Matrix ROM Manual

HP-83/85

COSMETICS

EQUIPMENT

CHEMICALS

TEXTILES

$$\mathbf{Z} = \begin{bmatrix} 10 & 00 \\ 01 & 00 \\ 00 & V_{11}\ V_{12} \\ 00 & V_{21}\ V_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & V \end{bmatrix}$$
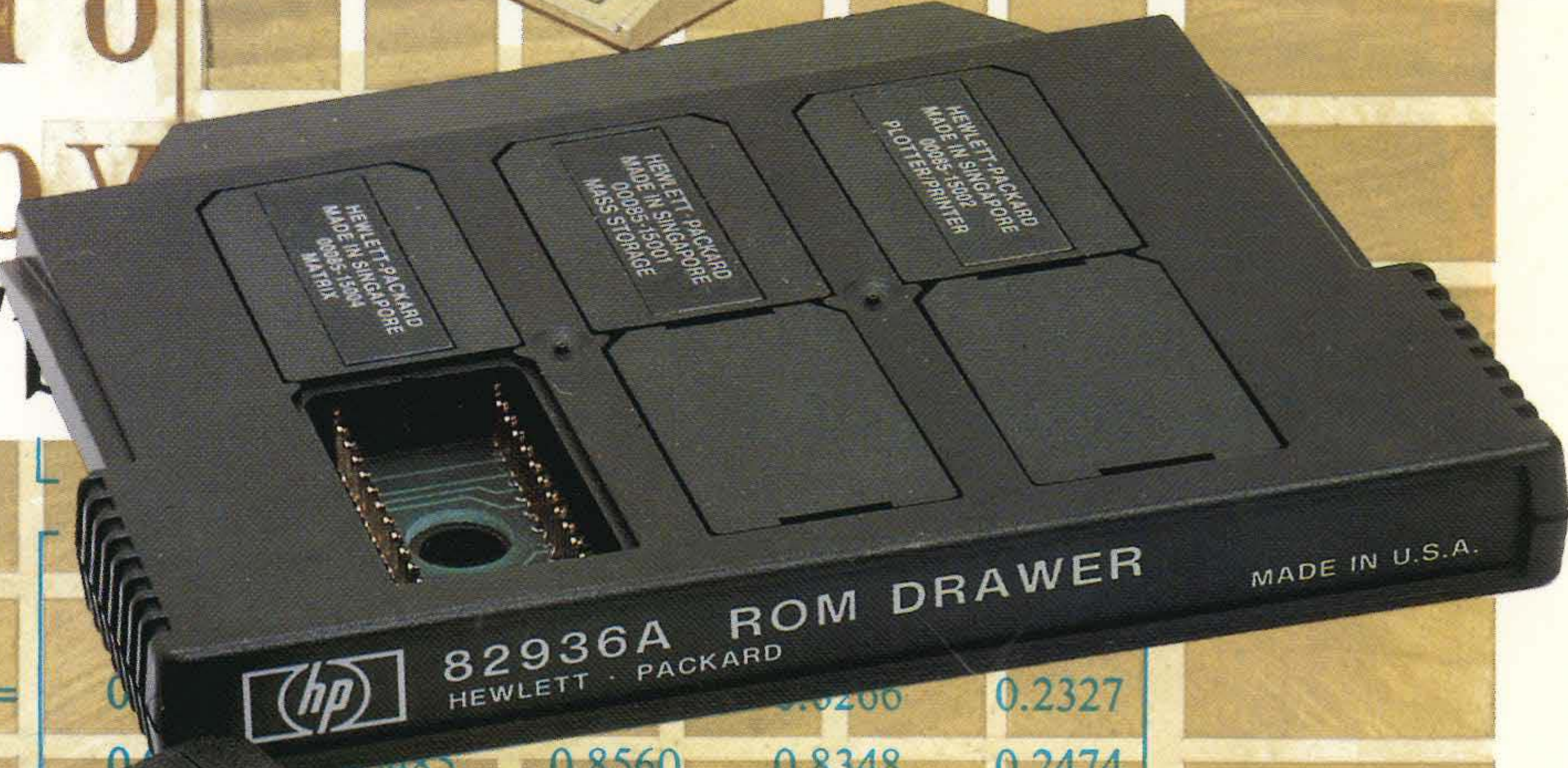
$$x = (A^T A)^{-1} A^T b$$

82936A ROM DRAWER   MADE IN U.S.A.

hp HEWLETT · PACKARD

$$A = \begin{bmatrix} \cdots & \cdots & \cdots & 0.0206 & 0.2327 \\ \cdots & 0.5485 & 0.8560 & 0.8348 & 0.2474 \\ \cdots & 0.9198 & 0.0080 & 0.7610 & 1.0506 \end{bmatrix}$$

$$K = \begin{bmatrix} \cdots & \cdots & 0.0 & 0.0569 & -0.2520 \\ 0.12\cdots & 0.0389 & 0.0 & -0.4688 & -0.5963 \\ -0.0709 & 0.1861 & -0.8383 & -0.4806 & 0.1632 \\ 0.0726 & -0.4109 & -0.5354 & 0.7253 & -0.1144 \\ -0.2002 & -0.6231 & 0.1027 & -0.1411 & 0.7357 \end{bmatrix}$$

$$R = \begin{bmatrix} 1.4446 & 1.6867 & 1.2530 \\ 0.0 & -1.3389 & -0.1486 \\ 0.0 & 0.0 & 1.1831 \end{bmatrix}$$

HEWLETT PACKARD

HP-83/85
Matrix ROM Manual

February 1984

Contents

Notes

Getting Started

Introduction

The HP-83/85 Matrix ROM provides you with a powerful set of statements and functions for working with arrays (matrices and vectors). These capabilities enable you to calculate with more convenience, speed, and accuracy than if you worked with arrays without the ROM.

This manual assumes that you have a working knowledge of linear algebra and matrix theory. In addition, it presumes that you are familiar with operating and programming an HP-83 or HP-85 Computer, as described in the *Owner's Manual and Programming Guide*. Section 8 of that manual (Using Variables: Arrays and Strings) contains information that is particularly relevant.

All Matrix ROM statements and functions—excepting only the MAT INPUT and MAT READ statements—work both in "calculator" mode (that is, from the keyboard) and in program mode.

The Matrix ROM uses 69 bytes of the computer's memory (not including any memory allocated to arrays). If you attempt to load into memory a large program written for the computer alone (including programs from application pacs) the message Error 19 : MEM OVF may result, indicating that the program does not fit into memory. Such a program can be loaded after installing a 16K Memory Module or removing the ROM drawer.

Throughout this manual, "HP-83/85" is used to refer to both the HP-83 and the HP-85.

Matrix ROM Installation

The Matrix ROM should be installed in an HP 82936A ROM Drawer. Please refer to the instructions accompanying the ROM Drawer, or to appendix B of the computer owner's manual, for complete ROM and ROM Drawer installation instructions.

Conventions

Arrays, Vectors, and Matrices

Throughout this manual, the term "array" is used in referring to a variable with either one or two dimensions (that is, a variable that has been declared with either one or two subscripts). The term "vector" is used in referring to an array with only one dimension; the term "matrix" is used in referring to a two-dimensional array. The Matrix ROM regards all vectors as column vectors, not row vectors. Except for the MAT ... CROSS operation and the DOT function, a matrix declared with only one column—like A(3,1) (with OPTION BASE 1 in effect)—can be used wherever a vector can be used. In all cases, a vector can be used wherever a one-column matrix can be used.

References to both matrices and vectors throughout this manual generally have the array name printed in **boldface** type (for example, matrix **A**, vector **B**). However, if the array name is part of a BASIC statement or function, it (as well as the rest of the statement or function) is printed in DOT MATRIX type (for example, MAT A = -A).

Statement and Function Notation

Statements and functions are described throughout this manual using the following conventions:

`DOT MATRIX`	Items shown in dot matrix type must be entered exactly as shown (in either uppercase or lowercase letters). If several items are shown stacked, one (but only one) of the items must be specified.
[ ]	Items shown between brackets are optional. If several items are stacked between brackets, you can specify any one or none of the items.
...	Three dots (ellipsis) following a set of brackets indicate that the items between the brackets may be repeated.
italics	Items shown in italic type are numeric expressions or names of arrays that should be specified in the statement or function.

The *result array,* specified on the left side of the $=$ sign in a `MAT` statement, is the array whose elements are assigned values. Many Matrix ROM statements specify one or two *operand arrays* as well as a result array. The operand array, specified on the right side of the $=$ sign in a `MAT` statement, is the array from which values are taken to be operated upon.

The array specified as the result array may be the same as the array specified as an operand array. Thus, values can be changed and stored in the original array without allocating storage to a new array variable.

Data Types

Like the HP-83/85 itself, the Matrix ROM works with *numeric* array variables (or, simply, numeric arrays); *string* array variables are not allowed.

As described in section 3 of the HP-83 and HP-85 owner's manuals, a numeric array can be any of three data types: `REAL`, `SHORT`, and `INTEGER`. Operations provided by the Matrix ROM put no restrictions on the types of array variables or simple variables named in statements. For example, a typical Matrix ROM statement specifies that values are to be taken from the elements of an operand array, manipulated in some particular way, and then assigned to the elements of a result array. In all cases, the type of the result array depends only on how the array was declared, not on the type of the operand array(s). When values are taken from a `REAL` variable (simple or array) and assigned to a `SHORT` or `INTEGER` array, the values are rounded before the assignment takes place.

Dimensioning an Array

Arrays should be dimensioned before they appear in a Matrix ROM statement or function. If this is not done, errors will result. It is best to dimension arrays *explicitly* using a `DIM`, `REAL`, `SHORT`, or `INTEGER` statement. (Refer to section 8 of the HP-83 or HP-85 owner's manual for a discussion of these declarative statements.)

You can also dimension arrays *implicitly* by assigning a value to an array element, but this allows the possibility of an unexpected Error 7 (`NULL DATA`). For example, suppose your program does not contain one of the four declarative statements mentioned above, but it does assign values to the elements in the first three rows and first three columns of an array **A**. As discussed in section 8 of the owner's manual, array **A** would automatically be dimensioned as an 11×11 matrix (assuming `OPTION BASE 0` is in effect). If you were subsequently to execute a `MAT DISP` statement (described on page 17 of this manual) to display the elements of **A**, 112 `NULL DATA` error messages would be displayed in addition to the nine (3×3) values you assigned.* This is because the elements in the last eight rows and the last eight columns had been implicitly allocated storage but were never assigned values. In this example, the error messages could have been avoided by dimensioning **A** as `A(2,2)` before specifying the array in the `MAT DISP` statement.

Remember to specify `OPTION BASE 1` if you want the row and column numbers of your arrays to begin with 1 rather than 0. (Refer to section 8 of the owner's manual for a discussion of `OPTION BASE`). Throughout this manual (unless otherwise specified), examples of operations in program mode use `OPTION BASE 1`, while examples of operations in calculator mode use `OPTION BASE 0`.

Redimensioning an Array

You can reorganize an array into a more useful configuration by redimensioning it. This changes the *working size* of the array; subsequent statements affect only the elements included in the new working size. However, elements *not* included in the new working size are still associated with the array. The values of these elements are not changed, and they can be accessed if the array is redimensioned again.

```
REDIM array (redim subscripts) [ , array (redim subscripts)] ...
```

The *redim*ensioning *subscripts* are numeric expressions, variables, or constants that specify a new upper bound for each dimension. The number of subscripts must be the same as the number specified in the original `DIM`, `REAL`, `SHORT`, or `INTEGER` statement. Furthermore, the total number of elements in the new working size cannot exceed the number originally dimensioned.

