TEKTRONIX®

4051R01

MATRIX FUNCTIONS

INSTRUCTION MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

070-2127-00

First Printing JUNE 1976

TABLE OF CONTENTS

SECTION 1 GENERAL DESCRIPTION Page

Introduction 1-1

Specifications 1-2

Installation Instructions 1-3

SECTION 2 MATRIX FUNCTIONS

Function Summary 2-1

The TRN Function 2-3

What the TRN Function Does 2-3

How to Use the TRN Function 2-4

Dimensioning the Parameter Variable 2-4

Dimensioning the Target Variable 2-4

A Sample Program 2-5

Naming the Target Variable 2-6

Things to Watch Out For When Using the TRN Function ... 2-7

The IDN Function 2-8

What the IDN Function Does 2-8

Creating an Identity Matrix 2-8

The IDN Function and Non-square Matrices 2-9

How to Use the IDN Function 2-10

Dimensioning the Target Variable 2-10

Naming the Target Variable 2-10

Sample Programs 2-10

Syntax Forms 2-11

Things to Watch Out For When Using the IDN Function ... 2-12

Additional Comments: Properties of Identity Matrices 2-13

TABLE OF CONTENTS (cont)

SECTION 2 MATRIX FUNCTIONS (cont) Page

The MPY Function 2-15

What the MPY Function Does 2-15

Performing Matrix Multiplication 2-15

When the MPY Function Can Be Performed 2-16

The Dimensions of the Result 2-18

How to Use the MPY Function 2-19

Dimensioning the Parameter Variables 2-19

Dimensioning the Target Variable 2-19

A Sample Program 2-20

Naming the Target Variable 2-20

Things to Watch Out For When Using the MPY Function ... 2-21

Additional Comments About Matrix Multiplication 2-22

The INV Function 2-26

What the INV Function Does 2-26

The Square Part 2-26

The Extra Columns 2-27

The INV Function and Square Matrices 2-30

How to Use the INV Function 2-31

Dimensioning the Parameter Variable 2-31

Dimensioning the Target Variable 2-31

Naming the Target Variable 2-31

A Sample Program 2-32

When the INV Function Can Be Performed 2-33

When the INV Function Cannot Be Performed 2-34

Things to Watch Out For When Using the INV Function 2-36

TABLE OF CONTENTS (cont)

SECTION 2 MATRIX FUNCTIONS (cont) Page

The DET Function 2-37

The Determinant of a Square Matrix 2-37

Computing the Determinant of a Square Matrix by Hand ... 2-37

What the DET Function Does 2-40

How to Use the DET Function 2-40

When to Perform the DET Function 2-40

Syntax Forms 2-41

A Sample Program 2-41

Assigning the Result to a Target Variable 2-42

Naming the Target Variable 2-42

Things to Watch Out For When Using the DET Function ... 2-42

APPENDIX A SUPPLEMENTARY INFORMATION

Mathematical Invertibility Versus Numerical Invertibility A-1

Truncation Error A-2

The Condition of a Matrix A-2

The Condition Number A-3

How to Use the Condition Number k to Evaluate the Result of the INV Function A-3

Evaluating the Result of the DET Function A-3

An Example A-4

APPENDIX B FUNCTION SUMMARY B-1

APPENDIX C ERROR MESSAGES C-1

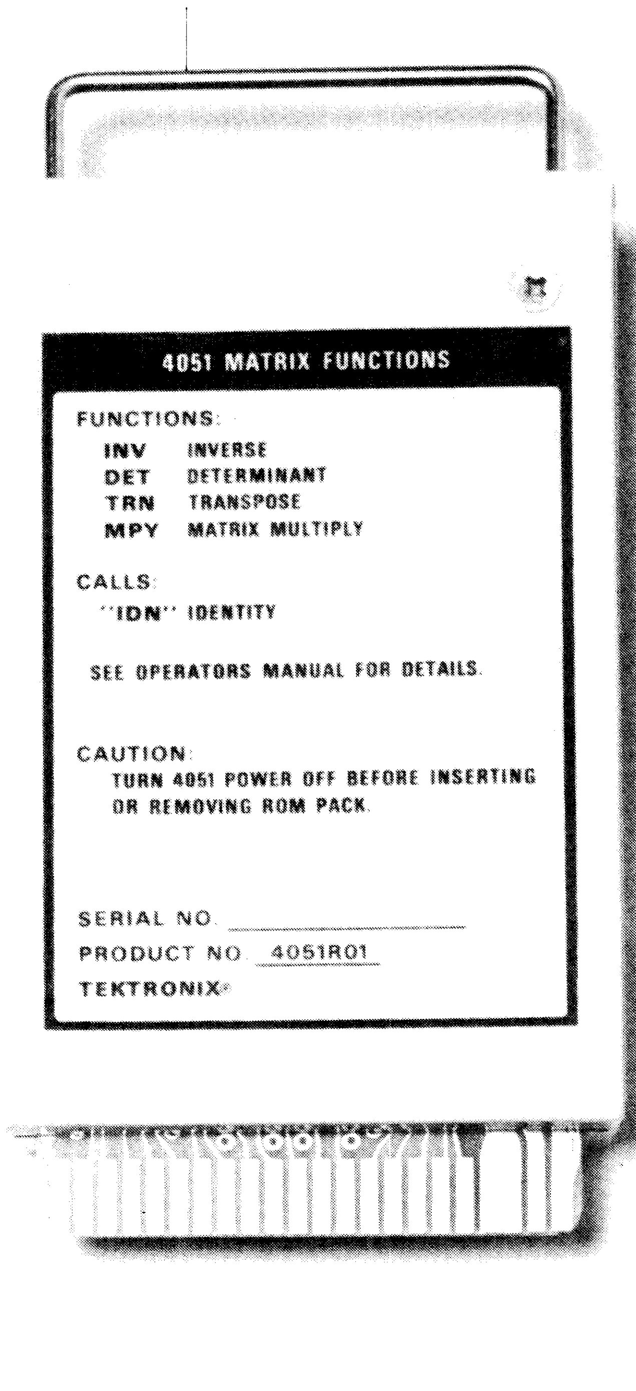


Fig. 1-1. 4051 MATRIX FUNCTIONS ROM Pack.

Section 1

GENERAL DESCRIPTION

INTRODUCTION

The Tektronix 4051R01 MATRIX FUNCTIONS are contained in a Read Only Memory device designed to be used with the Tektronix 4051 Graphic System. The MATRIX FUNCTIONS ROM Pack, shown in Fig. 1-1, is a small plastic case (ROM Pack) containing matrix firmware routines for the 4051 microprocessor. The firmware routines are special sets of instructions, which have been permanently stored on memory chips inside the ROM Pack. The MATRIX FUNCTIONS add to the capabilities of the Graphic System, without altering the operation of the system, or occupying any of the available RAM (Random Access Memory) space.

The MATRIX FUNCTIONS enable the Graphic System to perform certain functions which are essential to matrix algebra. Once the ROM Pack is inserted into a slot in the 4051 backpack or plugged into a ROM Expander Unit, the Graphic System is able to perform five matrix functions, DET, IDN, INV, MPY, and TRN. Briefly, these functions provide the determinant (DET), the identity matrix (IDN), the inverse and solutions to systems of linear equations (INV), the matrix product (MPY), and the transpose (TRN).

Using the MATRIX FUNCTIONS is more efficient and convenient than creating BASIC programs to perform matrix functions. BASIC programs used to perform the same functions "soak up" RAM (Random Access Memory) space. For instance, a BASIC program designed to perform the same function as INV takes up at least 60 lines of BASIC program, and ties up about 2K bytes of memory space. By contrast, the INV function provided by the MATRIX FUNCTIONS ROM Pack does not occupy RAM, and thus leaves more space available for storing BASIC programs and data.

Functions provided by the MATRIX FUNCTIONS ROM Pack are executed much faster than BASIC programs used to perform the same function. For example, the INV function inverts a 10X10 matrix approximately 8.5 times faster than the BASIC program which uses the same algorithm.

In summary, the MATRIX FUNCTIONS ROM Pack is a Read Only Memory device which allows the Graphic System to perform five matrix functions which would otherwise require BASIC program subroutines. Using the MATRIX FUNCTIONS to perform these functions saves programming time and execution time, and conserves memory space.

SPECIFICATIONS

POWER REQUIREMENTS

The 4051R01 MATRIX FUNCTIONS draws all necessary power from the 4051 power supplies. Connections to the power supplies are made through the backpack on the rear panel of the 4051 main chassis. The MATRIX FUNCTIONS device must be inserted into a slot in the backpack, or into a ROM Expander Unit, before power is applied to the system.

Voltage Supplies	Current Limit
+5 Vdc	278 mA
+12 Vdc	25 mA
−12 Vdc	20 mA

TEMPERATURE

Non-operating:	−40°C to +65°C
Operating:	+10°C to +40°C

ALTITUDE

Non-operating:	50,000 feet maximum
Operating:	15,000 feet maximum

HUMIDITY

95% non-condensing (storage)
80% non-condensing (operating)

VIBRATION (NON-OPERATING)

0.015" DA-10-50-10

SHOCK (NON-OPERATING)

1/2 Sine 11 ms duration, 30 G's

PHYSICAL DIMENSIONS (INCLUDING EDGE BOARD CONNECTOR)

Length: 4.662"
Width: 2.620"
Depth: 0.875"

WEIGHT

8 oz.

STANDARD ACCESSORIES

1 — Operators Manual P/N 070-2127-00

INSTALLATION INSTRUCTIONS

1. Flip the 4051 power switch to the OFF position.

CAUTION

Inserting any device into the backpack when the 4051 power is ON may cause memory to be erased. Make sure that important information in the RAM is stored on magnetic tape before turning the power OFF and proceeding.

2. With the power removed from the system, insert the MATRIX FUNCTIONS into a slot as shown in Fig. 1-2. Press down, and at the same time gently rock the plasic housing from side to side until the ROM Pack edgeboard connector is firmly seated in the receptacle connector.

