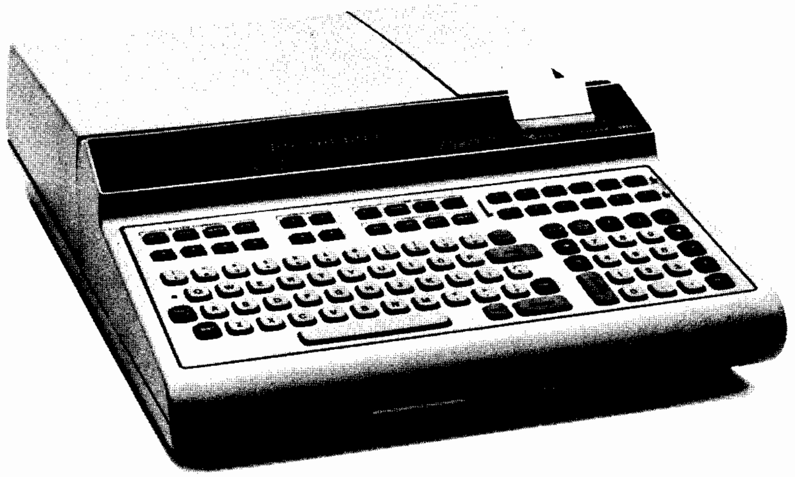
Matrix Programming

Computer Museum


HP 9825A Desktop Computer

Hewlett-Packard Computer Systems Division
Herrenberger Straße 110/130, 7030 Böblingen, West Germany

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Table of Contents

Preface

Before using this manual, you should be familiar with the 9825A Calculator and the HPL programming language described in the HP 9825A Operating and Programming Manual.

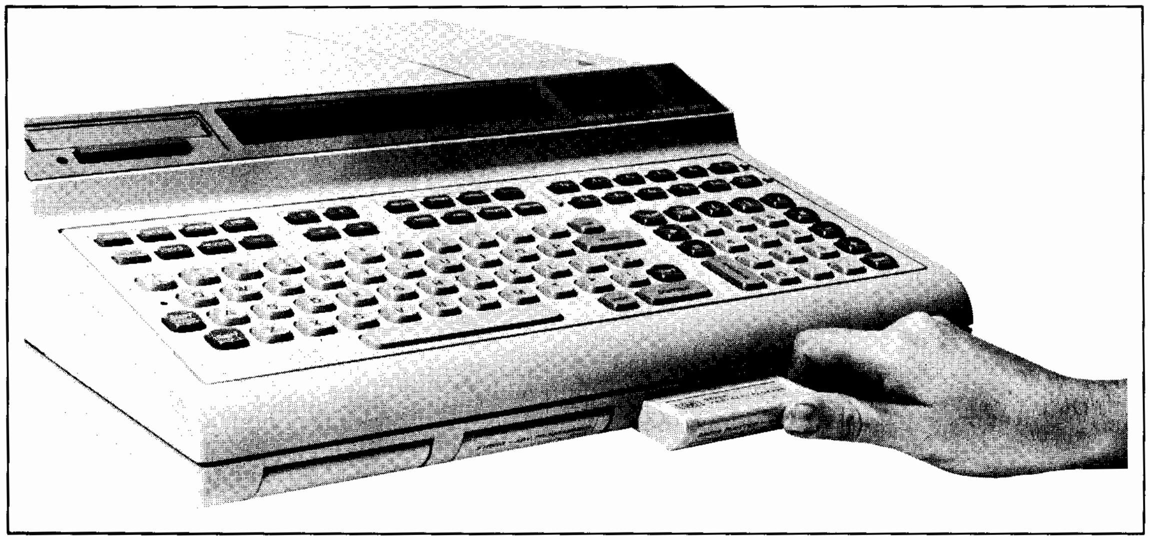
Chapter 1
Introduction

ROM Description

The Matrix ROM (Read Only Memory) provides additional statements to the HPL language for performing mathematical operations on matrices and arrays. The Matrix ROM uses no read/write memory when installed in the HP 9825A Calculator.

Inspection and Installation

The Matrix ROM can be plugged into any one of the four ROM slots located on the bottom front of the calculator, as shown below.



ROM Installation

To install your ROM card, first turn the calculator off. With the label right side up, slide the ROM through the ROM slot door. Press it in until the front of the ROM card is even with the front of the calculator. Then turn your calculator on.