Examples:

`DIM A(4),B(2,4),C(1,9)`	`OPTION BASE 0` assumed.
`REDIM A(3)`	Redimensions working size from five to four elements.
`REDIM B(3,2)`	Redimensions **B** from 3×5 matrix (15 elements) into 4×3 matrix (12 elements).
`REDIM A(4),B(2,4)`	Redimensions **A** and **B** back to original sizes.
`X=2 @ REDIM C(2*X-1,10/X-1)`	Redimensions **C** from 2×10 matrix into 4×5 matrix.

When a matrix is redimensioned, the values of its elements are reassigned to different positions within the matrix. Values of matrix elements are stored in order from left to right along each row, from the first row to the last. The redimensioning takes the elements out of the matrix in that order, and reassigns them in accordance with the new working size of the matrix.

---

* You could terminate the display of these messages—as well as halt program execution—by pressing (RESET).

The following example shows how values of matrix elements are reassigned when a matrix originally declared to be 3 × 3 is redimensioned into a 2 × 2 matrix. The values of the original matrix elements are integers that indicate the order in which the elements are stored before the redimensioning.

Example:

```
10 OPTION BASE 1
20 DIM A(3,3)

30 DATA 1,2,3,4,5,6,7,8,9
40 FOR I=1 TO 3
50 READ A(I,1),A(I,2),A(I,3)
60 NEXT I
70 K=3 @ GOSUB 130
80 REDIM A(2,2)
90 K=2 @ GOSUB 130
100 REDIM A(3,3)
110 K=3 @ GOSUB 130
120 END
130 FOR I=1 TO K
140 FOR J=1 TO K
150 DISP A(I,J);
160 NEXT J
170 DISP
180 NEXT I
190 DISP
200 RETURN
```

Array **A** is originally dimensioned as 3 × 3.

Assigns values 1 through 9 to the nine elements of matrix **A**.

Displays original 3 × 3 matrix.
Redimensions **A** down to a 2 × 2 matrix.
Displays redimensioned 2 × 2 matrix.
Redimensions **A** back up to a 3 × 3 matrix.
Displays redimensioned 3 × 3 matrix.

Subroutine that displays matrix **A**.

(RUN)

```
1   2   3
4   5   6
7   8   9


1   2
3   4


1   2   3
4   5   6
7   8   9
```

Nine elements of original 3 × 3 matrix.

Four elements of redimensioned 2 × 2 matrix. Values of elements have been reassigned sequentially within new working size.

Nine elements of 3 × 3 matrix with original values still assigned.

Note that redimensioning a matrix does *not* isolate a submatrix. In other words, if you redimension a 3 × 3 matrix into a 2 × 2 matrix, the resulting matrix is *not* the 2 × 2 submatrix from the upper left corner of the original matrix. (A submatrix *can* be isolated using a different Matrix ROM statement; refer to Copying From/Into a Subarray, page 40.)

REDIM is not the only statement that redimensions an array. The MAT ... CON, MAT ... ZER, and MAT ... IDN statements allow you optionally to specify redimensioning subscripts. These statements assign certain values to the array specified. If redimensioning subscripts are specified, the array is redimensioned before the assignment is done.

Furthermore, redimensioning may also occur with all Matrix ROM statements that specify both a result array and an operand array. In each case, the result array is redimensioned (if necessary) to accommodate the elements of the operand array before the new values are assigned. The number of rows in the result array will then equal the number of rows in the operand array, and the same is true for the number of columns. (This occurs in example 2 on page 42 and in the example on page 45.) If the size of the result array is greater than that of the operand array, the result array is first redimensioned downward to the size of the operand array. Conversely, if the current size of the result array is smaller than that of the operand array, the result array is first redimensioned upward to the size of the operand array; but this requires that the size of the result array *when originally dimensioned* was at least as large as the current size of the operand array.

Note: If an array has been redimensioned—either explicitly (that is, in a REDIM, MAT ... CON, MAT ... ZER, or MAT ... IDN statement) or implicitly (any other statement)—the array remains redimensioned even when the program that originally dimensioned it is run again. The array is *not* automatically dimensioned back to the original size declared in the program's DIM, REAL, SHORT, or INTEGER statement. If a program is rerun, and it contains an array that is redimensioned (either in the program or from the keyboard), a REDIM statement that specifies the original size should be included in the program between the DIM, REAL, SHORT, or INTEGER statement and the first statement or function that specifies the array.

Furthermore, if redimensioning occurs in a program, values of all variables (simple and array) can be traced only by executing TRACE VAR (or TRACE ALL) in the program, *not* from the keyboard.

Assigning Values to Array Elements

Assigning Values From the Keyboard

```
MAT INPUT array [ , array] ...
```

When this statement is executed, the computer displays the name of the first element of the array—for example, $R(0,0)$? (if OPTION BASE 0 is in effect) or $R(1,1)$? (if OPTION BASE 1 is in effect). You can then enter the value by typing it in and pressing [END LINE]. You can also enter values for several consecutive elements, separated by commas; if you do, the total number of characters (including spaces) cannot exceed 95—nearly three full lines.

All elements are assigned values in order from left to right on each row, from the first row to the last. After you press [END LINE], the computer displays the name of the next element to be assigned a value.

Values can be entered as numbers, as numeric variables, or as numeric expressions.

The MAT INPUT statement is programmable only; it cannot be executed from the keyboard.

Example:

```
 10 OPTION BASE 1
 20 DIM R(3),S(5,4),T(2,3)
•30 MAT INPUT R,S
 40 X=2 @ Y=5
•50 MAT INPUT T
 60 END
```

[RUN]

`R(1)?`	Computer prompts for first element of **R**.
`1.5`	Input value into first element of **R**.
`R(2)?`	
`2.5`	Input value into second element of **R**.
`R(3)?`	
`3.5,101,102,103,104`	Input values into third element of **R** and first four consecutive elements of **S**.
`S(2,1)?`	Computer prompts for next element to be assigned a value.
`201,202,203,204,301,302,303,304,` `401,402,403,404,501`	Input values into next 13 consecutive elements of **S**.

```
S(5,2)?

502,503,504
T(1,1)?
X,SIN(X),1-COS(2*X)
T(2,1)?
Y,COS(Y),1-SIN(2*Y)
```

Computer prompts for next element to be assigned a value.

Input values into last three elements of **S**.

Computer prompts for first element of **T**.

Input values, as expressions, into elements of **T**. Values are computed before assignment using values of X and Y assigned in statement 40.

Assigning Values in a Program

```
MAT READ array [ , array ] ...
```

This statement is used in conjunction with one or more DATA statements. When MAT READ is executed, elements of the array are assigned values from the list of numbers in a DATA statement. Array elements are assigned values in order from left to right on each row, from the first row to the last. Values are read from DATA statements as described in section 8 of the HP-83 and HP-85 owner's manuals. Remember that the items in a DATA statement that correspond to array elements must be numbers, not strings.

The MAT READ statement is programmable only; it cannot be executed from the keyboard.

Example:

```
 10 OPTION BASE 1
 20 INTEGER N(2,5)
 30 DIM A$[11],B$[12]
 40 DATA 1920,1930,1940,1950,1960
 50 DATA 14,38,48,62,87
 60 DATA "MILLIONS OF","U.S. DRIV
    ERS"
•70 MAT READ N
 80 READ A$,B$
 90 PRINT A$
100 PRINT B$
110 PRINT
120 FOR I=1 TO 5
130 PRINT N(1,I);N(2,I)
140 NEXT I
150 END
```

N is declared as an integer matrix.
A$ and B$ are string variables, not arrays.
Years.
Numbers of U.S. drivers.
Title.

Reads data matrix.
Reads title.

```
RUN
```

```
MILLIONS OF
U.S. DRIVERS
  1920   14
  1930   38
  1940   48
  1950   62
  1960   87
```

Assigning the Values 1 and 0

```
MAT result array = CON [ (redim subscripts) ]
```

```
MAT result array = ZER [ (redim subscripts) ]
```

`MAT ... CON` assigns the value 1 to all elements of the result array.

`MAT ... ZER` assigns the value 0 to all elements of the result array. A matrix in which all elements are zero is frequently called a *zero matrix*. Likewise, a vector of which all elements (components) are zero is frequently called a *zero vector*.

Examples:

```
DIM A(2,2),B(8)
MAT A = CON
MAT B = ZER
```

`OPTION BASE 0` assumed.
Assigns value 1 to all nine elements of **A**.
Assigns value 0 to all nine elements of **B**.

Example:

```
 10 OPTION BASE 1
 20 DIM C(4,3)

● 30 MAT C = ZER
 40 K=4 @ L=3 @ GOSUB 100
● 50 MAT C = CON(3,2)



 60 K=3 @ L=2 @ GOSUB 100
 70 REDIM C(4,3)
 80 K=4 @ L=3 @ GOSUB 100
 90 END
100 FOR I=1 TO K
110 FOR J=1 TO L
120 DISP C(I,J);
130 NEXT J
140 DISP
150 NEXT I
160 DISP
170 RETURN
```

Array **C** originally dimensioned as 4×3 matrix.
Assigns value 0 to all 12 elements of **C**.
Displays **C**.
Redimensions **C** down to a 3×2 matrix, then assigns value 1 to the six elements in current working size. Values of remaining six elements are not changed.
Displays **C**.
Redimensions **C** back up to a 4×3 matrix.
Displays **C**.

(RUN)

```
0   0   0
0   0   0
0   0   0
0   0   0
```
Value 0 assigned to all 12 elements of array **C**.

```
1   1
1   1
1   1
```
C redimensioned down to 3 × 2 matrix, and value 1 assigned to six elements in current working size.

```
1   1   1
1   1   1
0   0   0
0   0   0
```
C redimensioned back up to original size. Value 0 still assigned to remaining six elements.

Assigning the Value of a Numeric Expression

MAT *result array* = (*numeric expression*)

This statement assigns the value of the numeric expression to every element of the result array.

Examples:

```
MAT X = (30.48)
MAT Y = (M)

MAT Z = (2*PI*R(I)^2)
```

Assigns value 30.48 to all elements of **X**.
Assigns value of variable M to all elements of **Y**.
Assigns value of $2\pi R_i^2$ to all elements of **Z**.

Creating an Identity Matrix

MAT *result matrix* = IDN [(*redim subscripts*)]

MAT ... IDN assigns the value 1 to all diagonal elements of the result matrix and assigns the value 0 to all other elements. (Diagonal elements are those for which the row subscript is equal to the column subscript.) A matrix created by the MAT ... IDN statement is frequently called an *identity matrix* or *unit matrix*.

The array named must be a square matrix (after redimensioning, if necessary)—that is, it must have two dimensions, and the number of rows must be the same as the number of columns.

Examples:

```
DIM I(4,4),J(1,5)
MAT I = IDN
MAT J = IDN(2,2)
```

OPTION BASE 0 assumed.
Defines **I** as 5×5 identity matrix.
Redimensions **J** from a 2×6 matrix into a 3×3 matrix, then defines that 3×3 matrix as an identity matrix. Values of remaining three elements are not changed.

Section 3

Displaying and Printing Arrays

The Matrix ROM provides two kinds of statements for displaying and for printing arrays. The two kinds, like the `DISP` and `DISP USING` or `PRINT` and `PRINT USING` statements provided in the HP-83/85, differ in the degree of control you have over the format in the display or printout.

