMENT in the PROGRAM, the LABELS only being for the user convenience.

Arguments

THE ARGUMENTS OF any STATEMENT are of two types. Only ARGUMENTS of one type appear in any particular STATEMENT. STATEMENTS with ARGUMENTS of an incorrect type for the COMMAND are illegal. If the type of an ARGUMENT is numeric, then that ARGUMENT is a LABEL. It must therefore conform to the requirements of a LABEL, and must refer to some other INDIRECT

STATEMENT in existence at the time of execution of the STATEMENT containing that ARGUMENT.

If the type of the ARGUMENT is alphanumeric, then it is the name of a VARIABLE representing either a matrix or a scalar quantity. This VARIABLE name must have previously been defined in a LIST of VARIABLES. If the ARGUMENT is of this type, it must consist of two alphanumeric characters, the first of which must be alphabetic. The VARIABLE name OO is reserved and must not be redefined.

An ARGUMENT is invariably followed by a comma, whichever its type.

Commands

A COMMAND CONSISTS of the name of the operation it represents as defined in the later sections of this manual. Only the first three characters of the COMMAND are interpreted by MAP; A COMMAND is invariably followed by a carriage return, which also signifies to MAP that the user has completed the typing of the STATEMENT containing that COMMAND. Any COMMAND can be used both in DIRECT and INDIRECT STATEMENTS.

3. THE MODES OF OPERATION OF MAP

FOR THE CONVENIENCE of the user, the VARIABLE LIST and PROGRAM manipulation facilities of MAP are largely separated from the main execution facilities by the division of operation into two distinct MODES: EXECUTE MODE and EDIT MODE.

Execute Mode

IN EXECUTE MODE, MAP is either demanding and executing DIRECT STATEMENTS, or is executing a PROGRAM of INDIRECT STATEMENTS. If the first alternative is true, then MAP demands each new DIRECT STATEMENT by typing < ® and waiting. No LIST or PROGRAM entries can be created in this MODE.

MAP is normally in EXECUTE MODE and automatically transfers to EDIT MODE on the execution of certain STATEMENTS. This is the only way of entering EDIT MODE.

Edit Mode

IN EDIT MODE, MAP is either accepting VARIABLE names as elements in a LIST, or is creating a PROGRAM of INDIRECT STATEMENTS. In this MODE, DIRECT STATEMENTS are illegal. MAP demands each new entry by typing ___®. A return to EXECUTE MODE is effected by the user typing an asterisk instead of an entry. This also closes the LIST or PROGRAM under manipulation.

4. CREATION AND MANIPULATION OF A VARIABLE LIST

BEFORE THEIR USE in STATEMENTS denoting matrix or other operations, the user must define the names of VARIABLES that he wishes to use, by appending them with their dimensions to a LIST of VARIABLE names. Only one such LIST is generated by MAP; the user can add to this LIST or delete from it at any stage in his use of the language.

On first entry into the language, the LIST is empty except for the special VARIABLE name OO, which at this point represents a 10 by 10 matrix.

Adding to the Variable List

THE LIST OF VARIABLES can be added to at any stage of the user's computations by use of the STATEMENT

VARIABLES ®

On the execution of this STATEMENT, MAP automatically enters EDIT MODE, reopens the LIST and awaits entries. These entries have two forms:

MATRIX FORM: If the VARIABLE name to be defined represents a matrix the general form of the entry is

{v} = {m}, {n} ®

where {v} is any valid VARIABLE name;
{m} is the row dimension of the matrix;
{n} is the column dimension of the matrix.

SCALAR FORM: If the VARIABLE name to be defined represents a scalar, the general form of the entry may still be as above, of course setting {m} and {n} equal to unity. There is, however, the shortened form

{v} ®

where {v} is any valid VARIABLE name.

As explained previously, typing an asterisk instead of an entry closes the LIST and returns MAP to EXECUTE MODE.

EXAMPLE:

```
<  VARIABLES ®
   AA=2,2 ®
   B2 =4,1 ®
   XY=1,6 ®
   PP ®
   QQ ®
    *  ®
<
```

Deleting from the Variable List

THE LIST OF VARIABLES may be edited at any point in the user's computations by deleting entries, thereby releasing the VARIABLE STORE space associated with that entry. VARIABLE names may be deleted one at a time by the STATEMENT

{v}, OMIT ®

where {v} is any valid VARIABLE name.

MAP does not enter EDIT MODE for execution of this STATEMENT.

EXAMPLE: -continued from previous Example

```
<  B2, OMIT ®
<
```

Printing the Variable List

THE USER CAN print the LIST of VARIABLES at any time by using the STATEMENT

LIST ®

Scalars are printed as 1 by 1 matrices. LIST entries are printed out in the same order as they were typed in and stored (except for those deleted).

EXAMPLE: -continued from previous Example

```
<  LIST ®
OO        1Ø, 1Ø ®
AA        2,2 ®
XY        1,6 ®
PP        1,1 ®
QQ        1,1 ®
<
```

5. CREATION AND MANIPULATION OF A PROGRAM

MAP CAN STORE a PROGRAM consisting of a sequence of INDIRECT STATEMENTS typed in under EDIT MODE by the user. This PROGRAM may be edited by deleting or inserting further INDIRECT STATEMENTS, and may also be stored on a disk file. COMMENTS to be printed during execution may be added to this PROGRAM.

Only one PROGRAM can be in existence in the working store of MAP at any one time, although others may be in existence on disk files. As explained previously, each PROGRAM must have a length not exceeding 63 INDIRECT STATEMENTS. All VARIABLE names used in the PROGRAM must have previously been defined in the LIST of VARIABLES.

Before the first use of any STATEMENT creating sequences of INDIRECT STATEMENTS in a PROGRAM, that PROGRAM is said to be zero STATEMENTS long.

Appending Indirect Statements

TO APPEND NEW INDIRECT STATEMENTS to the end of an existing PROGRAM (or to start creating a new PROGRAM) the STAMENT

APPEND ®

is used. On execution of this STATEMENT, MAP automatically enters EDIT MODE, reopens the PROGRAM for manipulation and waits for entries. The user then types in the INDIRECT STATEMENTS he requires, as described in Section 2. When he reaches the end of the desired sequence, he then types an asterisk instead of an entry to close the PROGRAM and return MAP to EXECUTE MODE.

EXAMPLE:

```
< APPEND ®
___1Ø:AA, BB, ADD ®
___11:AA, CC, SUBTRACT ®
___12:OO, DD, MULTIPLY ®
___ * ®
<
```

Inserting Indirect Statements

TO INSERT A new sequence of INDIRECT STATEMENTS into an existing PROGRAM at any point, the STATEMENT

{n} , INSERT ®

is used. The LABEL-type ARGUMENT of the STATEMENT, {n}, is the LABEL of the INDIRECT STATEMENT in the PROGRAM immediately before which the insertion is to occur.

On execution of this STATEMENT, MAP automatically enters EDIT MODE, reopens the PROGRAM for manipulation and waits for entries. Procedure on the part of the user is now exactly the same as in the case of appending INDIRECT STATEMENTS.

For this STATEMENT to be valid {n} must be the LABEL for some INDIRECT STATEMENT already in existence at the time of execution of the STATEMENT causing the insertion.