Refer to the HP 9825A System Test Booklet to check the operation of the Matrix ROM.

Syntax

The following conventions apply to the syntax for the Matrix ROM statements found in this manual.

`dot matrix`	- All items in dot matrix are required, exactly as shown.
[ ]	- All items in square brackets are optional, unless the brackets are in dot matrix.
...	- Three dots indicate that the previous item may be duplicated.

All Matrix ROM statements can be executed from the keyboard, in live keyboard mode, and within a program.

Error Messages

The Matrix ROM adds error messages M1 through M5. Explanations of these errors can be found inside the back cover of this manual.

Chapter 2
General Information

Description

An array is a collection of data elements having any number of dimensions. Two special kinds of arrays are called matrices and vectors. A matrix is a two-dimensional table of data or any collection of data elements arranged in rows and columns. A vector is a collection of data elements arranged in a single column or row. The terms array and array variable are used interchangeably to specify an entire array; array element or element is used to specify a single item in an array.

Here is an example of a matrix:

Grade	**Boys**	**Girls**
1	10	7
2	9	8
3	9	10
4	7	9
5	7	10
6	9	11

Here is an example of a vector:

Test Scores
93
85
79
89
68
95
100

Dimensioning Arrays

An array must initially be defined in a dimension (**dim**) statement. A dim statement reserves storage space for arrays. Array names can be any single upper case letter from A through Z.

The following statement dimensions three arrays:

```
dim A[100],B[30,5], C[X,Y,Z]
```

Array A is a vector with 100 rows containing 1 element each, for a total of 100 elements.

Array B is a matrix with 30 rows of 5 elements each, for a total of 150 elements.

Array C is three dimensional array whose dimensions are given by the values of the simple variables X, Y, and Z.

When a dimension statement that defines an array is executed, all elements of that array are initialized to zero.

The working size of an array can be smaller than its defined size (the memory space reserved for it by the dim statement). For example, A[20,20] can store fewer than 400 data elements; the dim statement specifies only the maximum number of elements. The dimensions of an array can be redefined; see Redimensioning Arrays on page 16.

Specifying Bounds for Dimensions

A dimension may be specified by giving lower and upper bounds. The lower bound must be specified before the upper bound. The two are separated by a colon. The bounds must be in the range from −32767 through 32767. For example:

This statement reserves 12 matrix elements ▶ `dim A[1976:1978,4]`

The same amount of memory is reserved by ▶ `dim B[3,4]`

The elements in array A can be referenced as follows:

A[1976,1]	A[1976,2]	A[1976,3]	A[1976,4]
A[1977,1]	A[1977,2]	A[1977,3]	A[1977,4]
A[1978,1]	A[1978,2]	A[1978,3]	A[1978,4]

If a lower bound is not specified, as in `B[3,4]`, it is assumed to be 1, as in `B[1:3,1:4]`.

Array Elements

Array elements are specified by the array name followed by brackets which enclose the numbers specifying the array element. For example, X[1] specifies element 1 in vector X; X[23] specifies element 23 in vector X.

Range of Values of Array Elements

The internal representation of numbers used in the 9825A is floating-point format, with one digit to the left of the decimal point (scientific notation).

The Matrix ROM does not require that its final results be within the storage range as in the basic calculator. Results can have any values within the calculation range (exponents of 0 ± 511).

Final results, therefore, are treated the same as results in the calculation range with one exception: whenever a matrix or array operation generates a final result outside the normal storage range, the ROM sets flag 15. (Note that the operations of copying an array or transposing a matrix do not generate any new results, but only relocate existing values; these operations do not affect flag 15.)

Whenever attempts are made to use these out-of-range results in non-matrix operations, they are treated just like any other out-of-range results. For example, an attempt to print or display such a result without setting flag 14 will result in error 74 or 75. If flag 14 is set, the storage range default value is printed (the actual stored value is not changed to the default value, however).One way to view the digits contained in such out-of-range quantities is to multiply or divide the values by some constant to bring the result into the storage range.

Types of Operations

The operations performed by the Matrix ROM can be divided into two classes: matrix operations and array operations. The matrix operations (inv, mat, trn, & idn) can be performed only using arrays having 1 or 2 dimensions. All others are array operations, and can be used with arrays having any number of dimensions.

Each statement of the Matrix ROM can perform only one operation, except in those cases such as aprt (array print), where a single statement can be used to cause the indicated operation to be performed on more than one array. Thus, an operation such as: `ara A+B+C→D` is not allowed. Such an operation requires two steps: `ara A+B→D; ara C+D→D`.