`MAT DISP` and `MAT PRINT` give you three convenient display/print formats.

```
MAT DISP [ ROW ] array [ , [ ROW ] array] ... [ , ]
         [ COL ]        [   [ COL ]        ]   [   ]
```

```
MAT PRINT [ ROW ] array [ , [ ROW ] array] ... [ , ]
          [ COL ]        [   [ COL ]        ]   [   ]
```

The terminator (semicolon, comma, or slash) following the array name is used to specify the spacing between elements of the array.

Terminator	Spacing Between Elements
;	Close; elements will be separated by two spaces. A minus sign, if present, occupies one of these two spaces.
,	Wide; elements will be placed in 21-column fields. The number of fields per line depends on the line length of the system printer.
/	One element per line.

If no terminator appears after the last array specified, the elements of that array will appear spaced as they would if you had specified a comma after the array name.

`MAT PRINT` outputs to the `PRINTER IS` printer. If no `PRINTER IS` printer has been declared, output defaults to the CRT on the HP-83, and to the internal printer on the HP-85.

If you specify `ROW` before an array name, elements are displayed or printed on each line by rows, beginning with row 1. Each row begins on a new line, and the elements in each row are listed in order from the first column to the last. More than one line may be required to list the elements in each row; this depends on the terminator following the array name, the number of elements in each row, the number of digits in the values of the elements, and the printer line width.

If you specify `COL` before an array name, elements are displayed or printe don each line by columns, beginning with column 1. Each column begins on a new line; and the elements in each column are listed in order from the first row to the last. Again, more than one line may be required to list the elements in each column; this depends on the terminator following the array name, the number of elements in each column, the number of digits in the values of the elements, and the printer line width.

17

Specifying neither ROW nor COL before an array name has the same effect as specifying ROW.

If more than one array is specified, a blank line appears between the display or printout of each array.

Examples:

```
MAT DISP F

MAT PRINT G;
MAT PRINT H, I/


MAT DISP ROW J; COL K;


MAT PRINT L/ COL M
```

Displays array **F** by rows with wide spacing.

Prints array **G** by rows with close spacing.

Prints array **H** by rows with wide spacing and array **I** by rows with one element per line.

Displays array **J** by rows with close spacing and array **K** by columns with close spacing.

Prints array **L** by rows with one element per line and array **M** by columns with wide spacing.

Normally, vectors are displayed or printed with one element per line. If you specify COL before the vector name, however, elements of the vector are displayed or printed across a line.

Example:

```
10 OPTION BASE 1
20 DIM V(9)
30 DATA 1,2,3,4,5,6,7,8,9
40 MAT READ V
●50 MAT PRINT COL V;
60 END
```

(RUN)

```
1   2   3   4   5   6   7   8   9
```

You can achieve more complete control of the spacing between array elements with the MAT DISP USING and MAT PRINT USING statements.

$$\text{MAT DISP USING } \textit{format string} \left[\; ; \left[\begin{matrix} \text{ROW} \\ \text{COL} \end{matrix}\right] \textit{array} \left[\; , \left[\begin{matrix} \text{ROW} \\ \text{COL} \end{matrix}\right] \textit{array} \right] \dots \right]$$

$$\text{MAT DISP USING } \textit{statement number} \left[\; ; \left[\begin{matrix} \text{ROW} \\ \text{COL} \end{matrix}\right] \textit{array} \left[\; , \left[\begin{matrix} \text{ROW} \\ \text{COL} \end{matrix}\right] \textit{array} \right] \dots \right]$$

$$\text{MAT PRINT USING } \textit{format string} \left[\; ; \left[\begin{matrix} \text{ROW} \\ \text{COL} \end{matrix}\right] \textit{array} \left[\; , \left[\begin{matrix} \text{ROW} \\ \text{COL} \end{matrix}\right] \textit{array} \right] \dots \right]$$

$$\text{MAT PRINT USING } \textit{statement number} \left[\; ; \left[\begin{matrix} \text{ROW} \\ \text{COL} \end{matrix}\right] \textit{array} \left[\; , \left[\begin{matrix} \text{ROW} \\ \text{COL} \end{matrix}\right] \textit{array} \right] \dots \right]$$

The first form of each statement includes a format string that specifies how array elements are displayed or printed. The second form of each statement specifies the line number of an IMAGE statement that includes the format string. Section 10 of the HP-83 and HP-85 owner's manuals describes the contents of format strings and their results.

As with the MAT DISP and MAT PRINT statements, specifying COL before the array name causes elements to be displayed or printed in order from top to bottom along each column, from the first column to the last. Otherwise, elements are displayed or printed in order from left to right along each row, from the first row to the last.

Examples:

```
 10 OPTION BASE 1
 20 DIM A(4,3),B(10),C(4,5)
    :
 80 MAT PRINT USING "2X,3D.2D"; A
    :
140 IMAGE "ten codes:"/10(DDD)
150 MAT PRINT USING 140 ; COL B
    :
190 IMAGE 3(4(2D,X),X,3D)/
200 MAT PRINT USING 190 ; C
    :
```

Prints matrix **A** (by rows) using format string.

Prints title on one line, then vector **B** on the next line (by columns) using IMAGE statement.

Prints matrix **C** (by rows) using IMAGE statement.

RUN

```
   126.40      5.40   243.30
   364.40    248.20   215.70
   548.90    548.60    18.50
    75.00     10.30   518.10
```
Matrix **A**.

```
ten codes:
  48 21 94   4 18 44 27 98 72 69
```
Vector **B**.

```
25 23 17 12    77
17 13 11  7    48
21 18 12 13    64

63 54 40 32   189
```
Matrix **C**.

Array Operations

Array Transpose

```
MAT result array = TRN (operand array)
```

This statement finds the transpose of the operand array. The result array will contain the same elements as the operand array, but the rows and columns will be interchanged.

Example:

```
   10 OPTION BASE 1
   20 DIM A(2,3), B(5,5)
   30 DATA 1,2,3,4,5,6
   40 MAT READ A
   50 MAT PRINT A;
   60 PRINT
•  70 MAT B = TRN(A)
   80 MAT PRINT B;
   90 PRINT
• 100 MAT A = TRN(A)
  110 MAT PRINT A;
  120 END
```

Line 70 — Sets array **B** equal to transpose of array **A**.

Line 100 — An array can be replaced by its transpose.

(RUN)

```
1   2   3
4   5   6
```

Array **A**.

```
1   4
2   5
3   6
```

Array **B**, equal to transpose of **A**. Notice that **B** has been redimensioned from a 5 × 5 matrix down to a 3 × 2 matrix.

```
1   4
2   5
3   6
```

Array **A**, now equal to transpose of original array. **A** has been redimensioned from a 2 × 3 matrix into a 3 × 2 matrix.

Scalar Operations

MAT *result array* $=$ $(scalar)$ $\begin{matrix} + \\ - \\ * \\ / \end{matrix}$ *operand array*

Scalar operation statements perform arithmetic operations between a scalar (a number, numeric variable, or numeric expression) and each element of the operand array. The resulting values are assigned to the corresponding elements of the result array.

Example:

```
10 OPTION BASE 1
20 DIM A(4,4)
30 MAT A = IDN                    Defines A as an identity matrix.
40 MAT A = (2) * A                Multiplies all elements of A by 2.
50 MAT PRINT A;
60 END
```

RUN

```
2  0  0  0                        Array A.
0  2  0  0
0  0  2  0
0  0  0  2
```

If you need to change the sign of all elements in a matrix, you can do so simply with a statement of the following form:

MAT *result array* $=$ $-$ *operand array*

For example, modify the program above by inserting the following statement:

```
45 MAT A = -A
```

RUN

```
-2  0  0  0                       Array A, with signs of elements changed.
0 -2  0  0
0  0 -2  0
0  0  0 -2
```

Arithmetic Operations

$$\text{MAT } result\ array = operand\ array\ 1 \begin{matrix} + \\ - \\ \cdot \\ / \end{matrix} operand\ array\ 2$$

Arithmetic operation statements perform arithmetic operations (addition, subtraction, multiplication, division) between corresponding elements of the two operand arrays. The resulting values are assigned to the corresponding elements of the result array.

Multiplication of corresponding elements is denoted by a period (.); the asterisk (＊) is used to denote matrix multiplication, which is a different kind of operation. (Matrix multiplication is described on page 25.)

The two operand arrays must have the same number of elements in each dimension.

Example: Firebird Stove Works employs three welders building wood stoves. The table below shows for each welder the monthly efficiency goal (in average units per day), the actual number of units produced, and the number of days worked. Write a program that calculates how far the welders exceeded or fell short of their goals.

Welder	Efficiency Goal			Units Produced			Days Worked		
	JAN	FEB	MAR	JAN	FEB	MAR	JAN	FEB	MAR
A	3.5	3.8	4.0	75	72	81	21	19	20
B	2.7	3.1	3.4	53	55	58	21	19	17
C	2.2	2.3	2.5	33	35	38	16	17	15

Solution: In the following program, the efficiency goals are entered in matrix **G**, the number of units in matrix **U**, and the number of days in matrix **D**. The actual efficiency rates are calculated by dividing each element of **U** by the corresponding element of **D**. Finally, these rates are compared to the goals by subtracting from each element of the resulting matrix the corresponding element of **G**.

```
 10 OPTION BASE 1
 20 DIM G(3,3),U(3,3),D(3,3),E(3,
    3)
 30 DATA 3.5,3.8,4.0,2.7,3.1,3.4,
    2.2,2.3,2.5
 40 DATA 75,72,81,53,55,58,33,35,
    38
 50 DATA 21,19,20,21,19,17,16,17,
    15
 60 MAT READ G,U,D
●70 MAT E = U/D
●80 MAT E = E-G
 90 PRINT "WORKER JAN  FEB  MAR"
100 MAT PRINT USING 110 ; E
110 IMAGE /3X"A"X,3(XSZ.D),3X"B"
    X,3(XSZ.D),3X"C"X,3(XSZ.D)
120 END
```

Line 70: Sets elements of matrix **E** equal to actual efficiency rates.

Line 80: Sets elements of matrix **E** equal to difference between actual efficiency rates and efficiency goals.

RUN

```
WORKER JAN  FEB  MAR

   A  +0.1 -0.0 +0.1
   B  -0.2 -0.2 +0.0
   C  -0.1 -0.2 +0.0
```

Performance above (+) or below (−) efficiency goal.

The results of two scalar multiplications can be added in one statement. This saves the storage space that would otherwise be allocated to the results of each multiplication.

MAT *result array* = (scalar) * *operand array 1* + (scalar) * *operand array 2*

Example:

```
 10 OPTION BASE 1 @ DEG
 20 DIM A(2,4),B(2,4)
 30 DATA 12,52,76,33,81,70,72,14
 40 MAT READ A
 50 MAT B=(50) @ B(1,2),B(2,1)=0
●60 MAT A=(.7)*A+(.3*SIN(60))*B
 70 MAT DISP USING"X,DD.D";A
 80 END
```

Lines 30-40: Specifies matrix **A**.

Line 50: Specifies matrix **B**.

Line 60: Adds multiples of **A** and **B**.

RUN

```
21.4 36.4 66.2 36.1
56.7 62.0 63.4 22.8
```

Linear combination of **A** and **B**.

Multiplying or dividing the results of two scalar multiplications cannot be done in one statement. However, subtracting the results of two scalar multiplications can be accomplished in one statement by changing the sign of the second scalar.

Example: In the preceding example, change statement 60 to

```
60 MAT A=(.7)*A+(-.3*SIN(60))*B
```
Subtracts multiples of **A** and **B**.