EXAMPLE: -continued from previous Example

```
< 12,INSERT ®
__13:EE,OO,EQUATE ®
__14:EE,INVERT ®
__ * ®
<
```

Deleting Indirect Statements

TO DELETE A sequence of INDIRECT STATEMENTS from a PROGRAM the STATEMENT

{m} , {n} , DELETE ®

is used. INDIRECT STATEMENTS from that having LABEL {m} to that having LABEL {n} inclusive are deleted from the PROGRAM. MAP remains in EXECUTE MODE. The LABEL {m} must occur before the LABEL {n} in the PROGRAM at the time of execution of the STATEMENT causing the deletion, otherwise the STATEMENT is inexecutable.

A single INDIRECT STATEMENT is deleted by setting {m} equal to {n} and not by omitting either one of the ARGUMENTS of the STATEMENT causing the deletion.

EXAMPLE: -continued from previous Example

```
<  1Ø, 11DELETE ®
<
```

Printing the Program

THE USER CAN print the PROGRAM he has generated at any time by using the STATEMENT

PROGRAM ®

MAP prints out all INDIRECT STATEMENTS including their COMMENTS, if any, in the order in which they are stored.

EXAMPLE: -continued from previous Example

```
<  PROGRAM ®
13:EE, OO, EQU ®
14:EE, INV ®
12:OO, DD, MUL ®
<
```

Only the first three characters of the name of each COMMAND are printed out.

Attaching Comments

THE USER CAN cause a COMMENT to be attached to an INDIRECT STATEMENT, so that whenever that STATEMENT is executed during the execution of the PROGRAM containing the STATEMENT, the COMMENT is typed out by MAP for the benefit of the user. This facility may be used with advantage, for example in the case of an INDIRECT 'READ' STATEMENT. To cause a COMMENT to be attached, the STATEMENT

{n},COMMENT ®

is used. MAP then waits for the user to type the COMMENT he wishes: this COMMENT will be attached to the INDIRECT STATEMENT which has the LABEL {n}. Such an INDIRECT STATEMENT must of course be in existence at the time of execution of the STATEMENT attaching the COMMENT.

The COMMENT itself must be 72 or fewer characters long, and ends with the first carriage return. There is no other restriction on the form of the COMMENT.

If the user tries to attach more than one COMMENT to an INDIRECT STATEMENT, only the last COMMENT is attached. The user should note, however, that all such duplicated COMMENTS are kept in the COMMENT STORE area of MAP, and so reduce the available space.

To effectively erase the COMMENT of some INDIRECT STATEMENT, the user can attach a further COMMENT to that STATEMENT consisting merely of a carriage return.

EXAMPLE: -continued from previous Example

```
<   14,COMMENT ®
INVERSION OF THE MATRIX EE ®
<
```

6. PROGRAM STORAGE AND RETRIEVAL

AS BRIEFLY MENTIONED in the previous section a PROGRAM generated by the user can be saved on a disk file for reloading and use on a later occasion. MAP has a complementary pair of STATEMENTS which effect these operations. The user is referred to the Appendix for an explanation of the filing system for the SDS 940.

When the user saves a PROGRAM on disk, in addition to transcribing the PROGRAM itself onto a file, MAP also transcribes the LIST of VARIABLES and the COMMENTS for that PROGRAM. When the user reloads a PROGRAM from a file, the associated LIST of VARIABLES and COMMENTS are in addition reloaded. Any LIST, PROGRAM, or COMMENT existing before the reloading operation is erased. During the reloading operation, the user has the option of redefining the dimensions of the VARIABLES in the VARIABLE LIST.

No files except those created by MAP are acceptable to MAP for these operations. Only one whole PROGRAM and associated data can be placed on any one file. A second use of any file to <u>save</u> a PROGRAM erases any previous contents of that file. Reloading from a file does not affect that file in any way.

The VARIABLE STORE area of MAP is unaffected by PROGRAM storage and retrieval, so that the numerical values of a quantity remain unchanged. For correct reuse of this numerical data, however, the LIST of VARIABLES must be the same before and after the storage and retrieval operations.

Rules for Filenames

EACH FILE THAT the user stores a PROGRAM on must be given a name that follows certain rules. A valid FILENAME has the following general form:

/{name}/

where {name} is any sequence of characters up to a maximum length of nine characters.[1]

Saving a Program

THE USER CAN save a PROGRAM on a disk file by using the STATEMENT

SAVE ®

On execution of this STATEMENT, MAP first asks for the FILENAME of the file on which the PROGRAM is to be saved. The user types in any FILENAME he wishes, according to the rules already stated. If a file of that FILENAME already exists, MAP calls it an OLD FILE; or if not, a NEW FILE.

The user must now confirm execution of the operation by typing a period, whereupon MAP completes execution of the STATEMENT. Any other character typed instead of a period aborts the execution of the STATEMENT.

---

[1] There are certain names which are reserved for library files, but the user is not likely to encounter problems connected with this.

EXAMPLE: -continued from previous Example

< SAVE ®
FILE NAME / JUNK / OLD FILE . ®
<

Reloading a Program

THE USER CAN reload a PROGRAM from a disk file by using the STATEMENT

RESTORE ®

On execution of this STATEMENT, MAP first asks for the FILE-NAME of the file from which the PROGRAM is to be reloaded. The user types in the desired FILENAME according to the stated rules. If that FILENAME is acceptable, MAP proceeds by loading the PROGRAM, together with the associated COMMENTS and LIST of VARIABLES.

Next, for the benefit of the user, MAP prints out the LIST of VARIABLES, and then asks if the user requires the VARIABLES in the LIST to be redimensioned. The user either types

YES ®
or
NO ®

If the answer is in the negative, then execution of the STATEMENT is complete. If the answer is in the affirmative, execution now porceeds in a fashion somewhat similar to that of the 'VARIABLES' STATEMENT.

MAP types each of the VARIABLE names from the old LIST in turn, waiting for the user to enter the new dimensions for each name.

These can be entered in two ways:

MATRIX FORM: If the VARIABLE name is to represent a matrix, then the general form is

= {m}, {n} ®

where {m} is the row dimension of the matrix;
{n} is the column dimension of the matrix.

SCALAR FORM: If the VARIABLE name is to represent a scalar, the general form may still be the same as for the matrix case, of course setting {m} and {n} equal to unity. There is, however, the shortened form which merely consists of a carriage return.

When the whole of the LIST of VARIABLES has been treated in this way, execution of the STATEMENT is complete.

EXAMPLE: -continued from previous Example

```
< RESTORE ®
FILE NAME /JUNK / ®

VARIABLES USED ®

 OO  1Ø,1Ø ®
 AA  3,2 ®
 BB  3,2 ®
 CC  3,2 ®
 DD  2,5 ®
 EE  2,2 ®

REDIMENSION VARIABLES? ®

YES ®

 AA=4,2 ®
 BB=4,2 ®
 CC=4,2 ®
 DD=3,5 ®
 EE=3,3 ®

<
```

7. EXECUTION OF STATEMENTS AND PROGRAMS

THE EXECUTION OF DIRECT STATEMENTS is controlled by the user: the act of typing the STATEMENT also causes execution of that STATEMENT. The execution of a PROGRAM of INDIRECT STATEMENTS is controlled by MAP; the way in which control may be passed from the user to MAP for execution of a PROGRAM is discussed later in this section.