3. Flip the 4051 power switch to the ON position. After a few minutes of warm-up, the system is ready for use.

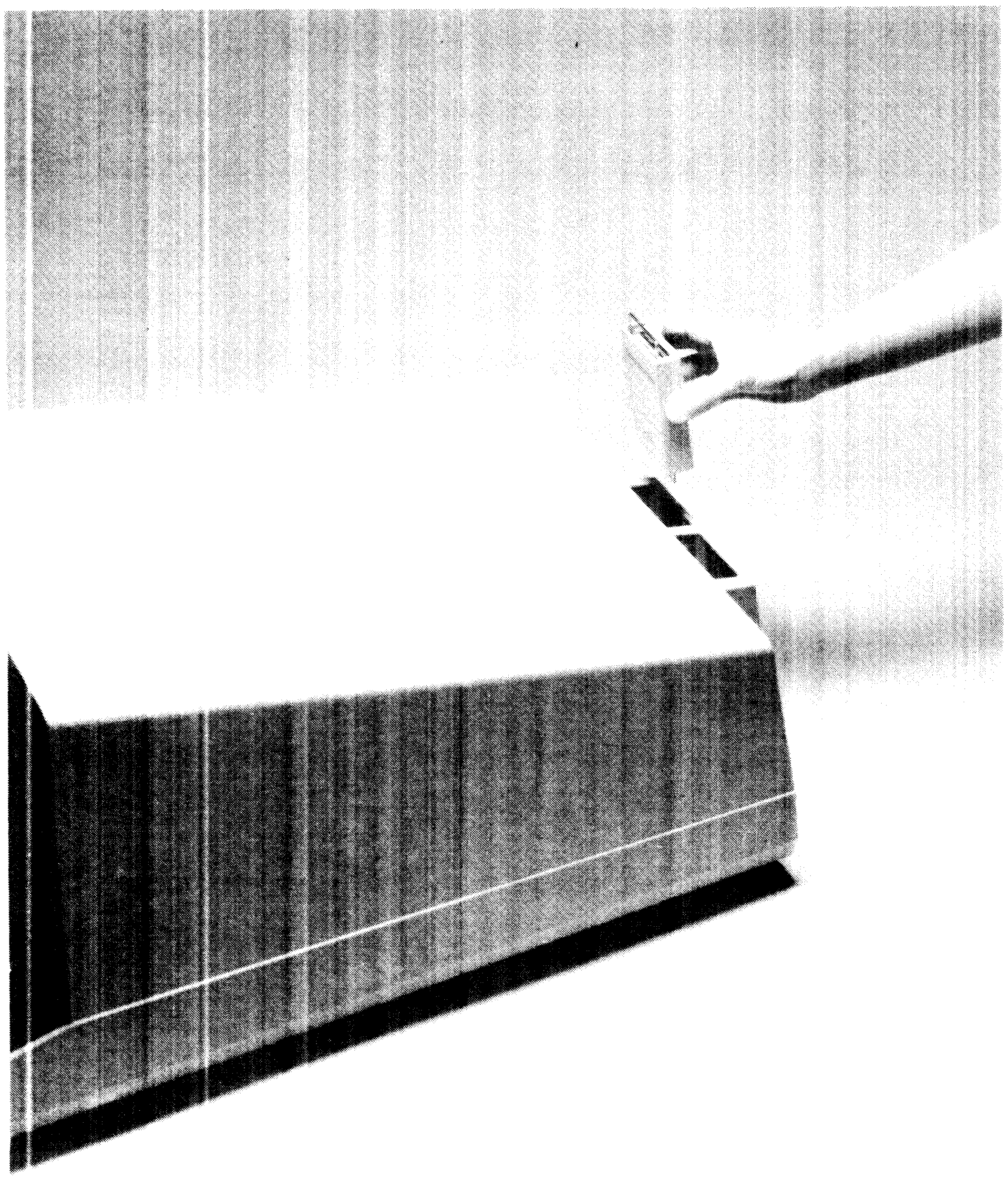


Fig. 1-2. Insert the **MATRIX FUNCTIONS ROM** Pack into a backpack slot.

Section 2

MATRIX FUNCTIONS

INTRODUCTION

This section contains a detailed explanation of each of the five functions provided by the MATRIX FUNCTIONS Rom Pack. The explanations are intended for someone who is generally familiar with the operation of the Tektronix 4051, but who is not necessarily an expert in matrix algebra. Background information about matrices is sometimes included, when it is helpful for understanding the matrix functions and how they can be used.

Examples and sample programs are made as simple as possible, and a brief list of pitfalls to watch out for is included at the end of each function description. This section begins with a "Function Summary" chart (Table 2-1), which may be used as a quick reference once you have learned how to work with the functions. The chart is also included as Appendix B, so you may remove it from the manual and use it as a separate sheet.

TABLE 2-1

FUNCTION SUMMARY

FUNCTION	PAGE NO.	EXAMPLE	EXPLANATION	ALGORITHM
TRN	2-3	B=TRN(A)	Returns the transpose of a matrix. Each row (or column) in A becomes the corresponding row column) in B=TRN(A). If A is an IXJ matrix, B=TRN(A) is a JXI matrix.	B(I,J) = A(J,I)
IDN	2-8	CALL "IDN", I or A$ = "IDN" CALL A$,I	Creates a matrix whose diagonal elements I(1,1), I(2,2), I(3,3),... are 1's, and all other elements are 0's.	I(I,J) = 1 if I = J I(I,J) = 0 if I ≠ J

TABLE 2-1 (cont)

FUNCTION	PAGE NO.	EXAMPLE	EXPLANATION	ALGORITHM
MPY	2-15	C = A MPY B	Returns the matrix product of two arrays. If A is an IXJ matrix, and B is a JXK matrix, the product C is an IXK matrix.	$C(I,K) = \sum_{L=1}^{K} A(I,L) * B(L,K)$
INV	2-26	B=INV(A)	Returns the inverse of the square part of the parameter matrix, and solves systems of equations represented by the matrix.	The Gauss-Jordan technique is performed in place, with full row and column pivoting.
DET	2-37	B=INV(A) followed by DET or X=DET	Returns the determinant of the square part of the matrix most recently supplied to the INV function.	The value of the determinant is the product of the pivot elements computed during the INV process.

The TRN Function

Syntax Form:

[Line number] array variable = TRN array variable

Descriptive Form:

[Line number] target variable = TRN parameter variable

PURPOSE

The TRN function returns the transpose of a matrix.

EXPLANATION

WHAT THE TRN FUNCTION DOES

When the TRN function is performed on a matrix, the result is a new matrix found by making the columns into rows, or the rows into columns. For instance, when B=TRN(A) is performed, each row in A becomes the corresponding column in B. This is equivalent to saying that each **column** in A becomes the corresponding **row** in B. For example:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 8 & -2 & 0 \end{bmatrix} \quad , B = \text{TRN (A)} = \begin{bmatrix} 1 & 8 \\ 3 & -2 \\ 5 & 0 \end{bmatrix}$$

In this example, A is a 2X3 matrix. When the TRN function is performed on A, the result is the 3X2 matrix B shown above.

Notice that the first row of A is the same as the first column of B, and the second row of A is the same as the second column of B. Likewise, the first **column** of A is the same as the first **row** of B, the second column of A is the same as the second row of B, and so on. Matrix B is the transpose of matrix A.

When the TRN function is performed, the number of rows and columns are reversed. In the above example, A is a 2X3 matrix, so B=TRN(A) is a 3X2 matrix. In general terms, when A is a matrix having K rows and N columns, B=TRN(A) is a matrix having N rows and K columns.

If A is a matrix consisting of one row, the TRN function returns a matrix consisting of one column, and vice versa. For instance:

$$A = \begin{bmatrix} 6 & 1 & 8 \end{bmatrix} \quad , \ B = \text{TRN (A)} = \begin{bmatrix} 6 \\ 1 \\ 8 \end{bmatrix}$$

In this example, A is a 1X3 matrix, that is, a row containing three elements. B is the transpose of A, and is therefore a 3X1 matrix, a column containing three elements.

HOW TO USE THE TRN FUNCTION

Dimensioning the Parameter Variable

If a matrix is to be used as a parameter for the TRN function, it must be dimensioned in a DIM statement, using two subscripts to indicate the number of rows and columns in the matrix. For example, if a 5X6 matrix A is to be supplied to the TRN function, A must be dimensioned in the following statement:

```
DIM A(5,6)
```

Using only one subscript to dimension the parameter variable causes an error to occur when the TRN function is performed. For instance, when the parameter for the TRN function is a matrix consisting of one row of four elements, the matrix should not be dimensioned in a statement such as DIM A(4). The correct way to dimension the matrix is shown below:

```
DIM A(1,4)
```

Dimensioning the Target Variable

When the TRN function is performed, the result is a new matrix, which must be assigned to an array variable, called the target variable because it serves as a "target" for the new matrix. The target variable must be dimensioned in a DIM statement, using two subscripts to indicate the number of rows and columns. The subscripts used to dimension the target variable must be the reverse of the subscripts used to dimension the parameter variable. For instance:

```
DIM A(5,6),B(6,5)
```

This statement dimensions a 5X6 matrix A which may be used as a parameter for the TRN function. Matrix B may be used as the target variable, because it is dimensioned to be a 6X5 matrix.

A Sample Program

The following program illustrates how the TRN function may be used to find the transpose of a matrix:

```
100 DELETE A,B
110 DIM A(2,3),B(3,2)
120 READ A
130 DATA 1,3,5,8,−2,0
140 B = TRN (A)
150 PRINT B
```

This program computes the transpose of a 2X3 matrix A, and assigns the result to target variable B. The first line of the program deletes variables A and B from memory. This is a precautionary measure to make sure that the variables are in an undefined state before line 110 is executed. (Previously defined numeric variables cannot be dimensioned as array variables without first deleting them from memory.)

Line 110 dimensions the variable A to be a 2X3 matrix. Variable B is to be used as the target variable, and is dimensioned in the same statement to be a 3X2 matrix.

When line 120 is executed, the numbers contained in line 130 are assigned to matrix A in row major order. The result is shown below:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 8 & -2 & 0 \end{bmatrix}$$

In line 140, TRN(A) is performed and the result is assigned to the target variable. Finally, line 150 causes the result to appear on the display. The result is shown below:

$$B = \text{TRN (A)} = \begin{bmatrix} 1 & 8 \\ 3 & -2 \\ 5 & 0 \end{bmatrix}$$

Naming the Target Variable

The transpose of a matrix A does not have the same shape as matrix A, unless A is square (has the same number of rows as columns). For this reason, caution must be used when giving the same name to the target variable as to the parameter variable. For example:

```
100 DELETE A
110 DIM A(5,2)
120 READ A
130 DATA 1,2,6,−4,3,0,8,1,−2
140 A = TRN (A)
150 PRINT A
```

This program causes a SHAPE ERROR message to appear on the display. The SHAPE ERROR occurs when line 140 is executed, because the statement 140 A=TRN(A) uses the same variable name for the target variable and the parameter of the TRN function. When the statement is executed, an attempt is made to assign the transpose of A, a 2X5 matrix, to array variable A, which is dimensioned in line 110 to be a 5X2 matrix. A SHAPE ERROR results.

When matrix A is square (has the same number of rows as columns), the transpose of A **does** have the same shape as A. In this case, the same variable name can be used for the target variable as for the parameter of the TRN function. For example:

```
100 DELETE A
110 DIM A(3,3)
120 READ A
130 DATA 1,2,6,−4,3,0,8,1,−2
140 A = TRN (A)
150 PRINT A
```

This program is the same as the previous one, but lines 100 and 120 have been modified to make A a square 3X3 matrix. This time, line 140 does not cause a SHAPE ERROR, because A and the transpose of A are conformable (they are of the same shape and have the same number of elements.) When the statement A=TRN(A) is executed, the transpose of A is computed and assigned to array variable A, replacing the original matrix A. The statement 140 PRINT A; causes the result to appear on the display as indicated below:

$$A = \begin{bmatrix} 1 & -4 & 8 \\ 2 & 3 & 1 \\ 6 & 0 & -2 \end{bmatrix}$$

The statement A=TRN(A) causes the original matrix A to be replaced by TRN(A). Care must be taken not to "overwrite" the original matrix A in this way, if A is needed in later parts of the program. In general, it is safer to use different variable names for the target variable and the parameter variable; however, if a matrix has been overwritten by its transpose, the original matrix can be recovered by performing the TRN function again.