RUN

```
-4.6 36.4 40.2 10.1
56.7 36.0 37.4 -3.2
```

Matrix Multiplication

> MAT *result array = operand array 1 * operand array 2*

This statement calculates the product of two arrays, such as the product **A = BC**. The value of each element of the result array is determined according to the usual rules of matrix multiplication.

The number of columns in the first operand array must be the same as the number of rows in the second operand array. The result array has the same number of rows as the first operand array and the same number of columns as the second operand array.

Either (but not both) of the operand arrays can be vectors.

Example: The Whackit Racket Company is considering raising the prices on each of its four models. Using the data in the following table, calculate and print a matrix that shows the total income (in thousands of dollars) in each of the three sales regions at the old and at the new prices. (The price increase is not expected to affect the number of units sold.)

Monthly Sales Forecast (Thousands of Units)

Sales Region	Model			
	WR01	WR02	WR03	WR04
East	25	23	17	12
Midwest	17	13	11	7
West	21	18	12	13

Price (Per Unit)

Model	Old	New
WR01	$10	$15
WR02	$20	$27
WR03	$35	$50
WR04	$60	$80

Solution: In each sales region, the total income (either at the old or at the new prices) can be determined by multiplying the quantity by the price for each model, then adding the results. Applying this process to the data in the forecast and price tables above, we multiply each entry in a row of the forecast table by the corresponding entry in a column of the price table and then add the results. The sum could be entered into the *same* row and column of another table, in which each row shows the total income in a sales region and each column shows the total income at the old or at the new prices. Since all this is just what happens in matrix multiplication, these calculations can be done compactly with the matrix multiplication $C = AB$, where:

Matrix A contains the sales forecasts (in thousands of units). The three rows correspond to the three sales regions; the four columns correspond to the four models.

Matrix B contains the prices (per unit). The four rows correspond to the four models; the two columns correspond to the two price lists (old and new).

Matrix C will contain the total income in each sales region at the old and at the new prices. The three rows will correspond to the three sales regions; the two columns will correspond to the two price lists.

```
10 OPTION BASE 1
20 DIM A(3,4),B(4,2),C(3,2)
30 DATA 25,23,17,12            Sales forecast for East region.
40 DATA 17,13,11,7             Midwest region.
50 DATA 21,18,12,13            West region.
60 DATA 10,15,20,27,35,50,60,80   Cost matrix.
70 MAT READ A,B
80 MAT C = A*B
90 PRINT " OLD K$    NEW K$"
100 PRINT
110 IMAGE X,DC3D,4X,DC3D
120 MAT PRINT USING 110 ; C
130 END
```

(RUN)

```
OLD K$      NEW K$


2,025       2,806
1,235       1,716
1,770       2,441
```

Income in East region.
Midwest region.
West region.

You can multiply the transpose of an array by an array using just one statement (as well as two):

MAT *result array* = TRN (*operand array 1*) ✳ *operand array 2*

MAT *result array* = *operand array 1* ✳ TRN (*operand array 2*)

The two operand arrays can be the same array.

Example: Since the manufacturing capacity of the Whackit Racket Company is limited this quarter, it can produce only a percentage of the rackets demanded. The table below shows the percentage that will be supplied to each region in the next two months. Using the forecast data in the first table on page 26, calculate and print a matrix showing how many of each racket model will be produced each month.

Production Quota (Percentage)

Sales Region	June	July
East	80	90
Midwest	75	85
West	85	95

Solution: The quantity of each racket model that will be produced (during either month) can be determined by multiplying the quantity by the percentage for each model, then adding the results. As in the preceding example, these calculations can be done compactly with a matrix multiplication of the data in the sales forecast table and the data in the production quota table. To do so, however, requires that the multiplication use the transpose either of the matrix containing the forecasts or of the matrix containing the quotas. In the following program, we multiply the transpose of the matrix containing the forecasts by a matrix containing the quotas.

```
 10 OPTION BASE 1
 20 DIM A(3,4),B(3,2),C(4,2)
 30 DATA 25,23,17,12
 40 DATA 17,13,11,7
 50 DATA 21,18,12,13
 60 DATA 80,90,75,85,85,95
 70 MAT READ A,B
 80 MAT B = (.01)*B
•90 MAT C = TRN(A)*B
100 PRINT " JUNE        JULY"
110 PRINT "K-UNITS  K-UNITS"
120 PRINT
130 IMAGE X,2D.D,5X,2D.D
140 MAT PRINT USING 130 ; C
150 END
```

Sales forecast for East region.
Midwest region.
West region.
Production quota matrix.

Converts percentages to decimal values.

(RUN)

```
 JUNE        JULY
K-UNITS  K-UNITS

 50.6      56.9
 43.5      48.9
 32.1      36.1
 25.9      29.1
```

Model WR01.
Model WR02.
Model WR03.
Model WR04.

Cross Product

MAT *result vector* = CROSS (*operand vector 1 , operand vector 2*)

The MAT ... CROSS statement calculates the *cross product* (or *vector product*) of two 3-element (3-component) vectors. Mathematically, the cross product of two vectors is expressed as $\mathbf{A} = \mathbf{B} \times \mathbf{C}$.

Each of the arrays named in the MAT ... CROSS statement must be vectors; that is, they must have only one dimension. Arrays dimensioned like A(3,1) are not allowed.

Example: A leaning tree is guyed to the corner of a house as shown in the illustration. What is the moment of the force exerted by the guy cable about the base of the tree, if the tension in the wire is 960 lb?
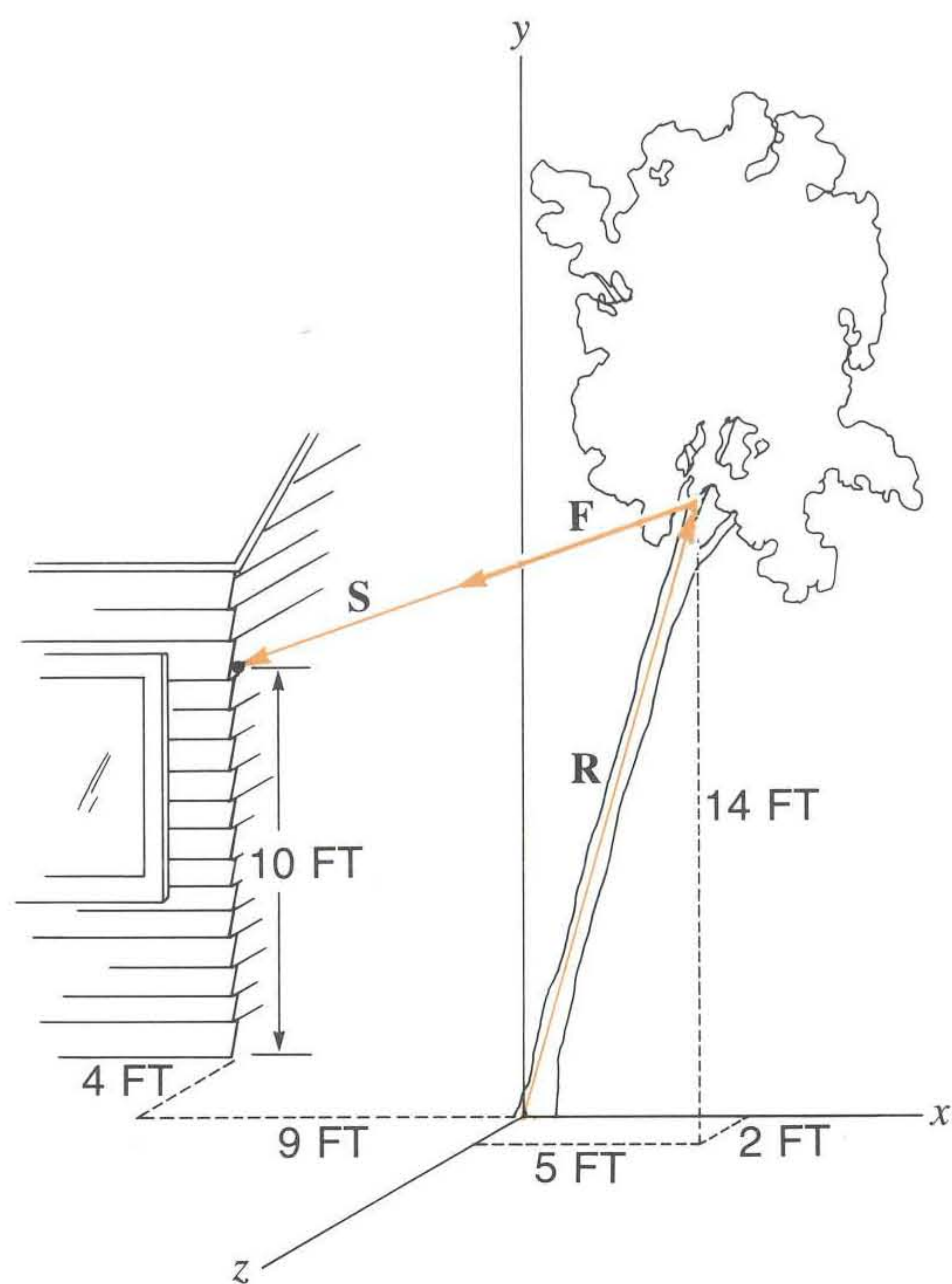
Solution: The moment is given by the cross product

$$\mathbf{M} = \mathbf{R} \times \mathbf{F}$$

where $\mathbf{R}$ is the position vector of the guy point (on the tree) with respect to the base of the tree, and $\mathbf{F}$ is the 960-lb force exerted by the guy cable. In the following program, vectors $\mathbf{R}$, $\mathbf{F}$, and $\mathbf{S}$ are expressed in terms of their components in the x–, y–, and z–directions. $\mathbf{S}$ is the vector from the guy point on the tree to the guy point on the house.

The components of $\mathbf{R}$, as can be seen in the illustration, are:

$$R_x = 5 \qquad R_y = 14 \qquad R_z = 2$$

The components and lengths of $\mathbf{F}$ and of $\mathbf{S}$ are proportional; that is,

$$\frac{F_x}{S_x} = \frac{F_y}{S_y} = \frac{F_z}{S_z} = \frac{\|\mathbf{F}\|}{\|\mathbf{S}\|} \ .$$

Therefore, each component of $\mathbf{F}$ can be calculated by multiplying the corresponding component of $\mathbf{S}$ by the ratio of the length of $\mathbf{F}$ to the length of $\mathbf{S}$. This is done simultaneously for all three components of $\mathbf{F}$ in statement 60 below. L, the length of $\mathbf{S}$, can be calculated as in statement 50 below (but it can be calculated more efficiently using the Matrix ROM's `FNORM` function; refer to section 6). The components of $\mathbf{S}$, as can be seen in the illustration, are:

$$S_x = -9 - 5 = -14 \qquad S_y = 10 - 14 = -4 \qquad S_z = -4 - 2 = -6$$

```
   10 OPTION BASE 1
   20 DIM R(3), F(3), S(3), M(3)
   30 DATA 5,14,2,-14,-4,-6
   40 MAT READ R,S

   50 L = SQR(S(1)^2+S(2)^2+S(3)^2)
   60 MAT F = (960/L) * S
 • 70 MAT M = CROSS(R,F)
   80 MAT PRINT USING "5D.DD"; M
   90 END
```

(RUN)

```
 -4632.96
   121.92
 10728.97
```

Reads components of **R** and **S** from DATA statement.