Execution of a STATEMENT or PROGRAM may be halted by pressing the 'ESCAPE' key once. After stopping execution, MAP waits for a new DIRECT STATEMENT. The 'ESCAPE' key is also used to make an exit from the language.[2]

The execution of a PROGRAM proceeds by NATURAL SEQUENCE; that is to say, the first-appearing STATEMENT is executed first, then the second and so on. The execution is <u>not</u> in order of the sequence of LABELS. Execution always proceeds by NATURAL SEQUENCE except when a STATEMENT changing the flow of execution is executed. The user can incorporate this type of STATEMENT into his PROGRAM, for example, to construct PROGRAM LOOPS.

Making allowance for these STATEMENTS, the execution of a PROGRAM is said to proceed by LOGICAL SEQUENCE. When there are no further STATEMENTS to be executed in the LOGICAL SEQUENCE, MAP automatically returns control of execution to the user, and waits for a new DIRECT STATEMENT to be typed.

Branches and Program Loops

A PROGRAM LOOP is a sequence of STATEMENTS so constructed as to be executed repetitively by MAP until some condition is fulfilled. MAP contains two flow-changing operations with which the user may construct PROGRAM LOOPS, or other conditional structures.

[2] The user is referred to the Appendix for an explanation of this procedure.

The 'BRANCH' STATEMENT - Execution of the STATEMENT

{n}, BRANCH ®

causes the flow of execution to be interrupted. Instead of executing the next STATEMENT in the NATURAL SEQUENCE, MAP locates the INDIRECT STATEMENT having LABEL {n}. The flow of execution then restarts with this STATEMENT, continuing again in the NATURAL SEQUENCE until another flow-changing operation is met.

The 'SKIP' STATEMENT - Execution of the STATEMENT

{a}, {b}, SKIP ®

where {a} and {b} are VARIABLE names representing scalars, causes MAP to take one of two different courses of action. If {a} > {b} then execution continues normally in the NATURAL SEQUENCE. If {a} ≤ {b} then MAP ignores the next STATEMENT in the NATURAL SEQUENCE, and the flow of execution resumes with the one after next.

In the following Example, it is shown how these two flow-changing operations are combined to form a PROGRAM LOOP.

EXAMPLE:

```
   1Ø:AA, NULL ®
 ↗ 11:AA, BB, ADD ®
/  12:OO, DETERMINANT ®
|  13:CC, OO, SKIP ®   ---\  CC > OO  -----\ CC ≤ OO
 \ 14:11, BRANCH ®    <--/                |
   15:OO, PRINT ®     <------------------/
```

Entering a Program for Execution

AT THE START of this section it was stated that control must be transferred from the user to MAP for the execution of a PROGRAM. This is achieved by using the branching operation in its DIRECT form. Obviously execution of the PROGRAM can be started at any STATEMENT by specifying the LABEL for that STATEMENT as the ARGUMENT of the DIRECT 'BRANCH' STATEMENT.

The following Example shows how the user would direct MAP to execute the PROGRAM illustrated in the previous Example, starting with the first STATEMENT.

EXAMPLE: -continued from previous Example

```
<__1Ø, BRANCH ®
(execution follows)
```

8. STANDARD MATRIX OPERATIONS OF MAP

MOST OF THE standard operations which are carried out on matrices or vectors are available in MAP. The major exceptions are those operations involving partitioning of matrices and the manipulation of their submatrices.

The compatibility of the matrices involved in each operation is checked before the execution of that operation. An error message is given and execution halted if an incompatibility is detected.

MAP matrix operations are divided into two classes. CLASS A operations are those in which the result of the operation is placed in the standard output matrix denoted by the VARIABLE name OO. CLASS B comprises the remainder of the matrix operations.

Class A Operations

THE RESULTS OF all CLASS A operations are placed in the standard output matrix denoted by the VARIABLE name OO. The VARIABLE takes a dimension appropriate to the operation. For this reason, when the LIST of VARIABLES is printed, the dimensions of OO shown will be those implied by the result of the last operation involving OO.

The VARIABLE OO may be used in the same way as any other VARIABLE the user defines, and may appear as an ARGUMENT in any STATEMENT[3].

The reuse of OO by any CLASS A operation automatically erases the previous value of OO.

CLASS A operations comprise the following STATEMENTS:

{a}, {b}, ADD ®	OO = {a} + {b}
{a}, {b}, SUBTRACT ®	OO = {a} - {b}
{a}, {b}, MULTIPLY ®	OO = {a} · {b}
{a}, {b}, SCALAR MULTIPLY ®	OO = {a} · {b} ({a} scalar)
{a}, NEGATE ®	OO = -{a}
{a}, TRANSPOSE ®	OO = $\{a\}^T$
{a}, INVERT ®	OO = $\{a\}^{-1}$
{a}, DETERMINANT ®	OO = Det. {a}
{a}, DIAGONAL SUM ®	OO = Trace {a}

In addition the operation of finding eigenvalues and vectors is considered a CLASS A operation. The values and vectors found are not stored, but are printed during execution of the operation. The VARIABLE OO is used for temporary storage purposes and is left as a square matrix of the same dimension as the matrix operated upon.

{a}, EIGENVALUE ®	OO = ∅

[3] The VARIABLE OO may not appear in the 'OMIT' STATEMENT because it cannot be deleted from the LIST of VARIABLES.

Class B Operations

CLASS B OPERATIONS have no common distinguishing feature. They comprise the following STATEMENTS:

{a}, NULL ®	{a} = ∅	
{a}, {b}, EQUATE ®	{a} = {b}	({b} unchanged)

9. INPUT AND OUTPUT OF NUMERICAL DATA

SO FAR NO mention has been made of the input and output of numerical values of VARIABLES. MAP contains four STATEMENTS which take care of these operations. Two are for the input and output of data via the teletype console. Two are for saving data on disk files.

Teletype Input and Output

THE TELETYPE INPUT and output of the values of matrices has the same general format as a matrix written out by hand. MAP uses one fixed format for the output of data; but the input of data by the user is relatively free-form.

TELETYPE OUTPUT - On execution of the STATEMENT

{a}, PRINT ®

MAP prints the values of the elements of the VARIABLE {a}. The values of the matrix are printed out row by row in the standard way. MAP prints out the values to four significant figures. Each value is normalized to between 1.000 and 9.999 and an exponent part is printed immediately below the fractional part of the element.

EXAMPLE:

for matrix $AA = \begin{bmatrix} 10 & 2 \\ 9 & 12 \end{bmatrix}$

```
<  AA,PRINT ®
   1.ØØØ   -2.ØØØ
   E+Ø1     E+ØØ

   9.ØØØ    1.2ØØ
   E+ØØ     E+Ø1

<
```

TELETYPE INPUT - On execution of the STATEMENT

{a},READ ®

MAP waits for the values of the elements of the VARIABLE {a} to be typed in by the user. These are typed in row by row in the same order as values written out by hand. There are relatively few restrictions that the user has to consider.

The value of each element, if it is not the final element of a row of the matrix, must be terminated by a space. If the user has reached the right-hand side of the paper without completing a row, he may terminate a value by a line feed character instead of a space. MAP then returns the carriage to the left-hand side of the paper and the user may continue as before.

The value of the final element of the row of the matrix must be terminated by a carriage return. At the beginning of each row of the matrix, MAP gives a bell.

The value of each element may be typed in as an integer or a decimal, and with or without an exponent part. The fractional part may have the decimal point in any position, and must be less than 12 digits long. Leading spaces in the fractional part are ignored.

The exponent part, if present, immediately follows the fractional part with no intervening spaces. It consists of the letter E followed by a signed integer. Spaces leading the signed integer are ignored. The exponent must be such that the value of the element lies within the range 10^{-77} to 10^{77}, or is zero. Larger values will be truncated.