Chapter 3
Input and Output of Array Data

Methods for storing data in an array and for printing data from an array are described next.

Data Input

Values can be assigned to array elements by individual assignment statements or by a single assignment statement within a loop.

Here's an example that uses individual assignment statements.

```
0: dim A[3,2]
1: 1→A[1,1];2→A[
 1,2]
2: 3→A[2,1];4→A[
 2,2]
3: 5→A[3,1];6→A[
 3,2]
```

Values can also be assigned to array elements using a loop.

```
4: dim B[3,2];
 1→I
5: I→B[I,1]→B[I,
 2]
6: I+1→I;if I<4;
 jmp -1
```

With the for/next loop capabilities of the Advanced Programming ROM, values can be assigned as shown:

```
0: dim C[3,2]
1: for I=1 to 3
2: for J=1 to 2
3: I+J→C[I,J]
4: next J
5: next I
```

The enter (**ent**) or enter print (**enp**) statements can also be used to assign values to array elements as in this example:

```
0: dim L[5]
1: for X=1 to 5
2: ent L[X]
3: next X
4: end
```

Array Initialization

An array can have all of its elements set to a single value specified by a number or simple variable A through Z using the initialize array (**ina**) statement.

Syntax: `ina` array variable [`:` number or simple variable]
[`,` array variable [`:` number or simple variable]...]

If a number or simple variable is not specified, zero is assumed. Multiple arrays can be initialized in a single statement:

```
ina A, B:4, C:X
```

This statement initializes each element of array A to zero; each element of array B to 4; and each element of array C to the value of the simple variable X.

Array Output

The array print (**aprt**) statement is used to print each element of an array on the 9825A's printer.

Syntax: `aprt` array variable [`,` array variable...]

Any number of array variables can be printed using a single aprt statement, as in `aprt A, B, C`.

In the following example, the array A is printed.

Array A

1	4
2	5
3	6

```
0: dim A[3,2]
1: 1→A[1,1];2→A[
 2,1];3→A[3,1]
2: 4→A[1,2];5→A[
 2,2];6→A[3,2]
3: aprt A
```

```
A[1,1]      1.00
A[2,1]      2.00
A[3,1]      3.00

A[1,2]      4.00
A[2,2]      5.00
A[3,2]      6.00
```

Notice that the leftmost subscript increments most rapidly.

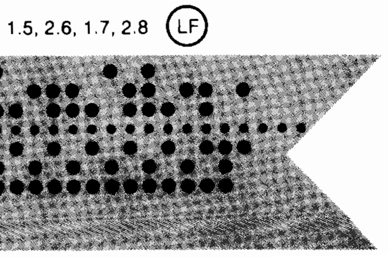
All array elements are printed using the current fixed or float setting. The elements are printed column by column with a space after each column. For arrays with more than two dimensions, an additional space is inserted each time the third subscript is incremented. The aprt statement also inserts a space before an array is printed and three spaces following the end of the array.

Other Methods of Input and Output

With the General I/O ROM, data for arrays can be entered from peripherals such as tape readers, card readers, and digital voltmeters using the read (**red**) statement. This is often the quickest method for entering data into large arrays.

In the following example, data is entered into an array X from a paper tape reader set at select code 3.

The tape has this format ➧



The following program segment is used to enter and print data from the tape ➧

```
0: dim X[4]
1: 1→I
2: red 3,X[I];
 jmp (I+1→I)>4
3: aprt X
```

The printout is ➧

```
1.50
2.60
1.70
2.80
```

Data in many different formats can be entered using the read statement with the format **(fmt)** statement. For more details, see the General I/O Programming Manual.

To output an array to a peripheral device, the aprt statement cannot be used. The write **(wrt)** statement from the General I/O ROM must be used. In the following example, all of the data from array X (above) is printed on the HP 9871A Printer at select code 6.

Execute this program segment ➧

```
4: 1→I
5: wrt 6,X[I];
 jmp (I+1→I)>4
6: stp
```

The following printout is obtained:

```
1.50
2.60
1.70
2.80
```

The element from an array can be output in many different forms using the write statement in conjunction with the format statement. For details, see the General I/O Programming Manual.

Chapter 4
Array Operations

The operations described in this chapter can be performed on arrays having any number of dimensions. Those operations that can be performed only on matrices are described in Chapter 5.