Calculates length of **S**.
Calculates components of **F**.
Calculates cross product.
Prints components of moment **M**.

x–component of **M** (in lb-ft).
y–component of **M**.
z–component of **M**.

Inverting a Matrix

MAT *result matrix* = INV *(operand matrix)*

This statement finds the inverse of the operand matrix. The inverse of the operand matrix is the matrix that, when matrix-multiplied by the operand matrix, results in an identity matrix.

The operand matrix must be square—that is, the number of rows must be the same as the number of columns.

Example: Find the inverse of the matrix shown below. Check that when the inverse is multiplied by the matrix itself, the result is an identity matrix.

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$$

```
   10 OPTION BASE 1
   20 DIM A(2,2),B(2,2),C(2,2)
   30 DATA 2,3,4,5
   40 IMAGE 4(4D.D)/
   50 MAT READ A
   60 MAT PRINT USING 40 ; A
 • 70 MAT B = INV(A)
   80 MAT PRINT USING 40 ; B
   90 MAT C = B*A
  100 MAT PRINT USING 40 ; C
  110 END
```

Elements of matrix **A**.

Sets matrix **B** equal to inverse of **A**.

Checks that product is identity matrix.

RUN

```
  2.0    3.0
  4.0    5.0
```
Matrix **A**.

```
 -2.5    1.5
  2.0   -1.0
```
Inverse of **A**.

```
  1.0    0.0
  0.0    1.0
```
Product of **A** and inverse of **A** is an identity matrix.

When the determinant of a matrix is zero, the matrix does not have an inverse. Therefore, if you attempt to find the inverse of such a matrix using the MAT ... INV statement, the result will be meaningless. You can use the DET (determinant) function to check the determinant before inverting a matrix, or use the DETL function after inverting. (Refer to page 49.)

You can multiply the inverse of a matrix by an array using just one statement (as well as two):

> MAT *result array* = INV (*operand array 1*) * *operand array 2*

Since this performs both an inversion and a matrix multiplication in one statement, the result is calculated with somewhat more accuracy than if it were calculated in two separate statements.

Example: The following program calculates the inverse of the same matrix as the preceding program, but the multiplication to check for the identity matrix is done in only one statement.

```
 10 OPTION BASE 1
 20 DIM A(2,2),C(2,2)
 30 DATA 2,3,4,5
 40 MAT READ A
•50 MAT C = INV(A) * A
 60 MAT PRINT A; C;
 70 END
```

RUN

```
  2    3
  4    5
```
Matrix **A**.

```
  1    0
  0    1
```
Product of **A** and inverse of **A**, calculated using just one statement.

Calculating the inverse of a matrix is typically done in the process of solving the matrix equation **AX** = **B**. However, a solution still more accurate than that provided by MAT X = INV(A)*B can be obtained using the MAT ... SYS statement (described next).

Solving the Equation $\mathbf{AX} = \mathbf{B}$

MAT *result array* = SYS (*coefficient matrix , constant array*)

Given any square matrix $\mathbf{A}$ and any other array $\mathbf{B}$, this statement can be used to solve the matrix equation $\mathbf{AX} = \mathbf{B}$ for the unknown array $\mathbf{X}$.

Most often, the need for this arises when solving a system of n linear equations in n unknowns. The system

$$a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n = b_1$$
$$a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n = b_2$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{n1} x_1 + a_{n2} x_2 + \ldots + a_{nn} x_n = b_n$$

can be expressed in matrix notation as

$$\mathbf{AX} = \mathbf{B},$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \multicolumn{4}{c}{\ldots\ldots\ldots\ldots\ldots} \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}, \; \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \text{ and } \mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

The solution to this system of equations is the set of elements of array $\mathbf{X}$. $\mathbf{A}$ is the coefficient matrix; $\mathbf{B}$ is the constant array.

Example: Solve the following system of equations:

$$2x + y - z = 0$$
$$x - y + z = 6$$
$$x + 2y + z = 3$$

Solution: Expressing this system of equations in matrix notation, $\mathbf{AX} = \mathbf{B}$, where

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & -1 \\ 1 & -1 & 1 \\ 1 & 2 & 1 \end{bmatrix}, \; \mathbf{X} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \text{ and } \mathbf{B} = \begin{bmatrix} 0 \\ 6 \\ 3 \end{bmatrix}.$$

```
10 OPTION BASE 1
20 DIM A(3,3),X(3),B(3)
30 DATA 2,1,-1,1,-1,1,1,2,1
40 DATA 0,6,3
50 MAT READ A,B
•60 MAT X = SYS(A,B)
70 MAT DISP X
80 END
```

(RUN)

2	x-value of solution.
-1	y-value.
3	z-value.

As we mentioned earlier (page 31), the solution to the matrix equation $\mathbf{AX} = \mathbf{B}$ can also be obtained using the statement `MAT X = INV(A)*B`. The solution obtained using the statement `MAT X = SYS(A,B)` is somewhat more accurate; but to achieve this accuracy the Matrix ROM uses two extra blocks of memory, each the size of the array $\mathbf{X}$.

Although in typical applications the result array $\mathbf{X}$ and constant array $\mathbf{B}$ are each vectors or one-column matrices, the `MAT ... SYS` statement does not restrict these arrays to only one column. This allows you, for example, to simultaneously solve two different systems of n equations in n unknowns, provided that the coefficients in both systems of equations are identical.

Example: Your company's Publications Manager wants to determine the cost factors used by her two outside printers. She knows that each printer estimates jobs based on the number of pages and the number of photographs, plus a fixed setup charge. Given the three estimates from each printer shown below, write a program that calculates their cost per page, cost per photograph, and setup charge.
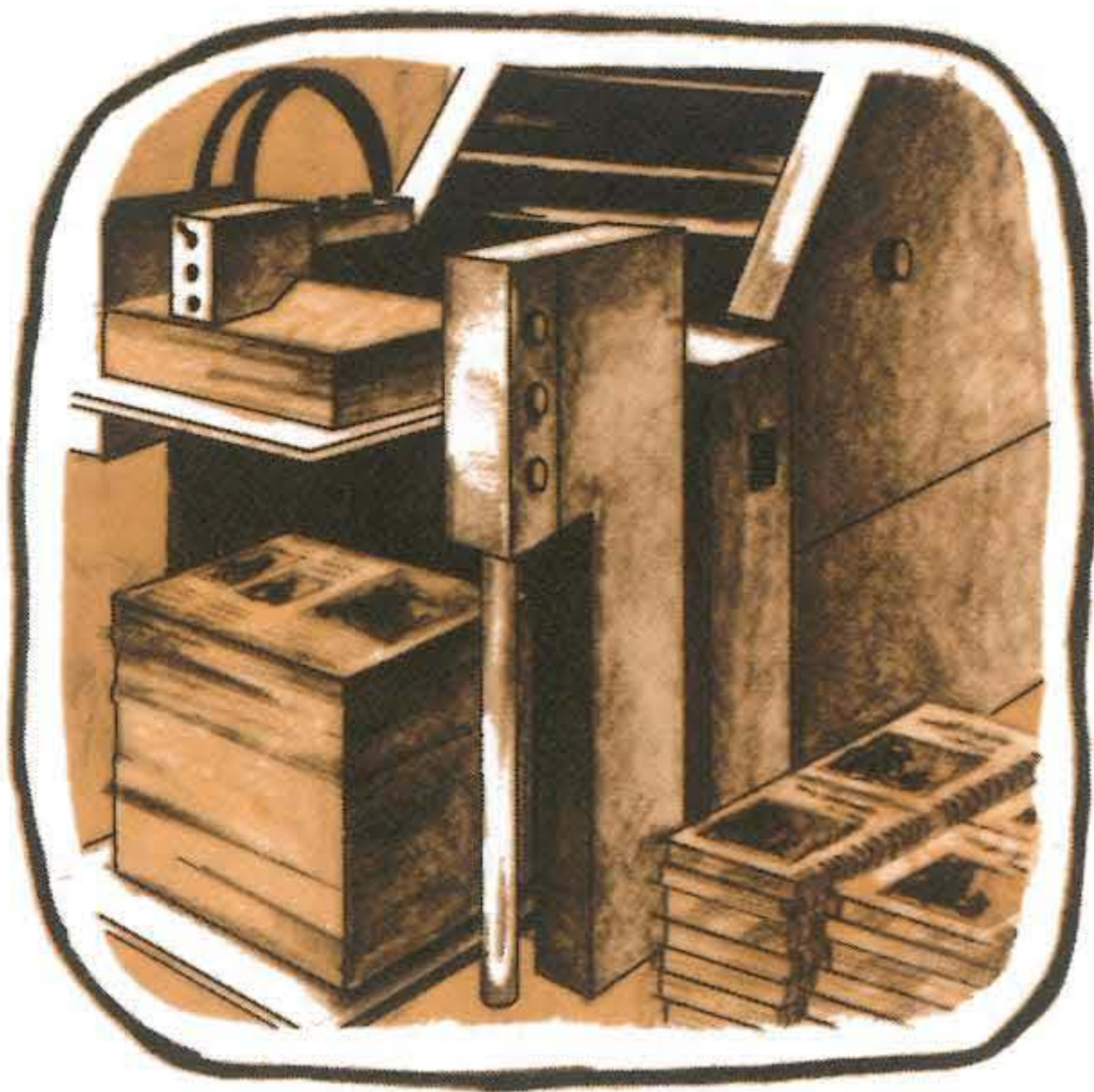
Job	Number of Pages	Number of Photographs	Total Cost Printer A	Printer B
1	273	35	$5835.00	$7362.50
2	150	8	$3240.00	$4085.00
3	124	19	$2775.00	$3517.50

Solution: We need to solve the following system of equations for two sets of cost estimates:

$$273\,x_1 + 35\,x_2 + 1\,x_3 = cost_1$$

$$150\,x_1 + 8\,x_2 + 1\,x_3 = cost_2$$

$$124\,x_1 + 19\,x_2 + 1\,x_3 = cost_3$$

These equations can be represented in matrix notation as **AX = B**, where:

A is the coefficient matrix, having the number of pages in its first column, the number of photographs in its second column, and the number of setup charges (one for each job) in its third column. Each row contains this data for a different job.

B is the constant array. Each row contains cost estimates for one job from the two printers; each column contains one printer's cost estimates for the three jobs.

X is the result array, having the unknown cost factors x_1, x_2, and x_3 in its rows. x_1 is the cost per page, x_2 is the cost per photograph, and x_3 is the setup charge. Since we are solving two systems of equations, the result array must be a matrix; that is, it should originally be declared with two dimensions. (Its size, if not the same size as that of the constant array **B**, will automatically be redimensioned down to the size of **B** before the MAT ... SYS statement is executed.) Each column will contain the cost factors for one printer.

```
 10 OPTION BASE 1
 20 DIM A(3,3),X(3,2),B(3,2)
 30 DATA 273,35,1                  Specifications for job 1.
 40 DATA 150,8,1                   Specifications for job 2.
 50 DATA 124,19,1                  Specifications for job 3.
 60 DATA 5835,7362.5               Estimates for job 1.
 70 DATA 3240,4085                 Estimates for job 2.
 80 DATA 2775,3517.5               Estimates for job 3.
 90 MAT READ A,B
●100 MAT X = SYS(A,B)
110 PRINT USING "9A,3X,9A,2/" ;
    "PRINTER A","PRINTER B"
120 MAT PRINT USING "X,3D.2D,6X,
    3D.2D" ; X
130 END
```

(RUN)

```
PRINTER A    PRINTER B

    20.00        25.00                  Cost per page.
     5.00         7.50                  Cost per photograph.
   200.00       275.00                  Setup charge.
```

Summing Rows and Columns

> MAT *result array* = RSUM (*operand array*)

> MAT *result array* = CSUM (*operand array*)

MAT ... RSUM adds the values of the elements in each row of the operand array, then assigns the sum to the corresponding element of the result array (a vector or one-column matrix). If the result array is a vector, it is first redimensioned (if necessary) to have as many elements as the number of rows in the operand array. If the result array is a matrix, it is first redimensioned (if necessary) to have one column and as many rows as in the operand array.