EXAMPLE:

for the matrix $AA = \begin{bmatrix} -0.64531 & 3.684 \times 10^{8} \\ 16543.1 & 8765.2 \end{bmatrix}$

```
<  AA,READ ®
-Ø.64531 3.684E+8 ®
1.65431E+4  8765.2 ®
<
```

Section 10 contains a procedure to be followed if the user should type erroneous data.

File Storage of Data

JUST AS A PROGRAM may be stored on a disk file for reloading on a later occasion, so may the user also store the values of a VARIABLE. The operations themselves are somewhat similar.

MAP contains two operations for storing and loading the values of a VARIABLE. The file concerned in the operations is given a FILENAME which conforms to exactly the same rules as the FILENAME of a file used for PROGRAM storage. The user is referred to Section 6 for these rules.

In the absence of any special arrangements, only those data files created by MAP are acceptable to MAP.[4]

STORING ON A FILE - On execution of the STATEMENT

{a}, STORE ®

> MAP stores the values of the elements of the VARIABLE {a} on a disk file. The exact procedure is as follows. First, MAP demands a FILENAME from the user and waits for it to be typed in. If a file of the FILENAME already exists, MAP calls it an OLD FILE; or if not, a NEW FILE.
>
> In either case MAP then waits for the user to type a confirmatory period, and then completes execution of the STATEMENT. Any other character typed will cause MAP to ask for the FILENAME again.
>
> If the file is an OLD FILE, the previous contents of the file are erased by execution of this STATEMENT.

---

[4] Details of a routine which may be used in SDS 940 FORTRAN programs to create files compatible with MAP files, will be found in the MAP REFERENCE MANUAL. [2]

EXAMPLE:

< AA, STORE ®
FILE NAME /TRASH/ OLD FILE. ®
<

LOADING FROM A FILE - On execution of the STATEMENT

{a}, LOAD ®

MAP loads the values of the elements of the VARIABLE {a} from a disk file. First MAP demands the FILE-NAME of the file containing the values, and waits for the user to type it in. If the FILENAME is acceptable, MAP completes execution of the STATEMENT.

The file itself remains unaffected by this operation.

EXAMPLE:

< AA, LOAD ®
FILE NAME /TRASH/ ®
<

10. ERROR MESSAGES AND PROCEDURES

MAP IS PROVIDED with a comprehensive error detection scheme. Any mistake which could cause MAP to malfunction is detected, and an error message printed out to the user. The subsequent action varies with the type of error.

There are three types of errors:

(a) EXECUTION ERRORS - those errors occurring during the execution of a STATEMENT ;

(b) TELETYPE ERRORS - those errors detected as the user types information or numerical data;

(c) STORAGE ERRORS - those errors due to the over-filling of storage space.

In this section, each of these three categories is dealt with in turn. The possible causes of each error message given by MAP are listed; and the subsequent action to be taken by the user is explained where appropriate.

Execution Errors

MAP SIGNALS AN EXECUTION ERROR if it finds it impossible to continue executing a STATEMENT, or if it cannot decide which STATEMENT to execute next.

If the error occurs during the execution of a DIRECT STATEMENT, execution is interrupted, and an error message printed out. If it occurs during the execution of an INDIRECT STATEMENT having LABEL {n}, execution is interrupted, and the following message is printed:

STOP IN STATEMENT {n} ®

This is followed by the printing of the error message. If execution is deleted, then MAP also terminates execution of the whole PROGRAM.

The following error messages may appear.

INCOMPATIBLE MATRIX ®

Some matrix operations cannot be executed if their operands have incompatible dimensions. MAP checks the

ARGUMENTS of the corresponding STATEMENTS for compatibility before execution.

This message is printed if an incompatibility is discovered. Execution of the STATEMENT is deleted, and MAP waits for the user to type a new DIRECT STATEMENT.

LABEL UNDEFINED ®

This message is given if the LABEL-type ARGUMENT of a STATEMENT refers to a LABEL which does not exist at execution time. Execution of the STATEMENT is deleted, and MAP waits for the user to type a new DIRECT STATEMENT.

DETERMINANT ZERO ®

This message is given if MAP tries to find the determinant of a singular matrix. The result of the operation is set to zero, and execution is resumed.

MATRIX SINGULAR - RANK = {n} ®

This message is given if MAP tries to find the inverse of a singular matrix. The result is set to a null matrix of the same dimension as the operand, and execution is resumed.

COMPUTATION FAILURE ®

This message is given during the calculation of the eigenvalues of a matrix, if calculation of the current eigenvalue cannot proceed. This is most likely to occur in ill-conditioned matrices. The calculation of further eigenvalues is terminated, but execution continues as if the operation had been successfully completed.

Teletype Errors

MAP signals a TELETYPE ERROR if it cannot understand what the user is typing, or if some other rule related to teletype input is contravened. MAP will interrupt the user when he types the first carriage return after the occurrence of the error, and print out an error message.

Generally the user must retype the line containing the error. He then continues from the point of interruption.

WHAT ? ®

This message is given whenever the user types something unintelligible to MAP.

If the user is typing a DIRECT STATEMENT, the error may be due to an illegal character, or the user may have typed the wrong number of ARGUMENTS. MAP waits for the user to type the STATEMENT again.

If the user is typing an INDIRECT STATEMENT in EDIT MODE, the error may have a similar origin. Alternatively, the user may have given the STATEMENT the same LABEL as a previous INDIRECT STATEMENT. MAP deletes the STATEMENT from the PROGRAM, and waits for the user to retype it.

If the user is typing an entry in the LIST of VARIABLES, the error is usually due to an illegal character. MAP deletes the entry from the LIST, and waits for the user to retype it.

If the user is typing in the values of the elements of a VARIABLE, the error is usually due to an illegal character. MAP deletes the values from all the elements in the row of the matrix containing the error, and waits for the user to retype the row.

If the user is redimensioning an entry in the VARIABLE LIST during the execution of a 'RESTORE' STATEMENT, MAP deletes the new dimensions, and waits for the user to retype them.

If the user is typing a FILENAME, the error is due either to a bad character, or to the specification of an unacceptable file. MAP rejects the FILENAME, and asks the user to type a new one.

VARIABLE UNDEFINED ®

This message is given if the user types a STATEMENT containing a VARIABLE name not previously defined in the LIST of VARIABLES. If the STATEMENT is DIRECT, MAP waits for a new DIRECT STATEMENT to be typed. If the STATEMENT is INDIRECT, MAP deletes it from the PROGRAM and waits for it to be retyped.

ILLEGAL VARIABLE ®

This message is given if the user types a STATEMENT with the wrong type of ARGUMENT. The procedure followed is the same as for the previous message.

If the message occurs while the user is typing an entry in the VARIABLE LIST, then a VARIABLE name has been duplicated. MAP deletes the entry from the LIST, and waits for the user to retype it.

MATRIX IS OVERSIZE ®

This message is given if the user defines a dimension of A VARIABLE larger than the maximum permissible. This may happen either when the user is typing in an entry in the LIST of VARIABLES, or during the redimensioning of the LIST while reclosing a PROGRAM.

If the user is typing a list entry, MAP deletes the entry and waits for the user to retype it.

If the user is redimensioning an entry, MAP deletes the new dimensions, and waits for the user to retype them.

Storage Errors

STORAGE ERRORS OCCUR when the user tries to use more space than is available in MAP. These errors are not likely to be encountered very often.