THINGS TO WATCH OUT FOR WHEN USING THE TRN FUNCTION

Undefined Variable	Not assigning a numeric value to one or more of the elements of the parameter matrix, causes an UNDEFINED VARIABLE error message to be printed on the display when the TRN function is performed.
Syntax Error	Not assigning the result of the TRN function to a target variable is a SYNTAX ERROR. For example, 100 TRN A is not a valid statement.
Invalid Function Argument	Forgetting to dimension the target array variable causes an INVALID FUNCTION ARGUMENT message to appear on the display.
Shape Error	Incorrectly dimensioning the target variable results in a SHAPE ERROR when the TRN function is performed. If the parameter for the TRN function is dimensioned to be a KXN matrix, the target variable must be dimensioned to be a NXK matrix. When the parameter for the TRN function has a non-square shape, using the same name for the parameter variable as for the target variable, causes a SHAPE ERROR. (Refer to "Naming the Target Variable".)

The IDN Function

Syntax Form:

$$[\text{ Line number }]\ \text{CALL}\begin{Bmatrix}\text{"IDN"}\\ \text{string variable}\end{Bmatrix}\text{, array variable}$$

Descriptive Form:

[Line number] CALL call name , target variable

PURPOSE

The IDN function creates a matrix whose elements are 1's along the diagonal, and 0's elsewhere.

EXPLANATION

WHAT THE IDN FUNCTION DOES

Creating an Identity Matrix

The IDN function generates a matrix whose elements are 1's along the diagonal, and 0's elsewhere. For example, the IDN function may be used to generate the following 3X3 matrix:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

I(1,1), I(2,2), and I(3,3) are the diagonal elements of matrix I, and have been assigned the value 1. All other elements are 0's.

When the IDN function is used to generate a square matrix as in the above example, the result is called an identity matrix. In this example, matrix I is the 3X3 identity matrix.

The result of the IDN function is assigned to an array variable, called the target variable. The result of the IDN function always has the same dimensions as the target variable. For instance, if I is to be the target for the result of the IDN function, and is dimensioned in a DIM statement to be a 5X5 matrix, performing the IDN function results in the 5X5 identity matrix shown below:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Identity matrices have special properties which are discussed in "Additional Comments: Properties of Identity Matrices" on the following pages.

The IDN Function and Non-square Matrices

A matrix does not have to have a square shape to be used as the target for the IDN function. The IDN function assigns values to the elements of a non-square array I, just as it did to the square matrices in the preceding examples: diagonal elements (elements I(1,1),I(2,2),....,I(K,K)) are assigned the value 1, and all other elements are assigned the value 0.

For example, when I is dimensioned to be a 3X5 array, performing the IDN function results in the following matrix:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Note that the diagonal elements I(1,1), I(2,2), and I(3,3), are all 1's, and the rest of the elements are 0's. Also, the last two columns of the matrix have 0's assigned to every element.

When the target variable is dimensioned to be a non-square matrix, the IDN function produces a matrix having at least one row or column filled with 0's. The rows or columns of 0's are the ones which lie outside the square part of the matrix. (The square part of a matrix is the largest square portion of the matrix which includes the upper-left corner.) For example, let matrix I be dimensioned as a 6X3 array:

```
110 DIM I(6,3)
```

When the IDN function is performed, matrix I is assigned the following values:

$$I = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right]$$

The shaded area shown above is the square part of matrix I because it is the largest square that can be blocked off, starting in the upper-left corner. Notice that the square part is the 3X3 identity matrix, and all elements outside the square part are 0's.

When the target variable has a non-square shape, the result of the IDN function resembles an identity matrix (the square part is an identity matrix). But because of the extra rows or columns of 0's, the non-square matrix does not have the properties of an identity matrix. For an explanation of the special properties of identity matrices, refer to the topic "Additional Comments: Properties of Identity Matrices" on the following pages.

HOW TO USE THE IDN FUNCTION

Dimensioning the Target Variable

Before performing the IDN function, a target variable must be dimensioned in a DIM statement, using two subscripts to indicate the number of rows and columns in the result matrix. For instance, if variable A is to receive the result of the DIM function, and the function is being used to generate the 3X3 identity matrix, A must be dimensioned as follows:

```
DIM A(3,3)
```

Using only one subscript to dimension the target variable causes an error to occur when the IDN function is performed.

Naming the Target Variable

Any valid numeric variable name may be used as the target for the result of the IDN function. An array can serve as the target for the IDN function without having numeric values assigned to each element. The only requirement for the target variable is that it be previously dimensioned in a DIM statement, using two subscripts to indicate the number of rows and columns in the result matrix.

An array variable name which has numeric values assigned to each element may also be used as the target variable. However, the result of the IDN function replaces all previously assigned values.

Sample Programs

The following program illustrates how the IDN function is used to create the 4X4 identity matrix:

```
100 DELETE I
110 DIM I(4,4)
120 CALL "IDN", I
130 PRINT "I = ";I
```

Only four statements are needed to form the identity matrix and display the result. In line 110, the variable I is dimensioned to be a square matrix having four rows and four columns. Matrix I is to be used as the target for the result of the IDN function, and must be previously dimensioned in a DIM statement, or an error occurs. When the IDN function is performed in line 120, matrix I becomes the 4X4 identity matrix; 1's are assigned to the diagonal elements of I, and 0's are assigned to all other elements. When line 130 is executed, the result appears on the display as follows:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix I is now the 4X4 identity matrix.

The dimensions of matrix I may be changed in order to make an identity matrix of a different size. For example, the following program creates the 6X6 identity matrix:

```
100 DELETE I
110 DIM I(6,6)
120 CALL "IDN",I
130 PRINT "I=";I
```

Line 110 contains the only difference between this program and the program that was used to make the 4X4 identity matrix. The DIM statement in line 110 determines which identity matrix is generated by the program. Here, I is dimensioned to be a 6X6 array, so the program makes the 6X6 identity matrix shown below:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Syntax Forms

In the sample programs discussed above, the CALL statement takes the following form:

```
120 CALL "IDN",I
```

The call name IDN in this statement may be replaced by a string variable, provided the string variable has been assigned the value "IDN". For example:

```
100 DELETE I
110 DIM I(6,6)
120 A$ = "IDN"
130 CALL A$,I
140 PRINT "I=";I
```

This program creates the 6X6 identity matrix, just as the last sample program did. This time, however, a string variable name appears in the CALL statement. The string variable A$ is assigned the value "IDN" in line 120, before appearing as the call name in line 130. The statements...

```
120 A$ = "IDN"
130 CALL A$,I
```

are equivalent to the statement...

```
120 CALL "IDN",I
```

NOTE

The important thing to remember about using a string variable in the CALL statement, is that the value "IDN" must be assigned to the string variable, or a CALL NAME INVALID error results when the IDN function is performed.

THINGS TO WATCH OUT FOR WHEN USING THE IDN FUNCTION

Syntax Error	Not inserting a comma between "IDN" and the array variable in the CALL statement, causes a SYNTAX ERROR. Likewise, leaving out the quotation marks around the call name IDN in the CALL statement, causes a SYNTAX ERROR message to appear on the display.
Call Name Invalid	Attempting to execute the statement CALL A$,I results in a CALL NAME INVALID error message, unless the value "IDN" has previously been assigned to A$. (See "Syntax Forms".)

Invalid Command Argument

Using a semicolon (;) between "IDN" and the target variable name, causes an INVALID COMMAND ARGUMENT error message to be printed on the display. A comma must appear in the CALL statement, not a semicolon.

Forgetting to dimension the target variable also causes an INVALID COMMAND ARGUMENT error to occur when the IDN function is performed.

Using only one subscript when dimensioning the target variable also causes an INVALID COMMAND ARGUMENT error to occur when the IDN function is performed. Two subscripts must be used when dimensioning a matrix which consists of a single row or column. For example:

```
100 DIM A(1,10)
110 CALL "IDN",I
```

Here, matrix I is dimensioned correctly using two subscripts (1 and 10).

ADDITIONAL COMMENTS: PROPERTIES OF IDENTITY MATRICES

Multiplying by an Identity Matrix

When a square matrix is multiplied by the identity matrix of the same size, the result is the original matrix. That is, the following is true for any square matrix A:

A MPY I = I MPY A = A

In this expression, MPY is matrix multiplication and I is the identity matrix having the same dimensions as A.

For example, let A be a 2X2 matrix and I be the 2X2 identity matrix as shown below:

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix} \quad , I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Multiplying A by I (or I by A) results in the same matrix A:

$$\text{A MPY I} = \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix} = A$$

$$\text{I MPY A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix} = \text{A}$$

(Refer to MPY for how to perform these calculations.)

The above rule only holds when A is square and I is the identity matrix of the same size as A. If I is a non-square matrix generated by performing the IDN function on a non-square target variable I, then it is not true that A MPY I = I MPY A = A. Instead, the product of A and I (or of I and A) may return a matrix which resembles A, but has chopped off part of A, or added rows or columns of 0's to A.

When an Identity Matrix is the Result of Performing Matrix Multiplication

One matrix is the inverse of another matrix if their product results in an identity matrix. In other words, matrix B is the inverse of matrix A if the following is true:

A MPY B = B MPY A = I

In this expression, MPY is matrix multiplication and I is the identity matrix having the same dimensions as A and B. A and B must both be square matrices.

As an example, let A and B be the 2X2 matrices shown below:

$$\text{A} = \begin{bmatrix} 4 & 3 \\ 3 & 2 \end{bmatrix} \quad , \text{B} = \begin{bmatrix} -2 & 3 \\ 3 & -4 \end{bmatrix}$$

The products A MPY B and B MPY A both result in the 2X2 identity matrix. This can be verified by performing the MPY function on A and B:

$$\text{C} = \text{A MPY B} = \begin{bmatrix} 4 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} -2 & 3 \\ 3 & -4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\text{D} = \text{B MPY A} = \begin{bmatrix} -2 & 3 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

(Refer to MPY for an explanation of how to compute these products.) Since both A MPY B and B MPY A result in the 2X2 identity matrix, A is the inverse of B.

The MPY Function

Syntax Form:

[Line number] array variable = array variable MPY array variable

Descriptive Form:

[Line number] target variable = parameter variable MPY parameter variable

PURPOSE

The MPY function returns the matrix product of two arrays.

EXPLANATION

WHAT THE MPY FUNCTION DOES

Performing Matrix Multiplication

When the MPY function is performed, the result is a new matrix. For example:

$$A = \begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix}$$

$$C = A \text{ MPY } B = \begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix} \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 16 \\ -9 & -12 \end{bmatrix}$$

The elements of the new matrix C are found by multiplying each element of the Ith **row** of the **first** matrix by the corresponding element of the Jth **column** of the **second** matrix, then adding the values. The sum is the I,Jth element of the product matrix.