Likewise, MAT ... CSUM adds the values of the elements in each column of the operand array, then assigns the sum to the corresponding element of the result array (a vector or one-row matrix). If the result array is a vector, it is first redimensioned (if necessary) to have as many elements as the number of columns in the operand array. If the result array is a matrix, it is first redimensioned (if necessary) to have one row and as many columns as in the operand array.

Example: Using the Whackit Racket Company's monthly forecast data from page 26, write a program that calculates and prints the total forecast for all racket models in each region and the total forecast for each racket model in all regions.

Solution: Since each row contains the forecasts for all models in a region, the total forecast for all models in each region can be found using the MAT ... RSUM statement. Likewise, since each column contains the forecasts for one model in all regions, the total forecast for each model in all regions can be found using the MAT ... CSUM statement.

```
 10 OPTION BASE 1
 20 DIM A(3,4),R(3),C(1,4)
 30 DATA 25,23,17,12          Sales forecast for East region.
 40 DATA 17,13,11,7           Midwest region.
 50 DATA 21,18,12,13          West region.
 60 MAT READ A
 70 MAT PRINT A;
 80 PRINT
 90 MAT R = RSUM(A)           Assigns row sums to vector R.
100 MAT PRINT R;
110 PRINT
120 MAT C = CSUM(A)           Assigns column sums to matrix C.
130 MAT PRINT C/
140 END
```

RUN

25	23	17	12
17	13	11	7
21	18	12	13

Matrix of monthly sales forecast (in thousands of units).

77

Total sales in East region.

48

Total sales in Midwest region.

64

Total sales in West region.

63

Total sales of model WR01.

54

Total sales of model WR02.

40

Total sales of model WR03.

32

Total sales of model WR04.

Notes

Copying Arrays and Subarrays

Copying an Array

```
MAT result array = operand array
```

This statement assigns the value of each element of the operand array to the corresponding element of the result array.

The following rules apply to copying arrays:

- If the result array is a matrix and the operand array is a matrix, the result matrix is first redimensioned (if necessary) to have the same number of rows and columns as the operand matrix.

- If the result array is a matrix and the operand array is a vector, the result matrix is first redimensioned (if necessary) to have one column and as many rows as the number of elements in the operand vector.

- If the result array is a vector, the operand array must be a vector, a one-column matrix, or a one-row matrix. The result vector is first redimensioned (if necessary) to have the same number of elements as the operand array.

Examples:

```
 10 OPTION BASE 1
 20 DIM A(5,5),B(5,5),C(3,4)
 30 DIM D(9),E(4),F(1,3)
       :
 80 MAT A = B
       :
120 MAT A = C

130 MAT B = E

140 MAT D = E

150 MAT E = F
```

Sets matrix **A** equal to matrix **B**.

Redimensions **A** from a 5×5 matrix down to a 3×4 matrix (the size of **B**), then sets **A** equal to **C**.

Redimensions **B** from a 5×5 matrix down to a 4×1 matrix (the size of **E**), then sets **B** equal to **E**.

Redimensions **D** from a 9-element vector down to a 4-element vector (the size of **E**), then sets **D** equal to **E**.

Redimensions **E** from a 4-element vector down to a 3-element vector (the size of **F**), then sets **E** equal to **F**.

Copying From/Into a Subarray

MAT *result array* [⟨ [... row number or first row number : last row number] , [... column number or first column number : last column number] ⟩]

= *operand array* [⟨ [... row number or first row number : last row number] , [... column number or first column number : last column number] ⟩]

With this statement you can copy values from and/or into a subarray (such as a partitioned matrix). Values are assigned from the specified elements of the operand array to the specified elements of the result array.

Row and column numbers, if specified after the name of the operand array or the result array, should conform to the rules listed below. After each rule is one or more references to statements illustrating the rule; these statements are shown in the examples following.

- If row and/or column numbers are specified, they must be enclosed in parentheses. (Examples 2 through 9.)

- If all elements are to be copied or assigned values, do not specify row numbers, column numbers, or parentheses after the array name. (Examples 1, 2, 6, 7, 9.) The array elements will be copied or assigned values in order from left to right along each row from the first row to the last.

- If no row or column numbers are specified after the result array, the result array is redimensioned (if necessary) before the values are assigned. (Example 2.) If row or column numbers *are* specified after the result array, values are assigned to the specified elements, but no redimensioning is done.

- If the array is a vector, specify only the row number(s). (Example 3.)

- If the array is a matrix, specify the column number(s) after the row number(s), separated by a comma. (Examples 2, 3, 4, 5, 8, 9.)

- If only one row is to be copied or assigned values, specify that one row number. (Examples 3, 4, 6.)

- If more than one row is to be copied or assigned values, specify the first row number and the last row number, separated by a colon. (Examples 2, 5, 8, 9.)

- If only one column is to be copied or assigned values, specify that one column number. (Examples 3, 5, 7.)

- If more than one column is to be copied or assigned values, specify the first column number and the last column number, separated by a colon. (Examples 2, 4, 8, 9.)

- If an entire row is to be copied or assigned values, you may omit the column numbers, but specify a comma after the row number(s). (Example 6.)

- If an entire column is to be copied or assigned values, you may omit the row numbers, but specify a comma before the column number(s). (Example 7.)

The following examples show, for each statement, the values in the operand array and the values in the result array. Values in each array that are *not* affected by the statement are shown in color. (Assume that OPTION BASE 1 is in effect and all values in the 5 × 5 array **B** are set to zero before each statement is executed.)

Example Number	Statement	Comment	Operand Array	Result Array
1.	`MAT B = A`	Copies the value from each element of array **A** into the corresponding element of array **B**.	**A** $\begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$	**B** $\begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$
2.	`MAT B = A(1:3,1:3)`	Redimensions **B** from a 5×5 matrix down to a 3×3 matrix (the size of the operand sub-matrix); then copies the values from the first through third rows in columns 1 through 3 of array **A** into the redimensioned array **B**.	**A** $\begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$	**B** $\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$
3.	`MAT B(3,2) = D(3)`	Copies the value from the third element of vector **D** into the third row in column 2 of array **B**.	**D** $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$	**B** $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
4.	`MAT B(3,1:3) = A(1,2:4)`	Copies the values from the second through fourth elements in row 1 of array **A** into the first through third elements in row 3 of array **B**.	**A** $\begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$	**B** $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 13 & 14 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$
5.	`MAT B(2:3,5) = A(4:5,1)`	Copies the values from the fourth and fifth rows in column 1 of array **A** into the second and third rows in column 5 of array **B**.	**A** $\begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$	**B** $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 41 \\ 0 & 0 & 0 & 0 & 51 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

6.

```
MAT B(3,) = D
```

Copies the vector **D** into the entire third row of array **B**.

$$D = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

7.

```
MAT D = A(,2)
```

Copies the entire second column of array **A** into the vector **D**.

$$A = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$$

$$D = \begin{bmatrix} 12 \\ 22 \\ 32 \\ 42 \\ 52 \end{bmatrix}$$

8.

```
MAT B(2:3,1:4) = A(1:2,2:5)
```

Copies the values from the first and second rows in columns 2 through 5 of array **A** into the second and third rows in columns 1 through 4 of array **B**.

$$A = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 12 & 13 & 14 & 15 & 0 \\ 22 & 23 & 24 & 25 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

9.

```
MAT C = A(1:2,3:5)
```

Copies the values from the first and second rows in columns 3 through 5 of array **A** into array **C**.

$$A = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix}$$

$$C = \begin{bmatrix} 13 & 14 & 15 \\ 23 & 24 & 25 \end{bmatrix}$$

The following additional rules apply to copying subarrays:

- Unless either the operand array or the result array is a vector, the number of rows *specified* after the result array must be the same as the number of rows to be copied from the operand array. The same is true for the number of columns in each array.

- Unless the operand array is a vector, a column from the operand array cannot be copied, *using just one statement,* into a row in the result array. This copy can, however, be done using two statements, as shown in the next example.

- Unless the result array is a vector, a row from the operand array cannot be copied, using just one statement, into a column of the result array. Again, this copy can be done using two statements.

In the following example, row 1 of array **A** is copied into column 3 of array **B**, then column 3 of array **A** is copied into row 2 of array **B**.

```
 10 OPTION BASE 1
 20 DIM A(3,3), B(3,3), C(3)
 30 DATA 1,2,3,4,5,6,7,8,9
 40 MAT READ A                      Reads values into array A.
 50 MAT DISP A;
 60 DISP
 70 MAT B = ZER                     Sets all elements of B equal to 0.
 80 MAT C = A(1,)                   Copies row 1 of matrix A into vector C.
 90 MAT B(,3) = C                   Copies vector C into column 3 of matrix B.
100 MAT DISP B;
110 DISP
120 MAT B = ZER                     Sets all elements of B again equal to 0.
130 MAT C = A(,3)                   Copies column 3 of matrix A into vector C.
140 MAT B(2,) = C                   Copies vector C into row 2 of matrix B.
150 MAT DISP B;
160 END
```

[RUN]

```
1   2   3                           Matrix A.
4   5   6
7   8   9

0   0   1                           Matrix B, with column 3 containing values
0   0   2                           from row 1 of matrix A.
0   0   3

0   0   0                           Matrix B, with row 2 containing values
3   6   9                           from column 3 of matrix A.
0   0   0
```

Example: The program for the example on page 35 prints a matrix of monthly sales forecasts followed by arrays showing the row sums (the total sales in each region) and the column sums (the total sales of each model). Modify that program so that it prints just one matrix containing (in addition to the monthly sales forecasts) the row sums of the original matrix in the fifth column, and the column sums of the original matrix in the fourth row.

```
 10 OPTION BASE 1
 20 DIM A(3,4),R(3),C(1,4),D(4,5)
 30 DATA 25,23,17,12
 40 DATA 17,13,11,7
 50 DATA 21,18,12,13
 60 MAT READ A
•70 MAT D(1:3,1:4) = A


 80 MAT R = RSUM(A)

•90 MAT D(1:3,5) = R

100 MAT C = CSUM(A)

110 MAT D(4,1:4) = C

120 MAT R = RSUM(C)




•130 MAT D(4,5) = R
140 PRINT USING 150 ; "MODEL","0
    1 02 03 04   ALL"
150 IMAGE 5X,5A/16A//
160 MAT PRINT USING 170 ; D
170 IMAGE 3(4(2D,X),X,3D)/
180 END
```

Copies 3 × 4 matrix **A** into first through third rows in columns 1 through 4 of matrix **D**.

Assigns row sums of **A** to 3-element vector **R**.

Copies **R** into first through third rows in column 5 of **D**.

Assigns column sums of **A** to 1 × 4 matrix **C**.

Copies **C** into fourth row in columns 1 through 4 of **D**.

Redimensions **R** down to one element (the number of rows in **C**), then assigns row sum of **C** (i.e., row sum of column sums of **A**) to **R**.

Copies **R** into fourth row in column 5 of **D**.