NO MORE STATEMENTS ®

This message is given when the user has reached the maximum allowable length of PROGRAM. It is printed out immediately after the user has typed the 63rd INDIRECT STATEMENT. This last STATEMENT is stored, but the user may type no more in. MAP closes the PROGRAM, returns to EXECUTE MODE, and waits for the user to type a new DIRECT STATEMENT.

TOO MANY VARIABLES ®

This message is given when the user tries to define more than 59 VARIABLES. It is printed immediately after the user defines the 60th VARIABLE. The VARIABLE is not entered in the LIST; instead MAP closes the LIST, returns to EXECUTE MODE, and waits for a new DIRECT STATEMENT.

VARIABLE STORE FULL ®

This message is given during the definition of a LIST of VARIABLES, or during the redimensioning of a LIST while a 'RESTORE' STATEMENT is being executed.

It means that the total number of elements in all the VARIABLES in the VARIABLE LIST, including OO, has exceeded 4000 . MAP deletes the entry in the LIST that the user last typed in, and waits for a new entry. This new entry must not violate the rule or the error message will be repeated.

COMMENT STORE FULL ®

This message is given during the execution of a 'COMMENT' STATEMENT if the total number of characters in all the COMMENTS including their terminating carriage returns exceeds 3000. After giving the error message, MAP terminates execution of the 'COMMENT' STATEMENT.

The contents of the COMMENT STORE cannot be erased except by making an exit from the language.

11. UPPER LIMITS ON STORAGE SPACE

THE MAXIMUM STORAGE space available in MAP for any purpose depends on two factors ; the structure of the language, and the storage space available on the user's own SDS 940 system.

The maximum lengths of the LIST of VARIABLES and of the PROGRAM depend on the structure of the language, and cannot be increased.

The maximum size of a matrix cannot easily be increased.

The sizes of the VARIABLE STORE or COMMENT STORE can be increased relatively easily. Instructions for doing this may be found in the MAP REFERENCE MANUAL.

The following are the maximum dimensions associated with MAP as it stands at the time of publication.

MAXIMUM SIZE OF VARIABLE-

Matrices of dimension not exceeding 10 by 10 .

MAXIMUM LENGTH OF VARIABLE LIST-

Not more than 60 VARIABLES, including OO, but depending on:

SIZE OF VARIABLE STORE-

A total of 4000 elements in all VARIABLES including OO .

LENGTH OF PROGRAM-

Not more than 63 INDIRECT STATEMENTS.

LENGTH OF EACH COMMENT-

Not more than 72 characters, including the terminal carriage return.

SIZE OF COMMENT STORE-

Not more than 3000 characters total, including those COMMENTS overwritten with later ones by the user.

12. AN EXAMPLE

THE EXAMPLE GIVEN in this section demonstrates some of the more important properties of MAP. An actual printout from the user's teletype console is reproduced.

The example is the calculation of a positive integer power of a matrix by the method of repeated multiplication.

First a PROGRAM for doing this calculation for any 3 by 3 matrix is created. This is executed to calculate

$$A^2 \quad \text{where} \quad A = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 4.0 & 5.0 & 6.0 \\ 7.0 & 8.0 & 9.0 \end{bmatrix}$$

The PROGRAM is saved on a disk file, and then reloaded so as to allow this calculation for 2 by 2 matrices. Lastly the PROGRAM is re-executed to calculate

$$A^1 \quad \text{where} \quad A = \begin{bmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{bmatrix}$$

By deleting the PROGRAM from the PROGRAM STORE between the saving and reloading operations, it is shown that storage on file has actually taken place. The PROGRAM remains on the file after log-out.

The processes of log-in, log-out, and loading of MAP are explained in the Appendix.

```
HARVARD TIME SHARING SYSTEM (D00-H19): 3-14-68
MAY 1, 1968   2:11 P.M.

ACCOUNT: 110                          .... log-in to system
PASSWORD:
NAME: AGRAWALA.

@DDT.
;T   /P 3B/    20047                  ..... loading of MAP
WORK;G


MATRIX MANIPULATOR  (110-3) MAY  1968

< VARIABLES
  XX=3,3
  YY=3,3                              .... creation of LIST
  N1                                       of VARIABLES
  N2
  UN
  *
< APPEND
  10:XX,READ
  11:N1,READ
  12:UN,READ
  13:YY,XX,EQUATE
  14:N2,UN,EQUATE                     .... creation of
  15:N2,UN,ADD                             PROGRAM
  16:N2,00,EQUATE
  17:N2,N1,SKIP
  18:22,BRANCH
  19:XX,YY,MULTIPLY
  20:YY,00,EQUATE
  21:15,BRANCH
  22:XX,PRINT
  23:YY,PRINT
  *
< 10,COMMENT
READ IN MATRIX
< 11,COMMENT
READ IN POWER OF MATRIX (AS DECIMAL)
< 12,COMMENT
READ IN UNIT SCALAR
< 22,COMMENT                          .... attachment of
MATRIX =                                   COMMENTS
< 23,COMMENT
POWER OF MATRIX =
< LIST
 00    10,10
 XX    3,3
 YY    3,3                            .... printout of LIST
 N1    1,1                                 of VARIABLES
 N2    1,1
 UN    1,1
< PROGRAM
```

```
READ IN MATRIX
10:XX, REA
READ IN POWER OF MATRIX (AS DECIMAL)
11:N1, REA
READ IN UNIT SCALAR
12: UN, REA
13:YY, XX, EQU
14:N2, UN, EQU
15:N2, UN, ADD
16:N2, 00, EQU
17:N2, N1, SKI
18: 22, BRA
19:XX, YY, MUL
20:YY, 00, EQU
21: 15, BRA
MATRIX =
22:XX, PRI
POWER OF MATRIX =
23:YY, PRI
```

.... printout of PROGRAM with COMMENTS

```
< 10, BRANCH
```

.... start of execution

```
READ IN MATRIX
1.0 2.0 3.0
4.0 5,0 6.0


WHAT?

4.0 5.0 6.0
7.0 8.0 9.0
```

.... typing of input data (note corrected error)

```
READ IN POWER OF MATRIX (AS DECIMAL)
2.0
READ IN UNIT SCALAR
1.0
MATRIX =
   1.000   2.000   3.000
    E+00    E+00    E+00

   4.000   5.000   6.000
    E+00    E+00    E+00

   7.000   8.000   9.000
    E+00    E+00    E+00
```

.... printout of results

```
POWER OF MATRIX =
   3.000   3.600   4.200
    E+01    E+01    E+01

   6.600   8.100   9.600
    E+01    E+01    E+01

   1.020   1.260   1.500
    E+02    E+02    E+02
```

.... end of execution

```
< SAVE
```

```
FILE NAME  /FILE1/ OLD FILE.
< 10, 23, DELETE
< PROGRAM
< RESTORE
```

....save PROGRAM on file and delete from core

```
FILE NAME  /FILE1/
```

.... Reload PROGRAM from file

```
VARIABLES USED

 OO    1, 1
 XX    3, 3
 YY    3, 3
 N1    1, 1
 N2    1, 1
 UN    1, 1

REDIMENSION VARIABLES?

YES
 XX=2, 2
 YY=2, 2
 N1
 N2
 UN
```