For instance, to do this matrix mulitiplication by hand, start with I=1 and J=1 and multiply the elements of the first row of A by the corresponding elements of the first column of B, then add:

$$1*2 + 2*3 = 8$$

This is the element found in the first row, first column of C:

$$C(1,1) = 8$$

Next, make I=1 and J=2. Multiply each element in the first row of A by the corresponding element in the second column of B, and add:

1*8 + 2*4 = 16

This is the value assigned to the first row, second column of C:

C(1,2) = 16

Now, let I=2 and J=1. Using the second row of A and the first column of B,

C(2,1) = 0*2 + (−3)*3 = −9

And finally, I=2 and J=2. Using the second row of A and the second column of B,

C(2,2) = 0*8 + (−3)*4 = −12

These calculations are summarized in the diagram below. At each step, the shaded row of the first matrix and the shaded column of the second matrix are the items used to compute the shaded element of the product matrix:

STEP 1: $\begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix} \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 8 & \\ & \end{bmatrix}$, $C(1,1) = \begin{matrix} 1 & 2 \end{matrix} \quad \begin{matrix} 2 \\ 3 \end{matrix} = 1*2 + 2*3 = 8$

STEP 2: $\begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix} \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 16 \\ & \end{bmatrix}$, $C(1,2) = \begin{matrix} 1 & 2 \end{matrix} \quad \begin{matrix} 8 \\ 4 \end{matrix} = 1*8 + 2*4 = 16$

STEP 3: $\begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix} \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 16 \\ -9 & \end{bmatrix}$, $C(2,1) = \begin{matrix} 0 & -3 \end{matrix} \quad \begin{matrix} 2 \\ 3 \end{matrix} = 0*2 + (-3)*3 = -9$

STEP 4: $\begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix} \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 16 \\ -9 & -12 \end{bmatrix}$, $C(2,2) = \begin{matrix} 0 & -3 \end{matrix} \quad \begin{matrix} 8 \\ 4 \end{matrix} = 0*8 + (-3)*4 = -12$

When the MPY Function Can Be Performed

The MPY function can only be performed when the number of columns in the first matrix is equal to the number of rows in the second matrix. For instance, A MPY B can be performed for the following matrices:

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

$$\text{2X3} \qquad\qquad\qquad \text{3X3}$$

As indicated above, matrix A has three columns and matrix B has three rows. Since the number of columns in A is the same as the number of rows in B, the product C = A MPY B can be computed.

Performing E MPY F results in an error for the following matrices:

$$E = \begin{bmatrix} 1 & 0 & 3 & 0 \\ 0 & 1 & 0 & 0 \\ -2 & 0 & 2 & 0 \\ 0 & 4 & 0 & 1 \end{bmatrix} \quad , \quad F = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 2 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

$$\text{4X4} \qquad\qquad\qquad \text{3X3}$$

Here, matrix E has four columns and matrix F has three rows. The product E MPY F does not exist, and performing G = E MPY F results in an error, because the number of columns in E is not the same as the number of rows in F.

The fact that A may be multiplied by B does not necessarily mean that B may be multiplied by A. For instance, in the first example C = A MPY B may be performed, but D = B MPY A results in an error. The product B MPY A does not exist, because the number of columns in B is not equal to the number of rows in A:

$$B = \begin{bmatrix} 1 & -2 & 1 \\ 2 & 1 & -3 \\ 0 & 1 & 1 \end{bmatrix} \quad , \quad A = \begin{bmatrix} 1 & 0 & 2 \\ 2 & -1 & 1 \end{bmatrix}$$

$$\text{3X3} \qquad\qquad\qquad \text{2X3}$$

Since matrix B has three columns and matrix A has two rows, the product B MPY A does not exist.

As a final example, if A is a 2X6 matrix, C = A MPY B may be performed for any matrix B which has six rows. That is, B may be a 6X1 matrix, a 6X2 matrix, a 6X3 matrix, or a 6X4 matrix, and so on.

In summary, C = A MPY B can be performed whenever B has as many rows as A has columns.

The Dimensions of the Result

The result of A MPY B is a new matrix which has as many rows as A, and as many columns as B. For example:

$$A=\begin{bmatrix}1&0&0\\0&1&0\end{bmatrix},\quad B=\begin{bmatrix}3&1&-2\\0&-1&4\\2&0&0\end{bmatrix},\quad C=A\text{ MPY }B=\begin{bmatrix}3&1&-2\\0&-1&4\end{bmatrix}$$

$$\begin{tikzcd}\text{2X3} \arrow[r, dash, bend right] & \text{3X3} & \text{2X3} \arrow[loop below, dash]\end{tikzcd}$$

Since A has two rows and B has three columns, there are two rows and three columns in the result C.

Here is another example:

$$A=\begin{bmatrix}3&1\\-1&1\\7&0\end{bmatrix},\quad B=\begin{bmatrix}2&-1&0\\1&5&-2\end{bmatrix}$$

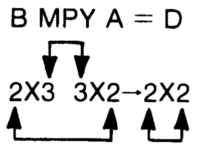
$$C=A\text{ MPY }B=\begin{bmatrix}3&1\\-1&1\\7&0\end{bmatrix}\begin{bmatrix}2&-1&0\\1&5&-2\end{bmatrix}=\begin{bmatrix}7&2&-2\\-1&6&-2\\14&-7&0\end{bmatrix}$$

$$D=B\text{ MPY }A=\begin{bmatrix}2&-1&0\\1&5&-2\end{bmatrix}\begin{bmatrix}3&1\\-1&1\\7&0\end{bmatrix}=\begin{bmatrix}7&1\\-16&6\end{bmatrix}$$

Since matrix A has two columns and matrix B has two rows, C = A MPY B can be computed. The result has as many rows as A and as many columns as B, which means that C is a 3X3 matrix:

$$\begin{tikzcd}[row sep=small] & \text{A MPY B = C} & \\ \text{3X2} \arrow[r, dash, bend left] \arrow[r, dash, bend right] & \text{2X3} \arrow[r, "\rightarrow"] & \text{3X3} \arrow[loop below, dash]\end{tikzcd}$$

Since B has three columns and A has three rows, the product D = B MPY A can also be computed. The result has as many rows as B and as many columns as A, which means D is a 2X2 matrix:



Notice that both A MPY B and B MPY A can be performed, but the resulting matrices are not the same, and are not even of the same size.

HOW TO USE THE MPY FUNCTION

Dimensioning the Parameter Variables

The MPY function returns the matrix product of two arrays. These parameter arrays must previously appear in a DIM statement, using two subscripts to dimension each variable. The dimensions of the parameters must be such that the arrays can be multiplied. That is, in order to perform A MPY B, the second subscript used to dimension A must be the same as the first subscript used to dimension B. For example:

```
100 DIM A(7,3),B(3,12)
```

Dimensioning the Target Variable

When the MPY function is performed, a new matrix is generated. The result must be assigned to a target variable. The target variable must be dimensioned in a DIM statement, using two subscripts to indicate the number of rows and columns in the resulting matrix. In order to perform C = A MPY B, the first subscript used to dimension C must be the same as the first subscript used to dimension A, and the second subscript used to dimension C must be the same as the second subscript used to dimension B. For example:

```
100 DIM A(7,3),B(3,12),C(7,12)
```

A Sample Program

The following program illustrates how the MPY function can be used to find the matrix product of two arrays:

```
100 DELETE A,B,C
110 DIM A(3,3),B(3,2),C(3,2)
120 READ A,B
130 DATA 1,2,0,4,1,2,−1,3,0,5,0,−6,1,2,0
140 C = A MPY B
150 PRINT "A=";A
160 PRINT "B=";B
170 PRINT "C = A MPY B=";C
```

This program performs matrix multiplication on two arrays A and B, and prints the result. The parameter matrices A and B are dimensioned in line 110. Since A is dimensioned to be a 3X3 matrix, and B is dimensioned to be a 3X2 matrix, the result of A MPY B is dimensioned in the same DIM statement to be a 3X2 matrix called C. When line 140 is executed, A MPY B is performed and the result is assigned to array variable C. The next three statements cause the results to appear on the display, as indicated below:

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 1 & 2 \\ -1 & 3 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 0 \\ -6 & 1 \\ 2 & 0 \end{bmatrix}, \quad C = A \text{ MPY } B = \begin{bmatrix} -7 & 2 \\ 18 & 1 \\ -23 & 3 \end{bmatrix}$$

Naming the Target Variable

The target for the result of the MPY function must not have the same variable name as either of the parameters. That is, the following types of statements are not valid:

```
A = A MPY B
B = A MPY B
```

Using the same variable name for the target array as for one of the parameters, results in an error.

THINGS TO WATCH OUT FOR WHEN USING THE MPY FUNCTION

Undefined Variable	Attempting to perform the MPY function without assigning a numeric value to every element of both parameters, causes an UNDEFINED VARIABLE error message to be printed on the display.
Syntax Error	Not assigning the result of the MPY function to a target variable, causes a SYNTAX ERROR: for example, the statement A MPY B is not a valid statement. Similarly, the statement A MPY B MPY C is not allowed: the result of A MPY B must be assigned to a target variable before attempting to multiply by C.
Invalid Function Argument	Using the same variable name for the target variable as for one of the parameters, causes an INVALID FUNCTION ARGUMENT error message to appear on the display. (Refer to the preceding topic "Naming the Target Variable".)
	Likewise, forgetting to dimension the target variable results in an INVALID FUNCTION ARGUMENT error when the MPY function is performed.
Shape Error	Attempting to perform the MPY function causes a SHAPE ERROR if the parameter matrices cannot be multiplied. Two matrices can only be multiplied if the number of columns in the first matrix is the same as the number of rows in the second matrix. (Refer to "When the MPY Function Can Be Performed".)
	Incorrectly dimensioning the target variable also causes a SHAPE ERROR to occur when the MPY function is performed. When performing C = A MPY B, the target variable C must be dimensioned to have as many rows as A and as many columns as B. (Refer to "Dimensioning the Target Variable".)
	Using only one subscript when dimensioning the parameters or the target variable, may cause a SHAPE ERROR when the MPY function is performed.

Size Error

Attempting to perform the MPY function causes a SIZE ERROR when one or more of the elements in the result is out of range (outside the range ±1.0E+308). After a SIZE ERROR message appears on the display, the result of performing the MPY function may be obtained by entering the target variable name and pressing RETURN. Elements in the result which are out of range (too large or too small) contain the largest or smallest number possible, ±8.988465774E+307.

If a SIZE ERROR occurs during execution of a BASIC program, the error may be handled without terminating the program by using the ON SIZE THEN ... statement. (Refer to the Graphic System Reference Manual for an explanation of ON...THEN... statements.)

ADDITIONAL COMMENTS ABOUT MATRIX MULTIPLICATION

A MPY B vs B MPY A

Matrix multiplication is not commutative, that is, A MPY B does not generally return the same result as B MPY A. For instance:

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

$$C = \text{A MPY B} = \begin{bmatrix} 0 & 1 \\ 2 & -5 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 8 & -4 \end{bmatrix}$$

$$D = \text{B MPY A} = \begin{bmatrix} 4 & 3 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 2 & -5 \end{bmatrix} = \begin{bmatrix} 6 & -11 \\ 4 & -10 \end{bmatrix}$$

Notice that matrices C = A MPY B and D = B MPY A are not the same. This is to be expected when performing matrix multiplication.

Since the results of A MPY B and B MPY A are usually different, care must be taken when determining which of the two products A MPY B or B MPY A is needed. Making the wrong choice may result in an incorrect answer, or even an error message: in many cases it is possible to perform A MPY B, but not B MPY A because of mismatched dimensions. (Refer to the topic "When the MPY Function Can Be Performed" for more information.)

