Or-1785

ANL-75-44

ANL-75-44

A GUIDE TO TSO SPEAKEASY

by

**S. Cohen, J. Fink, F. J. D. Serduke,
and H. Z. Kriloff**

MASTER

U of C-AUA-USERDA

ARGONNE NATIONAL LABORATORY, ARGONNE, ILLINOIS

**Prepared for the U. S. ENERGY RESEARCH
AND DEVELOPMENT ADMINISTRATION
under Contract W-31-109-Eng-38**

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

The facilities of Argonne National Laboratory are owned by the United States Government. Under the terms of a contract (W-31-109-Eng-38) between the U. S. Energy Research and Development Administration, Argonne Universities Association and The University of Chicago, the University employs the staff and operates the Laboratory in accordance with policies and programs formulated, approved and reviewed by the Association.

MEMBERS OF ARGONNE UNIVERSITIES ASSOCIATION

The University of Arizona	Kansas State University	The Ohio State University
Carnegie-Mellon University	The University of Kansas	Ohio University
Case Western Reserve University	Loyola University	The Pennsylvania State University
The University of Chicago	Marquette University	Purdue University
University of Cincinnati	Michigan State University	Saint Louis University
Illinois Institute of Technology	The University of Michigan	Southern Illinois University
University of Illinois	University of Minnesota	The University of Texas at Austin
Indiana University	University of Missouri	Washington University
Iowa State University	Northwestern University	Wayne State University
The University of Iowa	University of Notre Dame	The University of Wisconsin

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights. Mention of commercial products, their manufacturers, or their suppliers in this publication does not imply or connote approval or disapproval of the product by Argonne National Laboratory or the U. S. Energy Research and Development Administration.

Printed in the United States of America
Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22161
Price: Printed Copy \$5.45; Microfiche \$2.25

ANL-75-44

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

A GUIDE TO TSO SPEAKEASY

by

S. Cohen, J. Fink,* and F. J. D. Serduke

Physics Division, ANL

and

H. Z. Kriloff**

Computer Center
University of Illinois, Chicago Circle

October 1975

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

*Now with Chemical Engineering Division, ANL
**Present address: Boeing Computer Services, Inc., Seattle, Washington

THIS PAGE
WAS INTENTIONALLY
LEFT BLANK

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	5
1. INTRODUCTION	6
Activating TSO Speakeasy	7
2. MATHEMATICS WITH SPEAKEASY ARITHMETIC	8
The Use of ANSWER	10
Special Functions	10
Scientific Notation	12
Variables	13
Operations with Arrays of Numbers	15
Two-Dimensional Arrays	19
Operations on Two-Dimensional Arrays	21
Operations on Parts of Objects	23
3. RELATIONAL, LOGICAL, AND CONDITIONAL OPERATORS	24
Relational Operators	24
Logical Operators	25
Conditional Statements	27
4. DIAGNOSTICS AND ERROR DETECTION	28
5. MATRIX AND VECTOR OPERATIONS	30
Simultaneous Equations	30
Algebraic Details	33
Operations on Matrices	38
6. CALCULUS AND SPEAKEASY	39
Derivatives	40
Integration	41
Other Techniques of Evaluating Definite Integrals:	
Gaussian Quadrature	43

TABLE OF CONTENTS

	<u>Page</u>
7. COMPLEX NUMBERS	45
8. PROGRAM MODE	47
Creating a Program	48
Using a Defined Program	48
Modifying and Correcting a Program	49
Allowable Speakeasy Statements in Program Mode	52
9. CONTROLLING THE FORM OF OUTPUT	59
10. OTHER USEFUL COMMANDS	62
11. LEARNING MORE ABOUT SPEAKEASY	65
12. CONCLUSION	69
INDEX	70

A GUIDE TO TSO SPEAKEASY

by

S. Cohen, J. Fink, F. J. D. Serduke and H. Z. Kriloff

ABSTRACT

Speakeasy is a computer programming system that is powerful yet easily learned. This book is intended as an introductory guide to this system. It assumes that the reader is a complete novice, both with regard to Speakeasy and to the use of TSO. This book is divided into several sections in which the processor is used in a desk-calculator mode to demonstrate its capabilities for matrix operations, for calculations in calculus and for operations involving logical and relational tests. The program mode and the use of available teaching aids is also described.

1. INTRODUCTION

Speakeasy is a method for using a computer. The Speakeasy language will provide you, the user, with the means of easily carrying out calculations and obtaining answers to questions in a wide variety of fields. Moreover, since the design of Speakeasy enables it to incorporate new developments as they occur in use, it is a rapidly expanding system.

The scope of this book therefore is limited. We will show you how to use Speakeasy to carry out real calculations; we will provide the background for certain general types of operations that are available; and we will show by examples how you actually make use of such operators.