.... redimensioning of VARIABLE LIST

```
< PROGRAM
READ IN MATRIX
10:XX, REA
READ IN POWER OF MATRIX (AS DECIMAL)
11:N1, REA
READ IN UNIT SCALAR
12: UN, REA
13:YY, XX, EQU
14:N2, UN, EQU
15:N2, UN, ADD
16:N2, OO, EQU
17:N2, N1, SKI
18: 22, BRA
19:XX, YY, MUL
20:YY, OO, EQU
21: 15, BRA
MATRIX =
22:XX, PRI
POWER OF MATRIX =
23:YY, PRI
```

.... printout of reloaded PROGRAM

```
< 10, BRANCH
```

.... start execution again

```
READ IN MATRIX
1.Ø 2.Ø
3.Ø 4.Ø
READ IN POWER OF MATRIX (AS DECIMAL)
1.Ø
READ IN UNIT SCALAR                      .... typing of input data
1.Ø
MATRIX =
  1.ØØØ   2.ØØØ
   E+ØØ    E+ØØ

  3.ØØØ   4.ØØØ
   E+ØØ    E+ØØ
                                         .... printout of results
POWER OF MATRIX =
  1.ØØØ   2.ØØØ
   E+ØØ    E+ØØ

  3.ØØØ   4.ØØØ
   E+ØØ    E+ØØ

<                                        .... end of execution


@LOGOUT AGRAWALA.
MAY 1, 1968   2:25 P.M.

TIME USED                                .... log-out from system
CPU: ØØ:ØØ:Ø6
CONNECT: ØØ:14
X
```

APPENDIX

THIS APPENDIX IS devoted to an explanation of the operation of the SDS 940 as it affects the MAP user. For a more detailed treatment, the user should consult the SDS system manuals.

The instructions given here only apply to the SDS 940 system in operation at the Harvard University Computation Laboratory. While other 940 systems are essentially similar, there may be minor differences in operating procedure.

Logging In

ON BEING CONNECTED with the system, the user is asked for his account number, password, and name. The following sequences are typed:

HARVARD TIME SHARING SYSTEM (DØØ-H19) : 3-14-68 ®
{date} {time} ®

ACCOUNT: {user's account number} ®
PASSWORD: {user's password} ®
NAME: {user's name} . ®
@

If the system accepts the user's credentials, he is logged in. The symbol @ signifies that the system is under control of the Executive, and is waiting for a command. While in the Executive, the user can perform various operations. These are explained in the TERMINAL USER'S GUIDE for the HARVARD system. [4] At this point, the user can load the MAP processor.

Loading the MAP Processor

BEFORE MAP CAN be used, the MAP language processor must be loaded into core, and control delegated to it from the Executive. There are two different methods of loading MAP.

METHOD I. This is the preferred method of loading the processor. A copy of the processor is held in the LIBRARY of the system on the file MAP. . To load the file and delegate control to MAP, the user types:

```
@EXECUTE MAP. (LIBRARY) . ®
```

Note that the system completes typing the word 'EXECUTE' after the first three characters.

METHOD II. This method can only be used if the user possesses his own copy of the processor on a BINARY file. Such a file can be loaded by calling in the DDT subsystem[5].

Suppose that the user has a copy of the processor on the BINARY file /MAP/. Then the user loads MAP and delegates control to it by typing:

```
@DDT. ®
;T  /MAP/  20047 ®
WORK;G ®
```

After either method of loading, MAP types

```
MATRIX MANIPULATOR (110-3) MAY 1968 ®
<
```

and is ready for the user to type his first DIRECT STATEMENT.

[5] For details on DDT see the SDS 940 DDT REFERENCE MANUAL. [3]

Exit from MAP Language

AS EXPLAINED EARLIER, at any stage, the user can stop execution of a STATEMENT or PROGRAM by pressing the 'ESCAPE' key. MAP then waits for a new DIRECT STATEMENT to be typed.

The 'ESCAPE' key also controls the exit from MAP language. If the user presses the dey several times in rapid succession, or in conjunction uses the 'REPEAT' key, the system will return to the Executive and type @.

To return to MAP again, the user can follow the loading procedure once more. Note that in this case the new copy of the processor overwrites the old one in core, and the user cannot carry on from the point of exit. Alternatively, if MAP had been loaded by Method II, the old processor could be reinstated by typing

```
@CONTINUE DDT. ®
WORK;G ®
```

MAP starts by typing the usual sequence; and retains the LIST of VARIABLES, PROGRAM, and COMMENTS, if any, created before the exit.

Logging Out

IF THE USER wishes to log out he first escapes to the Executive. He then types:

```
@LOGOUT {user's name} . ®
 {date}  {time} ®

TIME USED ®
CPU: {hrs:mins:secs} ®
CONNECT: {hrs:mins} ®
```

In the above sequence, the CONNECT time is the time for which the user was logged in; and the CPU time is the actual time taken by the system on the user's work.

The Filing System

USER'S PROGRAMS AND data can be stored on disk files by using the appropriate commands in the language in which the user is working. These files can be defined and manipulated as complete entities in the Executive. Several Executive commands useful for the manipulation of MAP files are given here. The user will find a comprehensive list in the TERMINAL USER'S GUIDE. [4]

To delete a file:

@DELETE /{name}/ . ®

To rename a file:

@RENAME /{name 1}/ AS /{name 2}/ . ®

To copy one file to another:

@COPY /{name 1}/ TO /{name 2}/ . ®

Note that since all MAP files are BINARY files, the contents cannot be typed by using the command

@COPY /{name}/ TO TEL. ®

To make a file Public, Readout only:

@DEFINE /{name}/ AS PUBLIC. ®

To make a file Private:

@DEFINE /{name}/ AS PRIVATE. ®

All MAP files are Private on creation.

REFERENCES

[1] P. M. Newbold and A. K. Agrawala. "Two Conversational Languages for Control Theoretical Computations in the Time-Sharing Mode." Harvard University Technical Report TR 546, November 1967.

[2] P. M. Newbold. "M.A.P. - A Conversational Language for Numerical Matrix Operations. Part II: Reference Manual." Harvard University Technical Report TR 562, June 1968.

[3] "DDT Reference Manual for SDS-940 Time-sharing Computer Systems." Scientific Data Systems Publication 90 11 13A, May 1967.

[4] "Harvard Computing Center Time-sharing Terminal User's Guide". Harvard University Computing Center, January 1968.

Joint Services Electronics Program

N00014-67-A-0298-0006, 0005, and 0008

Academy Library (DFSLB)
U. S. Air Force Academy
Colorado Springs, Colorado 80912

AEDC (ARO, INC)
Attn: Library/Documents
Arnold AFB, Tenn. 37389

Aeronautics Library
Graduate Aeronautical Laboratories
California Institute of Technology
1201 E. California Blvd.
Pasadena, California 91109

Aerospace Corporation
P. O. Box 95085
Los Angeles, Calif. 90045
Attn: Library Acquisitions Group

Airborne Instruments Laboratory
Deerpark, New York 11729

AFAL (AVTE/R. D. Larson)
Wright-Paterson AFB
Ohio 45433

AFCRL (CRMXLR)
ARCRL Research Library, Stop 29
L. G. Hanscom Field
Bedford, Mass. 01731

AFETR (ETLIG - 1)
STINFO Officer (for library)
Patrick AFB, Florida 32925

AFETR Technical Library
(ETV, MU-135)
Patrick AFB, Florida 32925

AFFTC (FRBPP-2)
Technical Library
Edwards AFB, Calif. 93523

APGC (PHBPS-12)
Eglin AFB
Florida 32542

ARL (ARIY)
Wright-Paterson AFB
Ohio 45433

AUL3T-9663
Maxwell AFB
Alabama 36112

Mr. Henry L. Bachman
Assistant Chief Engineer
Wheeler Laboratories
122 Cuttermill Road
Great Neck, N. Y. 11021