The Zero Matrix

One unusual property of matrix multiplication concerns the matrix called 0 because it is filled entirely with 0's. In ordinary arithmetic, X*Y = 0 means that either X or Y or both are 0. But this is not the case in matrix algebra: A MPY B may be 0 without either A or B being 0. Here is an example:

$$A = \begin{bmatrix} 1 & -1 \\ 2 & -2 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 3 \\ 1 & 3 \end{bmatrix}, \quad A\ \mathrm{MPY}\ B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Here, neither A nor B is 0, yet the product A MPY B is 0. Keep in mind that this can happen when using the MPY function, and don't be alarmed when it does.

The Difference Between * and the MPY Function

There is more than one way to multiply matrices. One way uses the arithmetic operator * to form the element-by-element product of matrices A and B. The other way uses the MPY function to form the "matrix product" of A and B. The matrix product and the element-by-element product are computed in different ways, and yield different answers. For example:

$$A = \begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix},$$

$$C = A * B = \begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix} \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 16 \\ 0 & -12 \end{bmatrix}$$

$$D = A\ \mathrm{MPY}\ B = \begin{bmatrix} 1 & 2 \\ 0 & -3 \end{bmatrix} \begin{bmatrix} 2 & 8 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 16 \\ -9 & -12 \end{bmatrix}$$

Notice that A*B and A MPY B give different results. A*B is called the element-by-element product of A and B, because each element in the result is the product of the corresponding elements of A and B. That is, C = A*B is calculated as follows: C(1,1) = A(1,1)*B(1,1) = 1*2 = 2, C(1,2) = A(1,2)*B(1,2) = 2*8 = 16, and so on. However, D = A MPY B, the matrix product of A and B, is obtained in a different manner. (Refer to the topic "Performing Matrix Multiplication" for more information.)

Do not confuse the operation A*B with the matrix product provided by the MPY function. Matrix multiplication MPY has a special relationship to systems of linear equations, and has many other significant applications. You will find when working with matrices that the words "product" and "multiplication" often mean the **matrix** product and **matrix** multiplication, as provided by the MPY function. In such situations, using the operator *instead of the MPY function results in incorrect answers.

Raising Matrices to a Power

A matrix may be raised to a power in more than one way. One method uses the operator ↑ or * to raise each element of an array to a specified power. Another method finds the powers of a matrix by using the MPY function.

For example, A↑2 creates a matrix whose elements are the square of the corresponding elements in A. Likewise, A*A is the element-by-element product of A with itself, and causes each element of A to be squared in the same manner as A↑2. However, A MPY A squares A by computing the matrix product of A with itself, and gives a different result, as seen in the following example:

$$A = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \quad , \quad A*A=A\uparrow 2= \begin{bmatrix} 0 & 1 \\ 4 & 9 \end{bmatrix} \quad , \quad B=A \text{ MPY } A= \begin{bmatrix} 2 & 3 \\ 6 & 11 \end{bmatrix}$$

It is important not to confuse these methods of raising matrices to a power. Certain situations call for the "power" of a matrix to be computed by repeatedly performing the MPY function. In such cases, using the operator ↑ or * by mistake produces incorrect answers.

How Matrix Multiplication Relates to Systems of Linear Equations

A set of simultaneous equations can be represented by a matrix product. For example, consider the following equations:

$$\begin{aligned} 3x + 2y &= 5 \\ 4x + y &= 11 \end{aligned}$$

An equivalent way to state the equations uses a matrix product:

$$\begin{bmatrix} 3 & 2 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$$

The first matrix in the above expression is called the "coefficient matrix" because its elements are the coefficients of x and y in the original set of equations. The second matrix is a column consisting of the unknowns x and y, and the third matrix is a column consisting of the constants which appear to the right in the original set of equations.

If A is the coefficient matrix, X is the column of unknowns, and B is the column of constants, the above expression can be written more compactly as follows:

$$AX = B$$

AX is the matrix product of A and X. The expression AX=B is equivalent to the original set of equations. This fact may be verified by working out the matrix multiplication as follows:

$$AX = \begin{bmatrix} 3 & 2 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3x + 2y \\ 4x + y \end{bmatrix} = B$$

(Refer to the topic "Performing Matrix Multiplication" for how to compute the product by hand.)

But in this example, B is the column $\begin{bmatrix} 5 \\ 11 \end{bmatrix}$, therefore the equation AX = B says the following:

$$3x + 2y = 5$$

and

$$4x + y = 11$$

These are the original two equations. In other words, the matrix product form is an equivalent way to state a system of equations.

NOTE

Do not enter a matrix of the type $\begin{bmatrix} x \\ y \end{bmatrix}$ from the keyboard. The elements of a matrix must always be assigned numeric values, and attempting to assign variable names such as x or y results in an error. The discussion above is provided only to help you understand the relationship between linear equations and matrix multiplication.

The INV Function

Syntax Form:

[Line number] array variable = INV array variable

Descriptive Form:

[Line number] target variable = INV parameter variable

PURPOSE

The INV function performs matrix inversion and solves systems of linear equations.

EXPLANATION

WHAT THE INV FUNCTION DOES

The INV function returns a new matrix which has the same size and shape as the original array. The square part of the new matrix is the inverse of the square part of the original matrix, and any columns which lie outside the square part in the new matrix, represent solutions to sets of linear equations. A detailed explanation of what this means is given later. Meanwhile, an example is given to illustrate what the INV function does:

$$A = \begin{bmatrix} 2 & 3 & 1 & 4 \\ 1 & 2 & 3 & 0 \end{bmatrix} \quad , \quad B = \text{INV}(A) = \begin{bmatrix} 2 & -3 & -7 & 8 \\ -1 & 2 & 5 & -4 \end{bmatrix}$$

Notice that when the 2X4 matrix A is supplied to the INV function, the result B = INV(A) is also a 2X4 matrix.

The Square Part

The square part of matrix B is the inverse of the square part of matrix A. The square parts of A and B are the shaded portions shown below:

$$A = \begin{bmatrix} \boxed{\begin{matrix} 2 & 3 \\ 1 & 2 \end{matrix}} & \begin{matrix} 1 & 4 \\ 3 & 0 \end{matrix} \end{bmatrix} \quad , \quad B = \text{INV}(A) = \begin{bmatrix} \boxed{\begin{matrix} 2 & -3 \\ -1 & 2 \end{matrix}} & \begin{matrix} -7 & 8 \\ 5 & -4 \end{matrix} \end{bmatrix}$$

The shaded portion in each matrix is called the square part because it is the largest square that can be blocked off, starting in the upper left-hand corner of the matrix.

The shaded portion in B=INV(A) is the inverse of the shaded portion in A:

$$\begin{bmatrix} 2 & -3 \\ -1 & 2 \end{bmatrix} \quad \text{is the inverse of} \quad \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix}$$

Saying that one matrix is the inverse of another means that when the two are multiplied, the result is an identity matrix. (Refer to the IDN Function topic "Additional Comments: Properties of Identity Matrices".)

The Extra Columns

In the previous example, both matrices A and B have two columns which lie outside the square part. The extra columns are shown below:

$$A = \begin{bmatrix} 2 & 3 & 1 & 4 \\ 1 & 2 & 3 & 0 \end{bmatrix} \quad , \quad B = \text{INV (A)} = \begin{bmatrix} 2 & -3 & -7 & 8 \\ -1 & 2 & 5 & -4 \end{bmatrix}$$

The two extra columns in B represent solutions to two different sets of linear equations. The first set of equations is

$$\begin{aligned} 2x + 3y &= 1 \\ x + 2y &= 3 \end{aligned}$$

and the first extra column in B represents the solution x = −7 and y = 5. The second set of equations is

$$\begin{aligned} 2x + 3y &= 4 \\ x + 2y &= 0 \end{aligned}$$

and the second extra column in B represents the solution x = 8 and y = −4.

The coefficients of x and y are the same for both sets of linear equations, and can be found in the square part of matrix A:

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

The constants which appear to the right of the equal sign in the first set of equations, appear in the first column to the right of the square part in A:

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

Likewise, the constants which appear in the **second** set of equations, are found in the **second** column to the right of the square part in A:

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

The solution for the first set of equations is found in the first column to the right of the square part in B:

$$B = \text{INV}(A) = \begin{bmatrix} 2 & -3 & -7 & 8 \\ -1 & 2 & 5 & -4 \end{bmatrix}$$

The solution for the **second** set of equations is found in the **second** column to the right of the square part in B:

$$B = \text{INV}(A) = \begin{bmatrix} 2 & -3 & -7 & 8 \\ -1 & 2 & 5 & -4 \end{bmatrix}$$

In summary, each extra column in B=INV(A) represents the solution to a set of equations which uses the elements of the square part in A for the coefficients of x and y. For every extra column in B, there is a corresponding extra column in A which contains the constants that appear to the right in the set of equations. Fig. 2-1 summarizes the information obtained from this example.

NOTE

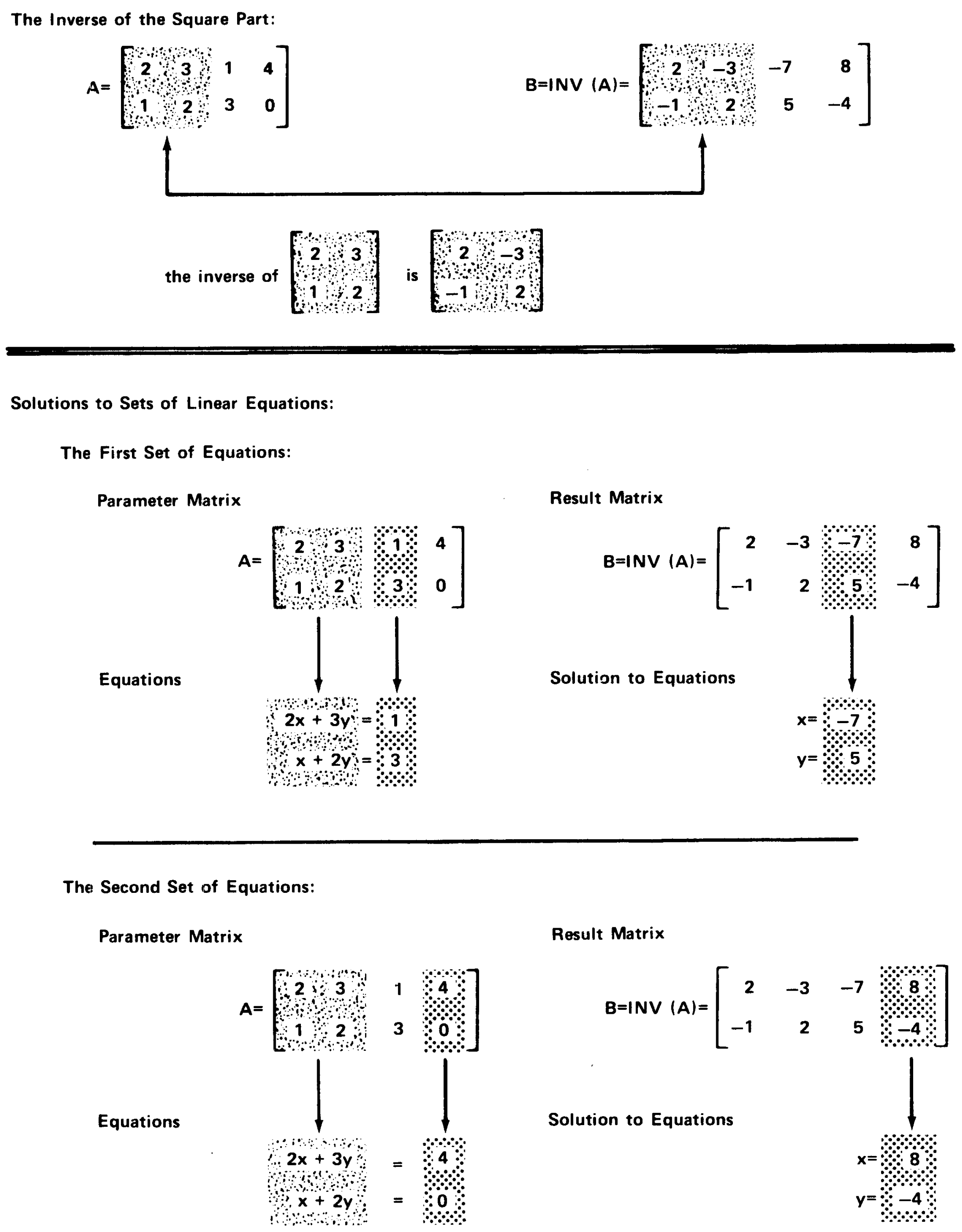
For those who are accustomed to representing systems of linear equations in matrix form, it may be more convenient to express the equations and solutions as follows:

First System of Equations — Solution

$$\begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -7 \\ 5 \end{bmatrix}$$

Second System of Equations

$$\begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ -4 \end{bmatrix}$$



$$A=\begin{bmatrix} 2 & 3 & 1 & 4 \\ 1 & 2 & 3 & 0 \end{bmatrix} \qquad B=\text{INV}(A)=\begin{bmatrix} 2 & -3 & -7 & 8 \\ -1 & 2 & 5 & -4 \end{bmatrix}$$

Fig. 2-1. An example of How to Interpret the Result of the INV Function.

CAUTION

It may be tempting in this example to think of matrix A as representing the following system of linear equations:

$$\begin{aligned} 2x + 3y + z &= 4 \\ x + 2y + z &= 0 \end{aligned}$$

This is **not** *a correct interpretation. The INV function always solves systems of N equations in N unknowns, where N stands for the number of rows in the parameter matrix. Since there are two rows in matrix A above, INV(A) provides solutions to systems of two equations in two unknowns (x and y).*

The INV Function and Square Matrices

When the INV function is performed on a square matrix, the result is the inverse of the original matrix. For example:

$$A = \begin{bmatrix} 1 & 0 & 2 \\ -1 & 3 & 0 \\ 0 & 1 & 1 \end{bmatrix} , \quad B = \mathrm{INV}(A) = \begin{bmatrix} 3 & 2 & -6 \\ 1 & 1 & -2 \\ -1 & -1 & 3 \end{bmatrix}$$

The shaded portions shown above are the square parts of matrices A and B. Notice that the square part of A is the entire matrix A, and the square part of B is all of matrix B.

When the INV function is performed, the square part of the result is the inverse of the square part of the original matrix. This means that for this example, matrix B is the inverse of matrix A.

When the INV function is performed, extra columns in the result represent solutions to sets of linear equations. But in this example, there are no extra columns to the right of the square part of A or B, so the result does not provide solutions to sets of linear equations.

To summarize, when the INV function is performed, the square part of the result is the inverse of the square part of the original matrix, and any remaining columns represent solutions to sets of linear equations. When the matrix supplied to the INV function has a square shape, there are no extra columns, and the resulting matrix is the inverse of the original matrix.

HOW TO USE THE INV FUNCTION

Dimensioning the Parameter Variable

The parameter of the INV function must be a matrix having at least as many columns as rows. For example, the INV function can be performed on the following matrices:

$$\begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix} , \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & 4 \end{bmatrix} , \begin{bmatrix} 1 & 2 & 3 & 0 \\ 0 & -1 & 4 & 6 \end{bmatrix}$$

If a matrix is to be used as a parameter for the INV function, it must previously be dimensioned in a DIM statement, using two subscripts to indicate the number of rows and columns. For example, the statement DIM A(3,4) dimensions a matrix which may be used as a parameter for the INV function.

Since the matrix must have at least as many columns as rows, the second subscript used to dimension the parameter must be greater than or equal to the first subscript, or an error will occur. For example, if matrix A is dimensioned to be a 4X3 matrix in the statement DIM A(4,3) attempting to perform INV(A) results in an error.

Dimensioning the Target Variable

When the INV function is performed, a new matrix is generated which has the same size and shape as the original matrix. The new matrix must be assigned to a target variable. The target variable must be dimensioned in a DIM statement, using two subscripts to indicate the number of rows and columns. Since the new matrix has the same size and shape as the original matrix, the subscripts used to dimension the target variable must be the same as those used to dimension the parameter variable.

Naming the Target Variable

The target variable may have the same name as the parameter variable. For example, the following statement can be used:

```
A = INV(A)
```

However, when the statement is executed, the original matrix A is replaced by the new matrix generated by the INV function. Care should be taken not to allow the original matrix A to be overwritten in this way, if A is to be used in later calculations. In general, it is good practice to use different names for the target variable and the parameter variable.

If a matrix has been overwritten by the result of the INV function, the original matrix may be recovered by performing the INV function again; however, truncation may occur during the calculations, causing the answer to be different from the original matrix. How closely the answer resembles the original matrix depends upon the "sensitivity" of the parameter matrix. A matrix is called "sensitive" if the values assigned to its elements make the matrix more susceptible to the truncation errors that normally occur when arithmetic operations are performed.

Matrices which are very sensitive to truncation error during the INV process are called "ill-conditioned" with respect to inversion. For discussion of truncation and the condition of a matrix, refer to the Appendix at the back of this manual.

A Sample Program

The following program illustrates how the INV function may be used to invert a matrix and solve a set of linear equations:

```
100 DEL A,B
110 DIM A(3,4),B(3,4)
120 READ A
130 DATA 1,0,1,0,2,−1,−2,3,−2,1,1,0
140 B = INV(A)
150 PRINT "B = INV A =";B
```

This program performs the INV function for the following 3X4 matrix:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 2 & -1 & -2 & 3 \\ -2 & 1 & 1 & 0 \end{bmatrix}$$

Line 140 computes INV(A) and assigns the result to the target variable B. When line 150 is executed, the result appears on the display:

$$B = INV(A) = \begin{bmatrix} 1 & 1 & 1 & 3 \\ 2 & 3 & 4 & 9 \\ 0 & -1 & -1 & -3 \end{bmatrix}$$

The square part of the resulting matrix B is the inverse of the square part of the original matrix A:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 0 & -1 & -1 \end{bmatrix} \quad \text{is the inverse of} \quad \begin{bmatrix} 1 & 0 & 1 \\ 2 & -1 & -2 \\ -2 & 1 & 1 \end{bmatrix}$$

The fourth column of matrix B represents the solution for the set of linear equations shown below:

$$\begin{array}{rrrr} x & & +z & =0 \\ 2x & -y & -2z & =3 \\ -2x & +y & +z & =0 \end{array}$$

The coefficients of x, y and z in the equations are the elements found in the square part of A, and the constants which appear on the right hand side of the equations are found in the last column of A.

The solution to the set of equations is represented by the last column of matrix B, as follows:

$$\begin{array}{rr} x= & 3 \\ y= & 9 \\ z= & -3 \end{array}$$

When the INV Function Can Be Performed

The Shape of the Parameter

The INV function can only be performed when the parameter matrix has at least as many columns as rows. (Refer to the topic "Dimensioning the Parameter".) Attempting to perform the INV function on a matrix which has more rows than columns, results in an error.

The Square Part of the Parameter

The INV function can only be successfully performed when the square part of the parameter matrix has an inverse. Not every square matrix has an inverse. For instance, the matrix C shown below does not have an inverse:

$$C = \begin{bmatrix} 2 & -2 \\ -5 & 5 \end{bmatrix}$$

When the INV function is unable to compute an answer because the square part of the parameter matrix has no inverse, an INVALID FUNCTION ARGUMENT error message appears on the display. For instance, attempting to perform the INV function results in an error for any of the matrices shown below:

$$\begin{bmatrix} 2 & -2 \\ -5 & 5 \end{bmatrix}, \begin{bmatrix} 2 & -2 & 0 \\ -5 & 5 & 1 \end{bmatrix}, \begin{bmatrix} 2 & -2 & 3 & -1 \\ -5 & 5 & 0 & 9 \end{bmatrix}, \begin{bmatrix} 2 & -2 & 1 & 2 & 0 \\ -5 & 5 & 3 & 4 & 5 \end{bmatrix}$$

For each of these matrices, the square part (the shaded portion) is the 2X2 matrix C from the last example. Since C does not have an inverse, attempting to perform the INV function on any of the above matrices results in an INVALID FUNCTION ARGUMENT error message.

When the INV Function Cannot Be Performed

Avoiding an Error Message

When the INV function is unable to compute a result because the square part of the parameter matrix has no inverse, an error occurs. An INVALID FUNCTION ARGUMENT message appears whether the error is caused while executing statements directly, or while running a BASIC program. However, if the error results while a program is executing, the BASIC interpreter behaves as if a SIZE ERROR has occurred, so that the ON SIZE THEN... statement may be used to handle the error without terminating program execution.

When an ON SIZE THEN... statement is executed in a BASIC program, subsequent SIZE ERRORs do not halt program execution; instead, control is diverted to the line number specified in the ON SIZE THEN... statement. (Refer to the Graphic System Reference Manual for an explanation of ON...THEN... statements.)

The following program illustrates how the ON SIZE THEN... statement may be used to avoid a SIZE ERROR:

```
100 ON SIZE THEN 220
110 DELETE A,B
120 DIM A(3,4),B(3,4)
130 PRINT "DO YOU WISH TO PERFORM THE INV FUNCTION?"
140 PRINT "IF SO, TYPE '1'; IF NOT, TYPE '0'."
150 INPUT T
160 IF T=0 THEN 250
170 PRINT "ENTER THE ELEMENTS OF A 3X4 MATRIX, IN ROW ORDER:"
180 INPUT A
190 B = INV (A)
200 PRINT " B = INV (A)=";B
210 GO TO 130
220 PRINT "THE VALUE OF THE DETERMINANT IS"; DET
230 PRINT "THE SQUARE PART OF THE MATRIX HAS NO INVERSE."
240 GO TO 130
250 END
```

This program performs the INV function on 3X4 matrices. The statement ON SIZE THEN 220 alerts the system to the possibility of a SIZE ERROR occurring at execution time. If a SIZE ERROR occurs while the program is running, control is transferred to line 220. A message then appears on the display, explaining that since the square part of the parameter has no inverse, the INV function cannot be performed. In this way, a SIZE ERROR has been avoided, and the program continues to ask for more data.

Solving Systems of Equations

When the square part of the parameter matrix has no inverse and attempting to perform the INV function results in an error, the equations represented by the matrix have no solution.

For example, the square part (the shaded portion) of the matrix shown below has no inverse, and attempting to perform the INV function results in an INVALID FUNCTION ARGUMENT error:

$$C = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 6 & 9 \end{bmatrix}$$

Matrix C represents the following set of two equations in two unknowns:

$$\begin{aligned} x + 2y &= 1 \\ 3x + 6y &= 9 \end{aligned}$$

Since no solution exists for the above equations, it makes sense that the INV function results in an error.