Array Addition

The corresponding elements of two arrays can be added together using the array arithmetic (**ara**) statement.

Syntax: `ara` array variable + array variable → array variable

Each element in the first array is added to the corresponding element in the second array and the result is stored in the corresponding position in the third array. The dimensions of the three arrays must be the same: the subscripts need not be the same, but the number of elements in each dimension must be the same. The number of dimensions must also be the same, even if some of the dimensions are 1. Thus A cannot be added to B if A is dimensioned [3] and B is dimensioned [3,1].

In the following example, matrices A and B are added together and stored in C. All three matrices are dimensioned as:

```
dim A[2,3],B[2,3],C[2,3]
```

Matrices A and B have the following elements:

Matrix A

1	4	3
2	6	5

Matrix B

4	2	1
3	8	7

After executing `ara A+B→C`, matrix C contains:

Matrix C

5	6	4
5	14	12

Calculator memory space can be saved if the result is accumulated in one of the arrays already containing data, as with X in the line:

```
ara A+X → X
```

Array Subtraction, Multiplication, and Division

The corresponding elements of two arrays can be subtracted, multiplied, or divided using the array arithmetic (**ara**) statement. The rules which apply to addition also apply to these three operations.

Syntax:	`ara` array variable `-` array variable `→` array variable
	`ara` array variable `*` array variable `→` array variable
	`ara` array variable array variable `→` array variable
	`ara` array variable `/` array variable `→` array variable

Note that two forms are allowed for multiplication. The `*` sign can be omitted for implied multiplication.

As an example, these three operations are performed on matrices A and B and the result is stored in matrix C:

Matrix A

4	2	1
7	5	2

Matrix B

5	1	4
4	3	1

	Matrix C		
`ara A - B → C`	−1	1	−3
	3	2	1
`ara A * B → C`	20	2	4
`ara AB → C`	28	15	2
`ara A/B → C`	0.80	2.00	0.25
	1.75	1.67	2.00

Two other statements involve multiplication: scalar multiply (**smpy**, below) and matrix multiply (**mat**, page 19).

Copying Arrays

The data elements in one array can be copied into the corresponding positions in another array .

Syntax: `ara` variable name `→` variable name

To copy array C into array A, execute:

```
ara C→ A
```

The two variables must have the same dimensions, as described under Array Addition on page 11.

Scalar Multiplication of Arrays

Each element of an array can be multiplied by a number or by the value of any simple variable A through Z by using the scalar multiply (**smpy**) statement. The number or simple variable must precede the array variable.

Syntax: `smpy` number or simple variable `*` array variable `→` array variable

The `*` sign may be omitted for implied multiplication.

The array variable being multiplied and the array variable where the result is stored must both have the same dimensions, as in array addition.

In the following example, matrix X is multiplied by the scalar 4 and the result is stored in matrix Y.

Matrix X

4	2	1
3	5	3

After executing the statement `smpy 4 # X + Y`, the matrix Y contains:

Matrix Y

16	8	4
12	20	12

Example

Below are two tables containing the Math, Science, and Reading grades achieved by five students during two quarters of one school year.

First Quarter

Student No.	Math	Science	Reading
1	80	85	78
2	71	80	72
3	97	92	83
4	77	82	98
5	93	94	98

Second Quarter

Student No.	Math	Science	Reading
1	78	81	80
2	73	82	88
3	93	90	85
4	81	88	94
5	91	90	84

We will use the grades in the first quarter for array A, and in the second quarter for array B:

Array A			Array B		
80	85	78	78	81	80
71	80	72	73	82	88
97	92	83	93	90	85
77	82	98	81	88	94
93	94	98	91	90	84

If the statement `ara A+B→C` is executed, array C contains:

158	166	158
144	162	160
190	182	168
158	170	192
184	184	182

The grades for each student in each class have been added. We can now find the average of the two term grades for each student in each class by executing:

```
smpy .5C→C
```

The result in C is now:

79	83	79
72	81	80
95	91	84
79	85	96
92	92	91

The array now represents the average grade for each student in each of the three subjects:

Average Mid-Year Grades

Student No.	**Math**	**Science**	**Reading**
1	79	83	79
2	72	81	80
3	95	91	84
4	79	85	96
5	92	92	91

Redimensioning Arrays

Arrays previously dimensioned using a dim statement can be redimensioned using the redimension (**rdm**) statement, with these limitations: In each array, the number of elements cannot exceed the original number, and the number of dimensions must be the same as the original number.