Bendix Pacific Division
11600 Sherman Way
North Hollywood, Calif. 91605

Colonel A. D. Blue
RTD (RTTL)
Bolling AFB
Washington, D. C. 20332

California Institute of Technology
Pasadena, California 91109
Attn: Documents Library

Carnegie Institute of Technology
Electrical Engineering Dept.
Pittsburg, Pa. 15213

Central Intelligence Agency
Attn: OCR/DD Publications
Washington, D. C. 20505

Chief of Naval Operations
OP-07
Washington, D. C. 20350 [2]

Chief of Naval Research
Department of the Navy
Washington, D. C. 20360
Attn: Code 427 [3]

Commandant
U. S. Army and General Staff College
Attn: Secretary
Fort Leavenworth, Kansas 66370

Commander
Naval Air Development and
Material Center
Johnsville, Pennsylvania 18974

Commanding General
Frankford Arsenal
Attn: SMUFA-L6000 (Dr. Sidney Ross)
Philadelphia, Pa. 19137

Commandant
U. S. Army Air Defense School
Attn: Missile Sciences Div. C and S Dept.
P. O. Box 9390
Fort Bliss, Texas 79916

Commander
U. S. Naval Air Missile Test Center
Point Magu, California 93041

Commanding General
Attn: STEWS-WS-VT
White Sands Missile Range
New Mexico 88002 [2]

Commanding General
U. S. Army Electronics Command
Fort Monmouth, N. J. 07703
Attn: AMSEL-SC
RD-D
RD-G
RD-GF
RD-MAT
XL-D
XL-E
XL-C
XL-S
HL-D
HL-CT-R
HL-CT-P
HL-CT-L
HL-CT-O
HL-CT-I
HL-CT-A
NL-D
NL-A
NL-P
NL-R
NL-S
KL-D
KL-E
KL-S
KL-T
VL-D
WL-D

Commanding General
U. S. Army Material Command
Attn: AMCRD-RS-DE-E
Washington, D. C. 20315

Commanding General
U. S. Army Missile Command
Attn: Technical Library
Redstone Arsenal, Alabama 35809

Commanding Officer
Naval Avionics Facility
Indianapolis, Indiana 46241

Commanding Officer
U. S. Army Limited War Laboratory
Attn: Technical Director
Aberdeen Proving Ground
Aberdeen, Maryland 21005

Commanding Officer
U. S. Army Materials Research Agency
Watertown Arsenal
Watertown, Massachusetts 02172

Commanding Officer
U. S. Army Security Agency
Arlington Hall
Arlington, Virginia 22212

Commanding Officer and Director
U. S. Naval Underwater Sound Lab.
Fort Trumbull
New London, Conn. 06840

Defense Documentation Center
Attn: TISIA
Cameron Station, Bldg. 5
Alexandria, Virginia 22314 [20]

Det No. 6, OAR (LOOAR)
Air Force Unit Post Office
Los Angeles, Calif. 90045

Director
Advanced Research Projects Agency
Department of Defense
Washington, D. C. 20301

Director for Materials Sciences
Advanced Research Projects Agency
Department of Defense
Washington, D. C. 20301

Director
Columbia Radiation Laboratory
Columbia University
538 West 120th Street
New York, New York 10027

Director
Coordinated Science Laboratory
University of Illinois
Urbana, Illinois 61803

Director
Electronics Research Laboratory
University of California
Berkeley, California 94720

Director
Electronic Sciences Laboratory
University of Southern California
Los Angeles, California 90007

Director
Microwave Laboratory
Stanford University
Stanford, California 94305

Director - Inst. for Exploratory
Research
U. S. Army Electronics Command
Attn: Mr. Robert O. Parker
Executive Secretary, JSTAC
(AMSEL-XL-D)
Fort Monmouth, N. J. 07703

Director
National Security Agency
Fort George G. Meade
Maryland 20755
Attn: James T. Tippett

Director, Naval Research Laboratory
Technical Information Officer
Washington, D. C.
Attn: Code 2000 [8]

Director
Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Mass. 02139

Director
Stanford Electronics Laboratories
Stanford University
Stanford, California 94305

Commanding Officer
Naval Ordnance Laboratory
Corona, California 91720

Commanding Officer
Naval Ordnance Laboratory
White Oak, Maryland 21502 [2]

Commanding Officer
Naval Ordnance Test Station
China Lake, Calif. 93555

Commanding Officer
Naval Training Device Center
Orlando, Florida 32811

Commanding Officer
Office of Naval Research Branch Office
1030 East Green Street
Pasadena, California

Commanding Officer
Office of Naval Research Branch Office
219 South Dearborn Street
Chicago, Illinois 60604

Commanding Officer
Office of Naval Research Branch Office
495 Summer Street
Boston, Massachusetts 02210

Commanding Officer
Office of Naval Research Branch Office
207 West 24th Street
New York, New York 10011

Commanding Officer
Office of Naval Research Branch Office
Box 39, Fleet Post Office
New York 09510 [2]

Commanding Officer
U. S. Army Electronics R & D Activity
White Sands Missile Range
New Mexico 88002

Commanding Officer
U. S. Army Engineer R & D Laboratory
Attn: STINFO Branch
Fort Belvoir, Virginia 22060

Commanding Officer
U. S. Army Research Office (Durham)
Attn: CRD-AA-IP (Richard O. Ulsh)
Box CM, Duke Station
Durham, North Carolina 27706

Commanding General
USASTRATCOM
Technical Information Center
Fort Huachuca, Arizona 85613

Commanding Officer
Harry Diamond
Attn: Dr. Berthold Altman (AMXDO-TI)
Connecticut Ave. & Van Ness St. NW
Washington, D. C. 20438

Commanding Officer
Human Engineering Laboratories
Aberdeen Proving Ground
Maryland 21005

Commanding Officer
U. S. Army Ballistics Research Lab.
Attn: V. W. Richards
Aberdeen Proving Ground
Maryland 21005

Director, USAF Project RAND
Via: Air Force Liaison Office
The RAND Corporation
1700 Main Street
Santa Monica, Calif. 90406
Attn: Library

Director
U. S. Army Engineer Geodesy,
Intelligence and Mapping
Research and Development Agency
Fort Belvoir, Virginia 22060

Director
U. S. Naval Observatory
Washington, D. C. 20390

Director, U. S. Naval Security Group
Attn: G43
3801 Nebraska Avenue
Washington, D. C. 20390

Division of Engineering and Applied
Physics
130 Pierce Hall
Harvard University
Cambridge, Massachusetts 02138

Professor A. A. Dougal, Director
Laboratories for Electronics and
Related Sciences Research
University of Texas
Austin, Texas 78712

ESD (ESTI)
L. G. Hanscom Field
Bedford, Mass. 01731 [2]

European Office of Aerospace Research
Shell Building
47 Rue Cantersteen
Brussels, Belgium [2]

Colonel Robert E. Fontana
Dept. of Electrical Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

General Electric Company
Research Laboratories
Schenectady, New York 12301

Professor Nicholas George
California Institute of Technology
Pasadena, California 91109

Goddard Space Flight Center
National Aeronautics and Space Admin.
Attn: Library, Documents Section
Code 252
Green Belt, Maryland 20771

Dr. John C. Hancock, Director
Electronic Systems Research Laboratory
Purdue University
Lafayette, Indiana 47907