(RUN)

```
     MODEL
 01 02 03 04   ALL

 25 23 17 12    77
 17 13 11  7    48
 21 18 12 13    64

 63 54 40 32   189
```

East region.
Midwest region.
West region.

Column sums: total sales of each model.

Row sums: total sales in each region.

The row and/or column number(s) can be specified not only as constants (like those in the preceding examples), but also as variables or expressions. If you do this, this first row or column number specified may—depending on the value of the variable or expression—be greater than the second number.

An important special case occurs when the first row number specified is just one greater than the second row number, or when the first column number specified is just one greater than the second column number.* In such cases, *no elements will be copied or assigned values*. Furthermore, if no row or column numbers are specified after the result array—and OPTION BASE 1 is in effect—the result array will be redimensioned to have *zero rows* or *zero columns*.** The value of these features will become apparent after we discuss these arrays a bit more.

Examples:

```
10 OPTION BASE 1
20 DIM A(4,4),B(4,4),C(4,4)
30 MAT A = CON
40 FOR K=1 TO 3
● 50 MAT B = A(1:K-1,2)

● 60 MAT C = A(,2:K)
   :
80 NEXT K
90 END
```

OPTION BASE 1 is in effect.
Arrays **A**, **B**, and **C** are 4 × 4 matrices.

When $K = 1$, redimensions **B** down to a 0 × 1 matrix.
When $K = 1$, redimensions **C** down to a 4 × 0 matrix.

Arrays with zero rows or zero columns can be considered to be "empty," since they contain no elements: the current working size of the array is 0. Empty arrays should not be confused with arrays that have not been initialized and therefore result in NULL DATA error messages when displayed or printed. If you should display or print an empty array, there will be no output since there are no elements in the array.

Empty arrays *can* be specified in subsequent statements and functions with meaningful results; the usual rules of redimensioning (refer to page 9) and row/column matching (in statements with two operand arrays) apply. The following situations are of particular interest:

● Statements specifying only one operand array will, if that array is empty, redimension the result array to be empty. For example, if the operand array has been redimensioned to be 0 × 3, the statement MAT A = B would redimension the result array to be 0 × 3, while the statement MAT A = TRN(B) would redimension the result array to be 3 × 0.

● The matrix multiplication statement, if *both* operand arrays are empty, can yield a result array that is *not* empty. In such cases, furthermore, the statement assigns the value 0 to all elements of the result array. For example, if matrix **B** has been redimensioned to be 3 × 0, and matrix **C** has been redimensioned to be 0 × 1, the statement MAT A = B*C redimensions matrix **A** to be (3 × 0) × (0 × 1) = 3 × 1 (in accordance with the usual rules of matrix multiplication). The result array is *not* empty, since neither the number of rows nor the number of columns is zero; and the value 0 has been assigned to all three elements.

---

*If the first row number specified is more than one greater than the second row number, or if the first column number specified is more than one greater than the second column number, elements will be copied or assigned values in reverse order. The row or column numbers specified are one above and one below the numbers of the rows or columns that will be copied or assigned values. For example, the statement MAT B(2:4,5:0) = A(6:2,2:5)—with OPTION BASE 1 in effect—copies the values from the fifth through third rows in the second through fifth columns of array **A** into the second through fourth rows in the fourth through first columns of array **B**. With OPTION BASE 0 in effect, the same thing is accomplished by the statement MAT B(1:3,4:-1) = A(5:1,1:4).

**If OPTION BASE 0 is in effect, the result array is not redimensioned, and a DIM SIZE error message is generated.

The fact that no elements are copied or assigned values when the first row or column number is one greater than the second, plus the characteristics described above for the resulting empty arrays, simplify HP-83/85 programs that do certain matrix manipulations.

Example: The program segment listed below forms an orthogonal matrix from a given 3×3 matrix.* The procedure, an implementation of the Gram-Schmidt orthogonalization process, is frequently used in solutions of "least squares" problems. Each pass through the FOR-NEXT loop replaces one column of the matrix with a vector that is orthogonal with respect to each of the previous columns and also normalized. Note that during the first pass through the loop, there are no previous columns to process with the current (that is, the first) column. The features described above eliminate the necessity for including additional program statements to handle this first pass, in which statement 70 otherwise would illegally specify a nonexistent column number.

```
10 OPTION BASE 1
20 DIM A(3,3),P(3,3),T(3,3),
   V(3),X(3)
   :
50 FOR K=1 TO 3
60 MAT X = A(,K)

70 MAT P = A(,1:K-1)



80 MAT V = TRN(P)*X




90 MAT T = P*V





100 MAT X = X-T

110 N = SQR(X(1)^2+X(2)^2+X(3)^2)
120 MAT X = (1/N)*X
130 MAT A(,K) = X
140 NEXT K
   :
```

Copies Kth column of **A** into **X**. When $K = 1$, copies column 1 of **A** into **X**.

Copies columns 1 through $K - 1$ of **A** into **P**. When $K = 1$, no elements are copied and **P** is redimensioned to be 3×0.

Multiplies **X** and transpose of **P**. When $K = 1$, TRN(P) is 0×3 and **X** is 3×1, so **V** is redimensioned to be $(0 \times 3) \times (3 \times 1) = 0 \times 1$.

Multiplies **P** and **V**. When $K = 1$, **P** is 3×0 and **V** is 0×1, so **T** is redimensioned to be $(3 \times 0) \times (0 \times 1) = 3 \times 1$. Also, value 0 is assigned to all elements of **T**.

Subtracts **T** from **X**. When $K = 1$, **X** is not changed because **T** is a zero matrix.

Calculates norm of **X**.

Normalizes **X**.

Replaces column K of **A** by **X**. When $K = 1$, net effect of loop is to normalize column 1 of **A**.

---

*You need not be concerned if you are not familiar with the concepts or terminology in this example; this familiarity is not required to understand the features illustrated in the program.

Array Functions

The Matrix ROM provides 21 functions relating to arrays. Each of these array functions can be used just like any other function on your HP-83/85. The following table lists these functions and their results.

Function	Result
ABSUM (*array*)	Sum of absolute values of elements in array.[6]
AMAX (*array*)	Value of largest element in array.[8]
AMAXCOL	Column number of largest element in array most recently specified in AMAX function.[1][2][6]
AMAXROW	Row number of largest element in array most recently specified in AMAX function.[1][6]
AMIN (*array*)	Value of smallest element in array.[9]
AMINCOL	Column number of smallest element in array most recently named in AMIN function.[1][2][6]
AMINROW	Row number of smallest element in array most recently specified in AMIN function.[1][6]
CNORM (*array*)	Largest sum of absolute values of elements in each column of array (*column norm*).[6]
CNORMCOL	Column number with largest sum of absolute values in array most recently specified in CNORM function.[2][3][6]
DET (*matrix*)	Determinant of matrix.[7]
DETL	Determinant of last matrix inverted in MAT ... INV statement, or specified as first argument in MAT ... SYS statement.[7]
DOT (*vector 1 , vector 2*)	Sum of products of corresponding elements of vectors (*dot product* or *scalar product*).[5][6]
FNORM (*array*)	Square root of sum of squares of elements in array (*Frobenius norm* or *Euclidean norm*).[6]
LBND (*array , expression*)	Lower bound of array subscript (1 or 2) specified by rounded integer value of expression. LBND is equal to OPTION BASE in effect.
MAXAB (*array*)	Largest absolute value of any element in array.[6]
MAXABCOL	Column number of element with largest absolute value in array most recently specified in MAXAB function.[1][2][6]
MAXABROW	Row number of element with largest absolute value in array most recently specified in MAXAB function.[1][6]

RNORM (*array*)	Largest sum of absolute values of elements in each row of array (*row norm*).[6]
RNORMROW	Row number with largest sum of absolute values in array most recently specified in RNORM function.[4][6]
SUM (*array*)	Sum of elements in array.[6]
UBND (*array, expression*)	Upper bound of array subscript (1 or 2) specified by rounded integer value of expression.[10]

Examples:

```
 10  OPTION BASE 1
 20  DIM A(3,3),B(3,3),V1(3),
     V2(3)
 30  DATA -3,2,3,5,-3,5,2,5,-1
 40  DATA 2,1,3,1,4,2
 50  IMAGE /3D
 60  IMAGE /3D.2D
 70  MAT READ A
 80  MAT READ V1,V2
 90  MAT PRINT A;V1,V2
100  PRINT USING 50 ; ABSUM(A)
110  PRINT USING 50 ; AMAX(A)
120  PRINT USING 50 ; AMAXCOL
130  PRINT USING 50 ; AMAXROW
140  PRINT USING 50 ; AMIN(A)
150  PRINT USING 50 ; AMINCOL
160  PRINT USING 50 ; AMINROW
170  PRINT USING 50 ; CNORM(A)
180  PRINT USING 50 ; CNORMCOL
190  PRINT USING 50 ; DET(A)
200  MAT B = INV(A)
210  PRINT USING 50 ; DETL
220  PRINT USING 50 ; DOT(V1,V2)
230  PRINT USING 60 ; FNORM(A)
```

---

[1] If more than one element has the largest or smallest value or absolute value, the element in the lowest-numbered row is chosen, and this number is returned as the value of the function AMAXROW, AMINROW, or MAXABROW. If both such elements are in the same row, the element in the lowest-numbered column is chosen, and this number is returned as the value of the function AMAXCOL, AMINCOL, or MAXABCOL.

[2] If the array specified in the most recent AMAX, AMIN, CNORM, or MAXAB function is a vector, evaluating this function results in a NULL DATA error message, and the number 0 is returned as the value of the function.

[3] If more than one column has the largest sum of absolute values, the lowest-numbered column is chosen, and this number is returned as the value of CNORMCOL.

[4] If more than one row has the largest sum of absolute values, the lowest-numbered row is chosen, and this number is returned as the value of RNORMROW.

[5] The two vectors specified must have the same number of elements. One-column matrices are not allowed.

[6] If the array specified is empty (refer to page 46), the number 0 is returned as the value of the function.

[7] If the array specified is empty (refer to page 46), the number 1 is returned as the value of the function.

[8] If the array specified is empty (refer to page 46), the number $-9.99999999999E499$ is returned as the value of the function.

[9] If the array specified is empty (refer to page 46), the number $9.99999999999E499$ is returned as the value of the function.

[10] If the array specified has been redimensioned to have zero rows (refer to page 46) and the value of the second subscript is 1, the number 0 is returned as the value of the function. If the array specified has been redimensioned to have zero columns and the value of the second subscript is 2, the number 0 is returned as the value of the function.

```
240 PRINT USING 50 ; LBND(A,1)
250 PRINT USING 50 ; MAXAB(A)
260 PRINT USING 50 ; MAXABCOL
270 PRINT USING 50 ; MAXABROW
280 PRINT USING 50 ; RNORM(A)
290 PRINT USING 50 ; RNORMROW
300 PRINT USING 50 ; SUM(A)
310 N = 2
320 PRINT USING 50 ; UBND(A,N)
330 END
```

RUN

```
-3   2   3          Array A.
 5  -3   5
 2   5  -1
```

```
 2                  Vector V1.
 1
 3
```

```
 1                  Vector V2.
 4
 2
```

29   ABSUM(A): the sum of the absolute values of the elements in A.

 5   AMAX(A): the value of the largest element in A.

 1   AMAXCOL: the lowest-numbered column containing AMAX(A).

 2   AMAXROW: the lowest-numbered row containing AMAX(A).

-3   AMIN(A): the value of the smallest element in A.