Syntax: rdm item [, item...]
where item is:
array variable [[number or simple variable ,]number or simple variable]

If an array is redimensioned more than once, the number of elements is limited only by the number of elements defined in the original dim statement; subsequent redimensioning to a smaller size does not further limit the size available.

Redimensioning an array does not change the amount of memory space reserved for the array; it merely permits the user to use the allocated space (or part of it, if the new size is smaller) in a different way. If the element values are to be recoded in a data file on the tape cartridge, the tape file size must be large enough to record the number of elements specified in the original dim statement.

In the following example, the 2 × 3 matrix A is redimensioned several times. The value associated with each element in the matrix represents its location in the original matrix (e.g., 21 represents A [2,1]).

```
0: dim A[2,3]
1: 11→A[1,1]
2: 12→A[1,2]
3: 13→A[1,3]
4: 21→A[2,1]
5: 22→A[2,2]
6: 23→A[2,3]
```

♦ Dimensions matrix and stores element values.

```
7: prt "A[2,3]:"
 ;aprt A
```

```
A[2,3]:

              11.00
              21.00

              12.00
              22.00

              13.00
              23.00
```

```
8: rdm A[1971:19
 76,1];prt "A[19
 71:1976,1]:";
 aprt A
```

```
A[1971:1976,1]:

                 11.00
                 21.00
                 12.00
                 22.00
                 13.00
                 23.00
```

```
9: rdm A[3,1];
 prt "A[3,1]:";
 aprt A
```

```
A[3,1]:

                 11.00
                 21.00
                 12.00
```

Notice in all cases that by incrementing the leftmost subscript most rapidly the values are revealed in the same order as in the original matrix.

Chapter 5
Matrix Operations

The operations described in this chapter can be performed only with matrices and vectors (arrays having either one or two dimensions). Other restrictions on the dimensions apply to each of the operations, as stated below.

Matrix Multiplication

Matrices can be multiplied only when the number of columns in the first matrix equals the number of rows in the second matrix; the resulting matrix must have the same number of rows as the first matrix and the same number of columns as the second matrix.

Syntax: `mat` array variable `*` array variable `→` array variable

The `*` sign can be omitted for implied multiplication.

The matrix to the left of the assignment arrow must not appear to the right of the assignment arrow, (i.e., you cannot say `mat X*Y→X`).

For a matrix, the left subscript represents the row and the right subscript represents the column. Thus, in a matrix A, A[4,2] represents the element in the 4th row, 2nd column.

In matrix multiplication, for `mat A * B → C`, the elements in each column of Matrix B are multiplied by the corresponding elements in each row of Matrix A. The row products are then added together and stored in the appropriate row and column of Matrix C.

Mathematically, the product of two matrices, mat AB → C, is represented as follows:

$$C(I,K) = \sum_{J=1}^{N} A(I,J)*B(J,K)$$

Where N is the number of columns in Matrix A, and rows in Matrix B.

In Summary

For any matrix multiplication, mat AB→C, if the dimensions of A are [M,N] and the dimensions of B are [N,P], the result is a matrix of dimensions [M,P]. For example, a 5 by 4 matrix multiplied by a 4 by 1 matrix results in a 5 by 1 matrix. The value of N, above, must be the same in the two matrices. Also, the result of mat BA→C is not necessarily the same as mat AB→C.

Following are examples of matrix multiplication:

Below is a table of ticket sales for four bus routes, and a table of ticket prices for the three kinds of tickets. Matrix multiplication can give you the total sales for each route.

Table A. Ticket Sales By Route

Route	Single Trip	Round Trip	Commuter
1	143	200	19
2	49	97	24
3	314	77	22
4	82	65	16

Table B. Ticket Prices

	Price
Single Trip	.25
Round Trip	.45
Commuter	18.00

Computer Museum

Here are the instructions to enter the values into Arrays A and B, respectively, and perform the matrix multiplication:

Without Advanced Programming ROM

```
0: dim A[4,3],
 B[3],C[4]
1: 1→I→J
2: ent A[I,J];
 jmp (I+1→I)>4
3: 1→I;if (J+
 1→J)<4;jmp -1
4: ent B[1],B[2]
 ,B[3]
5: mat AB→C;fxd
 2;aprt C
```

With Advanced Programming ROM

```
0: dim A[4,3],
 B[3],C[4]
1: for I=1 to 4;
 for J=1 to 3
2: ent A[I,J];
 next J;next I
3: ent B[1],B[2]
 ,B[3]
4: mat AB→C;fxd
 2;aprt C
```

After A and B have been multiplied, Matrix C contains the total sales, in dollars, for each route: C[1] contains sales for route 1, C[2] contains sales for route 2, and so forth.