As another example, suppose you need to solve the following equations:

$$\begin{aligned} y + 4z &= 1 \\ -x \quad + 3z &= 0 \\ 2x + y - 2z &= 1 \end{aligned}$$

The first step is to represent the equations by a matrix:

$$D = \begin{bmatrix} 0 & 1 & 4 & 1 \\ -1 & 0 & 3 & 0 \\ 2 & 1 & -2 & 1 \end{bmatrix}$$

The next step is to perform the INV function on matrix D, in order to obtain a new 3X4 matrix whose fourth column represents the solution for the set of equations.

However, the square part of matrix D has no inverse, and the INV function returns an INVALID FUNCTION ARGUMENT error. The INV function is unable to solve the equations.

As in the previous example, it makes sense that the INV function cannot compute an answer, because no solution exists for the three equations.

When a matrix is used to represent a system of linear equations as in the preceding two examples, the square part of the matrix is called the "matrix of coefficients" because its elements are the coefficients of the variables in the equations. When the INV function is unable to compute an answer because the matrix of coefficients has no inverse, the system of linear equations has no solution (or does not have a unique solution).

THINGS TO WATCH OUT FOR WHEN USING THE INV FUNCTION

Undefined Variable	Not assigning a numeric value to every element of the parameter matrix, causes an UNDEFINED VARIABLE error to occur when the INV function is performed.
Syntax Error	Not assigning the result of the INV function to a target variable, causes a SYNTAX ERROR: for example, the statement INV(A) is not a valid statement.
Invalid Function Argument	Forgetting to dimension the target variable results in an INVALID FUNCTION ARGUMENT error message.
	Likewise, not dimensioning the target variable to have the same size and shape as the parameter variable, causes an INVALID FUNCTION ARGUMENT error when the INV function is performed. The subscripts used to dimension the target variable must be the same as the subscripts used to dimension the parameter variable.
	Attempting to perform the INV function when the square part of the parameter matrix has no inverse, generally causes an INVALID FUNCTION ARGUMENT error message to appear on the display. However, if the error is caused during execution of a program, the BASIC interpreter behaves as if a SIZE ERROR has occurred. This enables the system to respond to the ON SIZE THEN... statement and handle the error without terminating program execution.
Shape Error	Attempting to perform the INV function on a matrix which has more rows than columns, causes a SHAPE ERROR: the parameter matrix must have at least as many columns as rows.
	Attempting to perform the INV function on a matrix which has more than 255 rows or more than 255 columns, also causes a SHAPE ERROR.

The DET Function

Syntax Form:

```
[ Line number ] array variable = INV array variable
[ [ Line number ]   numeric variable = ] DET
```

NOTE

As indicated in the syntax form above, the INV function must be performed in order to use the DET function. The value of the determinant is computed during the INV process. For this reason, it is important to understand the INV function before attempting to use DET.

PURPOSE

The DET function returns the value of the determinant.

EXPLANATION

THE DETERMINANT OF A SQUARE MATRIX

The determinant of a matrix is a function of the elements in the matrix. The value of the determinant is a numeric constant, and can be computed for any square matrix.

Computing the Determinant of a Square Matrix by Hand

NOTE

The following discussion is provided to illustrate what the determinant of a matrix is, by showing one method of computing its value by hand. This is not an explanation of the algorithm the DET function uses to compute the value of the determinant.

The value of the determinant is a numeric constant, and can be computed by hand from the elements of the array. For example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \text{DET} = -2$$

For a 2X2 matrix as in this example, the value of the determinant is the product of the diagonal elements in the upper-left and lower-right corners, minus the product of the elements on the other diagonal:

$$\text{DET} = 1*4 - 3*2 = -2$$

Computing the determinant of a 3X3 matrix by hand is a little harder.

For example:

$$A = \begin{bmatrix} -3 & 1 & 2 \\ 6 & 0 & 3 \\ 2 & -1 & 0 \end{bmatrix}, \quad \text{DET} = -15$$

One way to begin the calculation is to "expand" along the first row as follows: multiply each element along the first row by the determinant of the matrix found by covering up the row and column containing that particular element. For instance, multiply the element A(1,1) by the determinant of the matrix found by covering up the first row and first column of A:

$$\begin{bmatrix} -3 & 1 & 2 \\ 6 & 0 & 3 \\ 2 & -1 & 0 \end{bmatrix}, \quad -3*[0*0-(-1)*3]=-9$$

Notice that the determinant of the 2X2 matrix $\begin{bmatrix} 0 & 3 \\ 1 & 0 \end{bmatrix}$ is obtained by the method described in the first example, that is 0*0 −(−1)*3=3.

Next, multiply the element A(1,2) by the determinant of the matrix found by covering up the first row and second column of A:

$$\begin{bmatrix} -3 & 1 & 2 \\ 6 & 0 & 3 \\ 2 & -1 & 0 \end{bmatrix}, \quad 1*[6*0 - 2*3] = -6$$

Finally, multiply A(1,3) by the determinant of the matrix found by covering up the first row and third column of A:

$$\begin{bmatrix} -3 & 1 & 2 \\ 6 & 0 & 3 \\ 2 & -1 & 0 \end{bmatrix}, \quad 2*[6*-1 -2*0] = -12$$

Three numbers are obtained in this way, −9, −6, and −12. The value of the determinant of A is the first number, minus the second number, plus the third number:

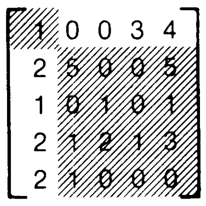
$$\text{DET} = -9 + 6 - 12 = -15$$

The same method can be used to find the determinant when A is a larger matrix. Each element along the first row of A is multiplied by the determinant of the matrix found by covering up the row and column containing that element. The value of the determinant of A is equal to the first number, minus the second number, plus the third number, minus the fourth number, and so on.

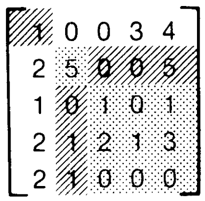
The computations are tedious for matrices having more than three rows and three columns. For example, take A to be the 5X5 matrix shown below:

$$A = \begin{bmatrix} 1 & 0 & 0 & 3 & 4 \\ 2 & 5 & 0 & 0 & 5 \\ 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 2 & 1 & 3 \\ 2 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad \text{DET} = -43$$

The first step in computing the determinant is to multiply A(1,1) by the determinant of the matrix found by covering up the first row and first column of A:



Now to compute the value of the determinant of the 4X4 matrix shaded above, begin by multiplying the first element in its first row by the determinant of a 3X3 matrix:



This process continues until all of the required "subdeterminants" have been reduced to a numeric value. Notice that **each** element in the first row of matrix A must be multiplied by a 4X4 determinant, whose value must be reduced to a numeric constant by performing the necessary expansions on 3X3 matrices, and so on.

It is easy to see the advantage of using the DET function to compute the determinant of such a matrix; the DET function calculates the value of the determinant in a fraction of the time it takes to complete the calculation by hand.

WHAT THE DET FUNCTION DOES

The DET function returns the determinant of the square part of a matrix which has been used as a parameter for the INV function. For example:

$$A=\begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & -3 \end{bmatrix}, \quad B = \text{INV}(A) = \begin{bmatrix} 2 & -1 & 3 \\ -1 & 1 & -3 \end{bmatrix}, \quad \text{DET} = 1$$

As indicated above, after the INV function is performed on matrix A, the DET function is used to compute the determinant of the square part (the shaded portion) of A.

HOW TO USE THE DET FUNCTION

When to Perform the DET Function

The DET function is always performed after the matrix has been supplied to the INV function: in other words, the INV function is performed first, then the DET function.

As an example, the following sequence of statements may be used to find the determinant of a 5X5 matrix:

```
DEL A,B
DIM A(5,5),B(5,5)
INPUT A
B = INV(A)
DET
```

Notice that the statement B = INV(A) is executed before the statement DET.

The DET function does not necessarily have to be performed **immediately** after the INV function. No matter how many BASIC statements have been executed since the last use of the INV function, the DET function always returns the determinant of the square part of the matrix most recently supplied to the INV function.

Syntax Forms

When executing statements directly from the keyboard, the value of the determinant may be obtained by entering DET and pressing the RETURN key.

When the DET function appears in a BASIC program, the value of the determinant may be obtained from a statement such as 100 PRINT DET. (A statement such as 100 DET results in an error.)

In either case, the INV function must be performed on the matrix before performing the DET function.

The value of the determinant may be assigned to a numeric variable in a statement such as X = DET or 100 X = DET. (Refer to the topic "Assigning the Result to a Target Variable" on the following pages.)

A Sample Program

The following program illustrates how the DET function can be used to find the determinant of the square part of a matrix:

```
100 DEL A,B
110 DIM A(2,3),B(2,3)
120 READ A
130 DATA 1,1,0, 1,2,−3
140 B = INV(A)
150 PRINT "A =";A;
160 PRINT "DET=";DET
```

This program computes the determinant of the square part (the shaded portion) of the following matrix:

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & -3 \end{bmatrix}$$

The INV function is performed on matrix A in line 140, and the DET function is used in line 160. When line 160 is executed, the value of the determinant appears on the display:

```
DET = 1
```

The important thing to keep in mind is that the INV function is always performed first, then the DET function.

Assigning the Result to a Target Variable

The DET function always returns the determinant of the square part of the matrix most recently supplied to the INV function. This means that as soon as the INV function is performed on a new matrix, the result of the DET function changes.

To store the value of the determinant for later use, the result of the DET function may be assigned to a numeric variable in a statement such as X = DET or 100 X = DET. The numeric variable is called the target variable because it serves as a "target" for the value of the determinant.

Naming the Target Variable

It is important to keep in mind that the result of the DET function is a numeric constant, and should be assigned to a numeric variable (that is, a variable which has not previously appeared in a DIM statement). Thus, the target variable should not have the same name as an array. For example, if the INV function is performed on a 3X3 matrix named A, the assignment A = DET replaces all nine elements of A by the numeric constant resulting from the DET function.

THINGS TO WATCH OUT FOR WHEN USING THE DET FUNCTION

Syntax Error	Entering a statement such as 100 DET is a SYNTAX ERROR. A correct statement is 100 PRINT DET. (When statements are entered without a line number, the keyword DET may be used by itself.)
Undefined Variable	Forgetting to perform the INV function before the DET function, causes an UNDEFINED VARIABLE error message to appear on the display. The INV function must be performed first, then the DET function.

NOTE

The DET function always returns the determinant of the square part of the matrix most recently supplied to the INV function. Forgetting to perform the INV function before the DET function may result in an incorrect answer. For example, when finding the determinant of a matrix called B, if DET is performed before INV(B), the answer which appears on the display is not the determinant of matrix B; it is the determinant of whatever matrix was last supplied to the INV function.

Appendix A

SUPPLEMENTARY INFORMATION

This Appendix provides supplementary information about the suitability of certain matrices for inversion and computation of the determinant. The "condition" of a matrix is a very complicated topic, and cannot be fully explained in a few pages. The information contained in this Appendix is presented as concisely as possible, and is intended only to help you understand how the properties of a particular matrix may affect the results of the INV and DET functions.

MATHEMATICAL INVERTIBILITY VERSUS NUMERICAL INVERTIBILITY

In mathematical theory, a matrix either has an inverse or does not have an inverse. If a matrix has no inverse, the value of the determinant is zero, and the matrix equation A MPY X = B has no solution (or cannot be uniquely solved). But if a matrix has an inverse, the determinant is non-zero, and A MPY X = B may be solved for any B.