Dr. H. Harrison, Code RRE
Chief, Electrophysics Branch
National Aeronautics and Space Admin.
Washington, D. C. 20546

Head, Technical Division
U. S. Naval Counter Intelligence
Support Center
Fairmont Building
4420 North Fairfax Drive
Arlington, Virginia 22203

Headquarters
Defense Communications Agency
The Pentagon
Washington, D. C. 20305

Dr. L. M. Hollinsworth
ARCRL (CRN)
L. G. Hanscom Field
Bedford, Massachusetts 01731

Hunt Library
Carnegie Institute of Technology
Schenely Park
Pittsburgh, Pa. 15213

The Johns Hopkins University
Applied Physics Laboratory
8621 Georgia Avenue
Silver Spring, Maryland 20910
Attn: Boris W. Kuvshinoof
Document Librarian

Lt. Col. Robert B. Kalisch
Chief, Electronics Division
Directorate of Engineering Sciences
Air Force Office of Scientific Research
Arlington, Virginia 22209 [5]

Colonel Kee
ARFSTE
Hqs. USAF
Room ID-429, The Pentagon
Washington, D. C. 20330

Dr. S. Benedict Levin, Director
Institute for Exploratory Research
U. S. Army Electronics Command
Fort Monmouth, New Jersey 07703

Los Alamos Scientific Laboratory
Attn: Reports Library
P. O. Box 1663
Los Alamos, New Mexico 87544

Librarian
U. S. Naval Electronics Laboratory
San Diego, California 95152 [2]

Lockheed Aircraft Corp.
P. O. Box 504
Sunnyvale, California 94088

Dr. I. R. Mirman
AFSC (SCT)
Andrews Air Force Base, Maryland

Lt. Col. Bernard S. Morgan
Frank J. Seiler Research Laboratory
U. S. Air Force Academy
Colorado Springs, Colorado 80912

Dr. G. J. Murphy
The Technological Institute
Northwestern University
Evanston, Illinois 60201

Mr. Peter Murray
Air Force Avionics Laboratory
Wright-Patterson AFB, Ohio 45433

NASA Lewis Research Center
Attn: Library
21000 Brookpark Road
Cleveland, Ohio 44135

NASA Scientific & Technical
Information Facility
Attn: Acquisitions Branch (S/AK/DL)
P. O. Box 33
College Park, Maryland 20740 [2]

National Science Foundation
Attn: Dr. John R. Lehmann
Division of Engineering
1800 G Street, NW
Washington, D. C. 20550

National Security Agency
Attn: R4 - James Tippet
Office of Research
Fort George C. Meade, Maryland 20755

Naval Air Systems Command
AIR 03
Washington, D. C. 20360 [2]

Naval Electronics Systems Command
ELEX 03
Falls Church, Virginia 22046 [2]

Naval Ordnance Systems Command
ORD 32
Washington, D. C. 20360 [2]

Naval Ordnance Systems Command
SHIP 035
Washington, D. C. 20360

Naval Ship Systems Command
SHIP 031
Washington, D. C. 20360

New York University
College of Engineering
New York, New York 10019

Dr. H. V. Noble
Air Force Avionics Laboratory
Wright-Patterson AFB, Ohio 45433

Office of Deputy Director
(Research and Information Rm. 3D1037)
Department of Defense
The Pentagon
Washington, D. C. 20301

Polytechnic Institute of Brooklyn
55 Johnson Street
Brooklyn, New York 11201
Attn: Mr. Jerome Fox
Research Coordination

RAD (EMLAL-1)
Griffiss AFB, New York 13442
Attn: Documents Library

Raytheon Company
Bedford, Mass. 01730
Attn: Librarian

Lt. Col. J. L. Reeves
AFSC (SCBB)
Andrews Air Force Base, Md. 20331

Dr. A. A. Dougal
Asst. Director of Research
Office of Defense Res. and Eng.
Department of Defense
Washington, D. C. 20301

Research Plans Office
U. S. Army Research Office
3045 Columbia Pike
Arlington, Virginia 22204

Dr. H. Robl, Deputy Chief Scientist
U. S. Army Research Office (Durham)
Durham, North Carolina 27706

Emil Schafer, Head
Electronics Properties Info. Center
Hughes Aircraft Company
Culver City, California 90230

School of Engineering Sciences
Arizona State University
Tempe, Arizona 85281

SAMSO (SMSDI-STINFO)
AF Unit Post Office
Los Angeles, California 90045

SSD (SSTRT/Lt. Starbuck)
AFUPO
Los Angeles, California 90045

Superintendent
U. S. Army Military Academy
West Point, New York 10996

Colonel A. Swan
Aerospace Medical Division
AMD (AMRXI)
Brooks AFB, Texas 78235

Syracuse University
Dept. of Electrical Engineering
Syracuse, New York 13210

University of California
Santa Barbara, California 93106
Attn: Library

University of Calif. at Los Angeles
Dept. of Engineering
Los Angeles, California 90024

University of Michigan
Electrical Engineering Dept.
Ann Arbor, Michigan 48104

U. S. Army Munitions Command
Attn: Technical Information Branch
Picatinney Arsenal
Dover, New Jersey 07801

U. S. Army Research Office
Attn: Physical Sciences Division
3045 Columbia Pike
Arlington, Virginia 22204

U. S. Atomic Energy Commission
Division of Technical Information Ext.
P. O. Box 62
Oak Ridge, Tenn. 37831

Dept. of Electrical Engineering
Texas Technological College
Lubbock, Texas 79409

U. S. Naval Weapons Laboratory
Dahlgren, Virginia 22448

Major Charles Waespy
Technical Division
Deputy for Technology
Space Systems Division, AFSC
Los Angeles, California 90045

The Walter Reed Institute of Research
Walter Reed Medical Center
Washington, D. C. 20012

AFSC (SCTR)
Andrews Air Force Base
Maryland 20331

Weapons Systems Test Division
Naval Air Test Center
Patuxtent River, Maryland 20670
Attn: Library

Weapons Systems Evaluation Group
Attn: Col. Daniel W. McElwee
Department of Defense
Washington, D. C. 20305

Yale University
Engineering Department
New Haven, Connecticut 06720

Mr. Charles F. Yost
Special Asst. to the Director of Research
NASA
Washington, D. C. 20546

Dr. Leo Young
Stanford Research Institute
Menlo Park, California 94025

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) Division of Engineering and Applied Physics Harvard University, Cambridge, Massachusetts	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE
THE MATRIX ALGEBRA PROGRAM A CONVERSATIONAL LANGUAGE FOR NUMERICAL MATRIX OPERATIONS - PART I: USER S MANUAL

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Interim Technical Report

5. AUTHOR(S) (First name, middle initial, last name)
P. M. Newbold

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
May 1968	49	4

8a. CONTRACT OR GRANT NO. N00014-67-A-0298-0006 and b. PROJECT NO. NASA NGR-22-007-068 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Technical Report No. 561 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT
Reproduction in whole or in part is permitted for any purpose of the United States Government.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research

13. ABSTRACT

This report is Part I of a two-part description of a new programming language MAP. The language is in a conversational mode, created expressly for direct-access time-sharing computer systems. It is designed to execute numerical matrix operations with the same ease and flexibility as scalar operations. No knowledge of any other language is required.
Part I is the User's Manual for the language.

Unclassified
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Direct-access computer language						
Conversational programming language						
Numerical matrix operations						