Matrix C

467.75	(.25*143+.45*200+18.00*19)
487.90	(.25*49+.45*97+18.00*24)
509.15	(.25*314+.45*77+18.00*22)
337.75	(.25*82+.45*65+18.00*16)

In the preceding example, A, a 4 by 3 matrix, is multiplied by B, a 3 by 1 matrix, giving C, a 4 by 1 matrix.

Suppose a price change is being considered and matrix B contains two columns of ticket prices:

	Old Price	New Price
Single Trip	.25	.30
Round Trip	.45	.50
Commuter	18.00	17.00

Then A, a 4 by 3 matrix, multiplied by B, a 3 by 2 matrix, results in C, a 4 by 2 matrix:

```
467.75    465.90
487.90    471.20
509.15    506.70
337.75    329.10
```

Here are the instructions to perform this:

Without Advanced Programming ROM

```
0: dim A[4,3],
 B[3,2],C[4,2]
1: 1→I→J
2: ent A[I,J];
 jmp (I+1→I)>4
3: 1→I;if (J+
 1→J)<4;jmp -1
4: 1→I→J
5: ent B[I,J];
 jmp (I+1→I)>3
6: 1→I;if (J+
 1→J)<3;jmp -1
7: mat AB→C;fxd
 2;aprt C
```

With Advanced Programming ROM

```
0: dim A[4,3],
 B[3,2],C[4,2]
1: for I=1 to 4;
 for J=1 to 3
2: ent A[I,J];
 next J;next I
3: for I=1 to 3;
 for J=1 to 2
4: ent B[I,J];
 next J;next I
5: mat AB→C;fxd
 2;aprt C
```

The Identity Matrix

The identity matrix is a square matrix containing zeros with the principal diagonal containing all ones. Use the identity (**idn**) statement to define a square matrix as an identity matrix.

Syntax: `idn` array variable [`,` array variable ...]

If matrix B has been dimensioned [3,3], the statement `idn B` results in:

```
1 0 0
0 1 0
0 0 1
```

In order to create an identity matrix using the idn statement the matrix must have been previously dimensioned as a square matrix.

The identity matrix is defined as the matrix which, when multiplied* by any matrix A, results in matrix A.

Transposition of Matrices

The transposition (**trn**) of a matrix causes the rows in the matrix to become columns, and the columns to become rows:

Syntax: trn array variable → array variable

The same matrix cannot appear on both sides of the assignment arrow. The dimensions of the transposed matrix must be equal to the reverse of the dimensions of the original matrix; that is, if the dimensions of B are [P,Q], transposition of Matrix B results in a matrix of dimensions [Q,P].

The transposition of:

$$\begin{matrix}2 & 3\\4 & 5\\6 & 7\end{matrix}$$

results in:

$$\begin{matrix}2 & 4 & 6\\3 & 5 & 7\end{matrix}$$

Row 1 in the first matrix becomes column 1 in the second. Then row 2 becomes column 2 and so on.

*That is, matrix multiplication, not array multiplication.

Matrix Inversion and Determinant

The inverse (if it exists) of a matrix can be computed and stored in another matrix and the determinant of the matrix can be assigned to a simple variable using the matrix inversion (**inv**) statement.

Syntax: `inv` array variable `→` array variable [`;` simple variable]

The determinant is stored in the simple variable if it is specified. If no simple variable is specified for the determinant value, the determinant is computed but the value is not stored. Both matrices must have been previously dimensioned as square matrices of the same size. If desired, the result can be stored in the original matrix variable. (e.g., `inv C→C`).

If an inversion is attempted and the 9825A computes a determinant value of exactly zero, indicating that an inverse cannot be computed, the calculator fills the result matrix with all zeros, sets flag 15, and gives error M5. If flag 14 is 1 (set) when this occurs, the error is not given, and operation continues.