In practice, however, a matrix is called non-invertible when the sequence of arithmetic operations used to compute the inverse reaches a point past which the calculations cannot proceed further. This definition is not the same as the mathematical definition of invertibility. Many matrices are invertible in the mathematical sense, but not in the computational sense. For instance, the following matrix is mathematically invertible, but the inverse cannot necessarily be computed if ε is much smaller than 1:

$$A = \begin{bmatrix} 1-\varepsilon & 1+\varepsilon \\ 1 & 1+2\varepsilon \end{bmatrix}$$

Although theoretically an inverse exists for this matrix, most machines are unable to compute an answer.

Conversely, a mathematically non-invertible matrix may be "inverted" on most machines without causing an error. For example:

$$B = \begin{bmatrix} a & 1 \\ a\uparrow 2 & a \end{bmatrix}$$

In theory, this matrix has no inverse, yet in practice most algorithms used to compute inverses return an answer if 1/a cannot be represented with complete precision (a=3 is an example of this).

TRUNCATION ERROR

Truncation is the reason for the difference between what should theoretically happen and what actually happens when a machine is used to calculate an inverse. Truncation can occur within any machine while arithmetic operations are being performed, and while numbers are being converted from decimal to binary representations. Whenever a binary number has more bits than the machine can represent, the extra bits are chopped off (truncated). Truncation is unavoidable, since there are many numbers which cannot be represented with complete precision on any machine.

When a sequence of arithmetic operations is performed, some truncation usually occurs. The total amount of error depends on the numbers involved in the calculations. For matrix inversion, the numbers involved are the values assigned to the elements of the matrix, and the "condition" of a matrix is an indication of how these values influence the total amount of truncation error.

THE CONDITION OF A MATRIX

A matrix C is called "ill-conditioned" with respect to inversion when small changes in the elements of C cause large changes in the computed inverse of C, or cause C to be non-invertible in a computational sense. It may not be worth attempting to compute the inverse of a very ill-conditioned matrix, because small errors in the original data (the values assigned to the elements of the matrix) can cause the computed answer to be unrecognizably different from the true inverse of the matrix. Ill-conditioned matrix also tend to be more susceptible to truncation error than other matrices.

A matrix which is ill-conditioned with respect to inversion is more susceptible to truncation error in the sense that as arithmetic operations proceed to compute the inverse, truncation error tends to accumulate faster than it would for better-conditioned matrices. Normal amounts of truncation error do "pile up" even for well-conditioned matrices, just because so many computations are needed to invert a matrix, and every step in the process provides another chance for error to occur. (Thus large matrices are more susceptible to truncation error than small ones.) Yet when an ill-conditioned matrix is inverted, much more than the normal amount of truncation error may accumulate, causing the result to be too inaccurate to be called an inverse at all.

The most extreme example of this is a non-invertible matrix, that is, one which has no inverse in the mathematical sense. This type of matrix is the worst case of ill-conditioning with respect to inversion, and the algorithm used to compute the inverse is able to return an answer, the answer is completely inaccurate. So much truncation error has accumulated during the calculations that the machine appears to have found an inverse, when actually no inverse exists.

THE CONDITION NUMBER

The condition of a matrix with respect to inversion can be measured, and the measure is called the condition number k. For any matrix C, the condition number may be calculated as follows:

k = | largest element of C * largest element of INV(C) |

If k is close to 1, matrix C is well-conditioned with respect to inversion. However, if k is much larger than 1, C is less well-conditioned; and if k is extremely large, C is called ill-conditioned, and the computed inverse of C is likely to be very far from the true inverse of C.

The number 1/k is a rough measure of the "distance" from matrix C to the nearest mathematically non-invertible matrix. For instance, if the condition number k is 10^{15}, C may be made non-invertible by increasing or decreasing at least one of its elements by about 10^{-15} times the largest element in the matrix.

How to Use the Condition Number k to Evaluate the Result of the INV Function

The condition number may be used as a check of the accuracy of the result of the INV function. If the distance 1/k from C to the closest non-invertible matrix is comparable in magnitude to possible errors in the data (the values assigned to the matrix), the result of performing the INV function may be so inaccurate that INV(C) is meaningless. For example, if the values assigned to the elements of matrix C are physical observations which are only accurate to three places, a condition number k of 10^3 or higher indicates that the result of INV(C) may be too inaccurate to be meaningful. Likewise, if the distance 1/k from C to the closest non-invertible matrix is comparable in magnitude to the precision of the machine, 10^{-15}, the result of INV(C) may be too inaccurate to be of any practical value. This means that if the condition number k is 10^{15}, the distance from C to the closest non-invertible matrix is roughly 10^{-15}, which is the same as the amount of error which might normally occur during any arithmetic operation such as addition or subtraction. In general, if the condition number k is 10^{15} or larger, the matrix may be so ill-conditioned with respect to inversion that the INV function cannot be expected to compute a meaningful result.

Evaluating the Result of the DET Function

The value of the determinant is computed during the INV process. For this reason, the result of the DET function is affected by the condition of the matrix in the same manner as the INV function. When the result of the INV function is inaccurate because the parameter matrix is ill-conditioned with respect to inversion, the value returned by the DET function is not likely to be close to the true value of the determinant.

Thus, if the values assigned to the elements of the matrix are based on physical observations, 1/k should not be smaller than possible errors in the original data. Likewise, a condition number k greater than or equal to 10^{15} indicates that the parameter matrix is so sensitive to truncation error that the result of the DET function may be very different from the true value of the determinant.

This means that the value returned by the DET function cannot be used as an indication of whether or not a matrix is invertible. In mathematical theory, a matrix has no inverse when the value of the determinant is 0; but in practice, if the parameter matrix is non-invertible or ill-conditioned with respect to inversion, the value returned by the DET function may not even be close to 0.

An Example

Suppose you want to use the INV function to find the inverse of the following 3x3 matrix:

$$R = \begin{bmatrix} 1 & 0 & 1 \\ 2 & -1 & 5 \\ 3 & -1 & 6 \end{bmatrix}$$

Performing INV(R) results in the following matrix:

$$S = INV(R) = \begin{bmatrix} 6.251999482E+13 & 6.254999482E+13 & -6.254999482E+13 \\ -1.876499845E+14 & -1.876499845E+14 & 1.876499845E+14 \\ -6.254999482E+13 & -6.254999482E+13 & 6.254999482E+13 \end{bmatrix}$$

Since performing the INV function does not result in an error, you might be tempted to accept matrix S as the inverse of matrix R, without checking the accuracy of the result S.

But a check of the condition number k reveals that matrix R is extremely ill-conditioned with respect to inversion:

$$\begin{aligned} k &= | \text{largest element of } R * \text{largest element of } S | \\ &= R(3,3)*S(2,3) \\ &= 1.125899907\ E+15 \end{aligned}$$

The condition number k is so large, and R is so ill-conditioned with respect to inversion, that the INV function cannot be expected to return an accurate inverse.

Since the size of the condition number indicates that R may not be accurately inverted, it is a good idea to check the result by performing R MPY S and S MPY R. How accurate the result returned by the INV function is, depends on how closely the elements of the products resemble those of the 3X3 identity matrix.

The products R MPY S and S MPY R are shown below:

$$M = R \text{ MPY } S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 2 & 2 & 0 \end{bmatrix}, \qquad N = S \text{ MPY } R = \begin{bmatrix} 3 & -0.5 & 2 \\ -6 & 2 & -8 \\ -2 & 0.5 & -2 \end{bmatrix}$$

Neither product resembles the 3X3 identity matrix, confirming that matrix R is too ill-conditioned to be accurately inverted.

In general, after using the INV function to invert a matrix and solve systems of linear equations, or after performing the DET function, it is useful to check the condition number k. An extremely large condition number indicates that the parameter matrix is not suitable for matrix inversion or computation of the determinant, and that the results of the INV and DET functions should not be assumed to be accurate.

Appendix B

FUNCTION SUMMARY

FUNCTION	PAGE NO.	EXAMPLE	EXPLANATION	ALGORITHM
TRN	2-3	B=TRN(A)	Returns the transpose of a matrix. Each row (or column) in A becomes the corresponding row column) in B=TRN(A). If A is an IXJ matrix, B=TRN(A) is a JXI matrix.	B(I,J) = A(J,I)
IDN	2-8	CALL "IDN", I or A$ = "IDN" CALL A$,I	Creates a matrix whose diagonal elements I(1,1), I(2,2), I(3,3),... are 1's, and all other elements are 0's.	I(I,J) = 1 if I = J I(I,J) = 0 if I ≠ J
MPY	2-15	C = A MPY B	Returns the matrix product of two arrays. If A is an IXJ matrix, and B is a JXK matrix, the product C is an IXK matrix.	$C(I,K) = \sum_{L=1}^{K} A(I,L)*B(L,K)$
INV	2-26	B=INV(A)	Returns the inverse of the square part of the parameter matrix, and solves systems of equations represented by the matrix.	The Gauss-Jordan technique is performed in place, with full row and column pivoting.
DET	2-37	B=INV(A) followed by DET or X=DET	Returns the determinant of the square part of the matrix most recently supplied to the INV function.	The value of the determinant is the product of the pivot elements computed during the INV process.

Appendix C

ERROR MESSAGES

CALL NAME INVALID IN IMMEDIATE LINE — MESSAGE NUMBER 32

— A string variable appears in the CALL statement without being previously assigned the value "IDN".

INVALID COMMAND ARGUMENT — MESSAGE NUMBER 12

— The target variable for the IDN function has not been dimensioned in a DIM statement.

— The target variable for the IDN function has been dimensioned using only one subscript.

— A semicolon appears in the CALL statement e.g. CALL "IDN";I

INVALID FUNCTION ARGUMENT — MESSAGE NUMBER 27

— The target variable for the INV, MPY, or TRN function has not been dimensioned in a DIM statement.

INVALID FUNCTION ARGUMENT — MESSAGE NUMBER 21

— The target variable for the INV function is incorrectly dimensioned.

INVALID FUNCTION ARGUMENT — MESSAGE NUMBER 30

— The target variable for the MPY function has the same name as one of the parameters.

INVALID FUNCTION ARGUMENT — MESSAGE NUMBER 31

— The square part of the parameter for the INV function has no inverse.

SHAPE — ERROR—MESSAGE NUMBER 8

— The parameter for the INV function has more rows than columns.

— The parameter for the INV function has more than 255 rows or more than 255 columns.

— The target variable for the MPY or TRN function is incorrectly dimensioned.

— The target variable or one of the parameters for the MPY function has been dimensioned using only one subscript.

— The parameters for the MPY function cannot be multiplied, because the number of columns in the first parameter is not equal to the number of rows in the second parameter.

— The parameter for the TRN function has a non-squareshape, and the the target variable has the same name as the parameter variable e.g. C=TRN(C).

SIZE ERROR — MESSAGE NUMBER 1

— Floating point overflow occurs when the MPY function is performed.

UNDEFINED VARIABLE — MESSAGE NUMBER 36

— One or more of the elements of a parameter for the INV, MPY, or TRN function has not been assigned a numeric value.

— The DET function has been performed without first supplying the matrix to the INV function.