 1   AMINCOL: the lowest-numbered column containing AMIN(A).

 1   AMINROW: the lowest-numbered row containing AMIN(A).

10   CNORM(A): the largest sum of the absolute values of the elements in each column of A.

 1   CNORMCOL: the lowest-numbered column with the largest sum of absolute values.

189   DET(A): Determinant of A.

189   DETL: Determinant of matrix (A) inverted in preceding MAT ... INV statement.

12   DOT(V1,V2): the sum of the products of corresponding elements of V1 and V2.

10.54   FNORM(A): the square root of the sum of the squares of the elements in A.

1   LBND(A,1): the lower bound of the first subscript of **A**.

5   MAXAB(A): the largest absolute value of any element in **A**.

1   MAXABCOL: the lowest-numbered column containing the element with largest absolute value.

2   MAXABROW: the lowest-numbered row containing the element with largest absolute value.

13   RNORM(A): the largest sum of the absolute values of the elements in each row of **A**.

2   RNORMROW: the lowest-numbered row with the largest sum of absolute values.

15   SUM(A): the sum of the elements in **A**.

3   UBND(A,2): the upper bound of the second subscript of **A**.

Notes

Maintenance, Service, and Warranty

Maintenance

The Matrix ROM does not require maintenance. However, there are several areas of caution that you should be aware of. They are:

WARNING: Do not place fingers, tools, or other foreign objects into the plug-in ports. Such actions may result in minor electrical shock hazard and interference with some pacemaker devices. Damage to plug-in port contacts and the computer's internal circuitry may also result.

CAUTION: Always switch off the computer and any peripherals involved when inserting or removing modules. Use only plug-in modules designed by Hewlett-Packard specifically for the HP-83/85. Failure to do so could damage the module, the computer, or the peripherals.

CAUTION: If a module or ROM drawer jams when inserted into a port, it may be upside down or designed for another port. Attempting to force it may damage the computer or the module. Remove the module carefully and reinsert it.

CAUTION: Do not touch the spring-finger connectors in the ROM drawer with your fingers or other foreign objects. Static discharge could damage electrical components.

CAUTION: Handle the plug-in ROMs very carefully while they are out of the ROM drawer. Do not insert any objects in the contact holes on the ROM. Always keep the protective cap in place over the ROM contacts while the ROM is not plugged into the ROM drawer. Failure to observe these cautions may result in damage to the ROM or ROM drawer.

For instructions on how to insert and remove the ROM and ROM drawer, please refer to the instructions accompanying the ROM drawer or to appendix B of the HP-83 or HP-85 owner's manual.

Service

If at any time you suspect that the Matrix ROM or the ROM drawer may be malfunctioning, do the following:

1. Turn the computer and all peripherals off. Disconnect all peripherals and remove the ROM drawer from the computer port. Turn the HP-83/85 back on. If the computer does not respond or displays `Error 23 : SELF TEST`, the HP-83/85 requires service.

2. Turn the HP-83/85 off. Install the ROM drawer, with the Matrix ROM installed, into any port. Turn the computer on again.

   ● If `Error 112 : MATRIX ROM` is displayed, indicating that the ROM is not operating properly, turn the HP-83/85 off and try the ROM in another ROM drawer slot. This will help you determine if particular slots in the ROM drawer are malfunctioning, or if the ROM itself is malfunctioning.

   ● If the cursor does not appear, the system is not operating properly. To help determine what is causing the improper operation, repeat step 2 with the ROM drawer installed in a different port, both with the Matrix ROM installed in the ROM drawer and with the Matrix ROM removed from the ROM drawer.

3. Refer to How to Obtain Repair Service for information on how to obtain repair service for the malfunctioning device.

Warranty Information

The complete warranty statement is included in the information packet shipped with your ROM. Additional copies may be obtained from any authorized HP-83/85 dealer, or the HP sales and service office where you purchased your system.

If you have questions concerning the warranty, and you are unable to contact the authorized HP-83/85 dealer or the HP sales and service office where you purchased your computer, please contact:

In the U.S.:

Hewlett-Packard
Corvallis Division Customer Support
1000 N.E. Circld Blvd.
Corvallis, OR 97330
Tel. (503) 758-1010
Toll Free Number: (800) 547-3400 (except
    in Oregon, Hawaii and Alaska).

In Europe:

Hewlett-Packard S.A.
7, rue du Bois-du-lan
P. O. Box
CH-1217 Meyrin 2
Geneva
Switzerland

Other Countries:

> Hewlett-Packard Intercontinental
> 3495 Deer Creek Rd.
> Palo Alto, California 94304
> U.S.A.
> Tel. (415) 857-1501

How to Obtain Repair Service

Not all Hewlett-Packard facilities offer service for the HP-83/85 and its peripherals. For information on service in your area, contact your nearest authorized HP dealer or the nearest Hewlett-Packard sales and service office.

If your system malfunctions and repair is required, you can help assure efficient service by providing the following items with your unit(s):

1. A description of the configuration of the HP-83/85, exactly as it was at the time of malfunction, including any plug-in modules, tape cartridges, or other accessories.

2. A brief description of the malfunction symptoms for service personnel.

3. Printouts or any other materials that illustrate the problem area.

4. A copy of the sales slip or other proof of purchase to establish the warranty coverage period.

Computer and peripheral design and circuitry are proprietary to Hewlett-Packard, and service manuals are not available to customers.

Serial Number

Each HP-83/85 and peripheral carries an individual serial number. It is recommended that you keep a separate record of this number. Should your unit be stolen or lost, the serial number is often necessary for tracing and recovery, as well as for any insurance claims. Hewlett-Packard does not maintain records of individual owner's names and unit serial numbers.

General Shipping Instructions

Should you ever need to ship any portion of your HP-83/85 system, be sure that it is packed in a protective package (use the original shipping case), to avoid in-transit damage. Hewlett-Packard suggests that the customer always insure shipments.

If you happen to be outside of the country where you bought your computer or peripheral, contact the nearest authorized HP-83/85 dealer or the local Hewlett-Packard office. All customs and duties are your responsibility.

Syntax Summary

Syntax Guidelines

DOT MATRIX	Items shown in dot matrix type must be entered exactly as shown (in either uppercase or lowercase letters). If several items are shown stacked, one (but only one) of the items must be specified.
[ ]	Items shown between brackets are optional. If several items are stacked between brackets, you can specify any one or none of the items.
...	Three dots (ellipsis) following a set of brackets indicate that the items between the brackets may be repeated.
italics	Items shown in italic type are numeric expressions or names of arrays that should be specified in the statement or function.

MAT *result array* = (*numeric expression*)   **Page 14**

MAT *result array* = *operand array*   **Page 39**

MAT *result array* [⟨ [*row number* or *first row number* : *last row number*] , [*column number* or *first column number* : *last column number*] ⟩] = *operand array* [⟨ [*row number* or *first row number* : *last row number*] , [*column number* or *first column number* : *last column number*] ⟩]   **Page 40**

MAT *result array* = − *operand array*   **Page 22**

MAT *result array* = *operand array 1* $\begin{matrix}+\\-\\.\\/\end{matrix}$ *operand array 2*   **Page 23**

MAT *result array* = *operand array 1* ✳ *operand array 2*   **Page 25**

MAT *result array* = (*scalar*) $\begin{matrix}+\\-\\✳\\/\end{matrix}$ *operand array*   **Page 22**

MAT *result array* = (*scalar*) ✳ *operand array 1* + (*scalar*) ✳ *operand array 2*   **Page 24**

MAT *result array* = CON [(*redim subscripts*)]   **Page 13**

MAT *result vector* = CROSS (*operand vector 1* , *operand vector 2*)   **Page 28**

MAT *result array* = CSUM (*operand array*)   **Page 35**

MAT *result matrix* = IDN [(*redim subscripts*)]   **Page 14**

MAT *result matrix* = INV (*operand matrix*)   **Page 30**

MAT *result array* = INV (*operand array 1*) ✳ *operand array 2*   **Page 31**

MAT *result array* = RSUM (*operand array*)   **Page 35**

MAT *result array* = SYS (*coefficient matrix* , *constant array*)   **Page 32**

MAT *result matrix* = ZER [(*redim subscripts*)]   **Page 13**

MAT *result array* = TRN (*operand array*)   **Page 21**

MAT *result array* = TRN (*operand array 1*) ✳ *operand array 2*   **Page 27**

MAT *result array* = *operand array 1* * TRN (*operand array 2*)   **Page 27**

MAT DISP [$\begin{matrix} ROW \\ COL \end{matrix}$] *array* [$\begin{matrix} ; \\ , \end{matrix}$ [$\begin{matrix} ROW \\ COL \end{matrix}$] *array*] ... [$\begin{matrix} ; \\ , \end{matrix}$]   **Page 17**

MAT DISP USING $\begin{matrix} \textit{format string} \\ \text{or} \\ \textit{statement number} \end{matrix}$ [; [$\begin{matrix} ROW \\ COL \end{matrix}$] *array* [, [$\begin{matrix} ROW \\ COL \end{matrix}$] *array*] ...]   **Page 18**

MAT INPUT *array* [, *array*] ...   **Page 11**

MAT PRINT [$\begin{matrix} ROW \\ COL \end{matrix}$] *array* [$\begin{matrix} ; \\ , \end{matrix}$ [$\begin{matrix} ROW \\ COL \end{matrix}$] *array*] ... [$\begin{matrix} ; \\ , \end{matrix}$]   **Page 17**

MAT PRINT USING $\begin{matrix} \textit{format string} \\ \text{or} \\ \textit{statement number} \end{matrix}$ [; [$\begin{matrix} ROW \\ COL \end{matrix}$] *array* [, [$\begin{matrix} ROW \\ COL \end{matrix}$] *array*] ...]   **Page 18**

MAT READ *array* [, *array*] ...   **Page 12**

MAXAB (*array*)   **Page 49**

MAXABCOL   **Page 49**

MAXABROW   **Page 49**

REDIM *array* (*redim subscripts*) [, *array* (*redim subscripts*)] ...   **Page 7**

RNORM (*array*)   **Page 50**

RNORMROW   **Page 50**

SUM (*array*)   **Page 50**

UBND (*array* , *expression*)   **Page 50**

Notes

Error Messages

A complete list of all HP-83/85 errors appear in appendix E of the HP-83 and HP-85 owner's manuals. In addition to those, there are seven error messages that may be generated by the Matrix ROM.

Error Message	Error Condition
109 : # DIMS	Incorrect number of dimensions.
110 : NOT A 3-VECTOR	Vector specified does not have three elements.
111 : DIM MISMATCH	Incorrect number of elements.
112 : MATRIX ROM	Matrix ROM requires service.
113 : DIM SIZE	• Total number of elements specified when redimensioning exceeds the number originally dimensioned. • Attempt to create empty array with OPTION BASE 0 in effect. (Refer to page 46.) • Statement specifies result array created with OPTION BASE 0 in effect and *empty* operand array created with OPTION BASE 1 in effect.
114 : NOT SQUARE	Array specified is not square. (The number of rows is not the same as the number of columns.)
115 : NON-VECTOR	Array specified is not a vector.

The Matrix ROM provides a special function, ERROM, that returns a number designating the last plug-in ROM to generate an error message. This is useful when you have more than one ROM plugged in, since certain ROMs generate the same error number or error message. The number designating the Matrix ROM is 176. If the error originated in the HP-83/85 itself (rather than in a plug-in ROM), or if no error has occurred, ERROM returns the value 0.

hp HEWLETT
PACKARD

1000 N.E. Circle Blvd., Corvallis, OR 97330