In performing the inversions and calculating the determinant, the 9825A uses an internal work area of 12 N+4 bytes (for an N by N matrix). Thus, the maximum size of matrix which can be inverted is:

Memory Size	**Dimensions**
Standard 6844 bytes	28 × 28
Opt. 1 15036 bytes	42 × 42
Opt. 2 23228 bytes	53 × 53
Opt. 3 31420 bytes	61 × 61

The inverse of a matrix A is a matrix B which, when multiplied by matrix A, produces an identity matrix. Only square matrices can be inverted. The determinant of an N by N matrix A is defined mathematically as:

$$\sum \pm A[1,I]\,A[2,J]\,A[3,K] \ldots\ A[N,R]$$

where the second subscripts I, J, K, ...,R form a permutation of the integers 1 to N, and the sign of each product depends on the order of the permutation. If the determinant of a square matrix of simultaneous linear equations is not zero, the system of equations has a solution; that is, the inverse of the matrix can be obtained. However, if the determinant is zero, the system has no solution, the inverse does not exist; and the matrix is termed "singular".

The Matrix ROM employs a modified Gauss-Jordan reduction technique using the maximum pivot strategy. This method is superior to the standard Gauss-Jordan elimination or the diagonal pivot strategy since it will successfully invert all but singular or very near-singular matrices. Also, by using maximum elements as the pivots, the accuracy of the results is maximized.

Matrix inversion can be used in solving sets of simultaneous linear equations. For example:

$$3X + 4Y = 47$$
$$2X + 2Y = 28$$

In the notation of matrix algebra:

$$\begin{bmatrix} 3 & 4 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 47 \\ 28 \end{bmatrix} \text{ or } [A]\,[B] = [C]$$

Multiplying both sides of the equation by the inverse of A (denoted by $[A^{-1}]$) gives:

$$[B] = [A^{-1}]\;[C]$$

This can be accomplished by creating matrices A, B, and C, followed by:

```
inv A→A; mat AC→B; fxd 2; aprt B
```

The printer output is: ⧫

```
                                  9.00
                                  5.00
```

Thus the solution to the problem is:

$$X = 9$$
$$Y = 5$$

Notes

Notes

Array Operations

Dimension

```
dim A[5,10,3],B[X,Y]
```

Dimensions arrays as specified, reserving memory as required. Arrays may have any number of dimensions.

Redimension

```
rdm A[6,3:4,12],B[4,1]
```

Redimensions previously dimensioned arrays. The number of dimensions must be the same as the original number. The new size cannot exceed the original size.

Array print

```
aprt A,B,C
```

Prints entire arrays on the 9825A Printer. Arrays are printed column by column.

Array arithmetic and assignment

```
ara A+B→C
ara A-B→C
ara A*B→C
ara AB→C
ara A/B→C
ara A→C
```

Performs the indicated operation using corresponding elements from each of the arrays specified. Arrays must have identical dimensions. The same array may appear on both sides of →.

Scalar multiplication

```
smpy S*A→B
smpy SA→B
```

Multiplies each element of an array by a scalar. The scalar may be a number or a simple variable. The same array may appear on both sides of →.

Initialization of Arrays

```
ina A, B, C
ina A:S,B:T
```

Initializes arrays to the values specified. Values may be numbers or simple variables. If a value is not specified, zero is used.

Matrix Operations

Matrix inversion and determinant

```
inv A→B, D
```

Inverts matrix A and puts the inverse into matrix B. The determinant of matrix A is put into the simple variable D, if specified. Matrices must be the same size and square. The same matrix may appear on both sides of →.

Matrix multiplication

```
mat A*B→C
mat AB→C
```

If the dimensions of A are [M,N], then the dimensions of B and C must be [N,P] and [M,P], respectively. The same matrix cannot appear on both sides of →.

Matrix transposition

```
trn A→B
```

The dimensions of B must be the reverse of the dimensions of A. The same matrix cannot appear on both sides of →.

Identity matrices

```
idn A, B, C
```

Matrices listed become identity matrices. Matrices must be square.

Subject Index

Matrix ROM Error Messages

error M1*	Syntax error.
error M2	Improper dimensions. Array dimensions are incompatible with each other or are incompatible with the stated operation.
error M3	Improper redimension specification: New number of dimensions must equal original number. New size cannot exceed original size.
error M4*	Operation not allowed. An array which appears to the left of → cannot also appear on the right.
error M5	Matrix cannot be inverted. Computed determinant equals 0.

*These errors give a cursor when RESET is pressed, showing the location of the error.