Most importantly, we will show you how to use Speakeasy to learn more about Speakeasy. The usability of a system such as this depends very much on how well it is able to communicate with you, the user. Speakeasy has many features specifically designed to help you understand how to use what is available to you. This book will introduce you to those facilities.

The most dramatic way to use Speakeasy is under a time-sharing system such as IBM's TSO. The examples shown in this book are actual output from a terminal that gives access to such a version. Operating under TSO, you may experiment with Speakeasy and immediately see the consequence of each trial. This book is specifically designed to be used while sitting at a terminal with a TSO version of Speakeasy.

We hope that as you read this book you will experiment with Speakeasy and expand upon the examples that we have given. Whenever you become curious about the consequences of certain operations, feel free to try them out. It is by such experimentation that you will really learn the capabilities of this processor. The most tragic consequence of such experimentation will be an informative message by Speakeasy to explain a possible misunderstanding.

This book is divided into several sections. It is not necessary, therefore, that you completely read the book before beginning to use Speakeasy. As you will see, the very first part of this book provides you with a usable and powerful tool.

The first part of this book introduces you to Speakeasy in the responsive (i. e., manual) mode of operation. When used in this way, Speakeasy can be viewed as a super desk calculator. The presentation is, in fact, oriented towards this viewpoint. The concept of defined variables, structured sets of numbers, and many of the built-in functions are introduced by a series of examples interspersed with explanatory sentences. Although this section appears somewhat trivial, it is of utmost importance that you master it by making real use of the TSO-Speakeasy processor. The concepts are very simple and easily understood, but it is the natural

feeling of response that should be acquired at this simple level so that you can really feel at home with the processor in your work.

Facilities for matrix algebra, vector algebra, complex notation, and calculus are then presented. These are facilities that are available to you and that greatly enhance the power of Speakeasy to carry out your work. If you have not yet been exposed to these subjects, you may in fact use Speakeasy as a teaching aid to learn about them.

The program mode is then introduced to provide you with the means of instructing the system to carry out repetitive types of calculations. The final sections describe the HELP facility. This basic internal documentation of Speakeasy enables you to learn about other features available to you.

Activating TSO Speakeasy

TSO Speakeasy operates under the standard TSO system supplied by IBM. In order for you to make use of Speakeasy, it is therefore necessary that you first establish communication with that system. This is done in a manner that is determined locally by your computer installation and cannot be described here. In addition, it is possible to use many different kinds of terminals when communicating with TSO. This book describes the operation of Speakeasy while using an IBM-2741 type terminal. Other terminals are probably available to you; they are equally easily used, but specific information concerning operation will be required for each such terminal. You are referred to your user consultant for such specific information. He will also provide you with information about Speakeasy documentation and facilities that supplement this book.

If you have activated TSO properly, it will indicate that it is waiting for a request from you by typing the word

READY

You can then activate the Speakeasy processor by typing in the single word

SPEAKEZ

and hitting the return key. After a short pause the Speakeasy header line of the general form

TSO SPEAKEASY (the version) (the time) (the date)

is typed out and then the symbols[†]

:—

[†] Variants on these symbols may occur on special devices or under special TSO releases.

appear on the next line. This pair of symbols indicates that Speakeasy is waiting for a request from you. You enter the request by typing it in and pressing the return key. The forms of these requests and the responses from Speakeasy are the subject of this book.

2. MATHEMATICS WITH SPEAKEASY ARITHMETIC

In its most elementary mode of operation, Speakeasy functions as a desk calculator. An arithmetic expression is typed and entered, and Speakeasy returns the answer. Entering the line[†]

```
:_3+7
```

elicits the response

```
3+7 = 10
:_
```

or

```
:_8-1/2
```

triggers

```
8-1/2 = 7.5
:_
```

The input line can be longer and can involve many operations. For example, the entered problem and the response might be:

```
:_2+4+6+8-1-3-5-7
2+4+6+8-1-3-5-7 = 4
:_
```

The symbols used for the arithmetic operations are

**	exponentiation	3^2 is written 3**2
*	multiplication	2×3 is written 2*3
/	division	$\frac{8}{3}$ or $8 \div 3$ is written 8/3
+	addition	
-	subtraction	

[†]All of the examples in this book are direct reproductions of input and output from a terminal. Lines beginning with :_ are followed by the input as entered. Lines with the same type face are responses from Speakeasy.

The sequences of these operations is significant. This is the hierarchy or priority of operation by which an expression is evaluated. The three levels of the hierarchy, in order of decreasing priority, are

- 1) exponentiation,
- 2) multiplication and division, and
- 3) addition and subtraction.

In a given statement, exponentiations are performed first, multiplications and divisions are next, and additions and subtractions are last.

The expression $2^{**}5 - 4/2$ means

$$2^5 - \frac{4}{2} = 30$$

$$\text{and not } \frac{(2^5 - 4)}{2} = 14, \text{ nor } 2^{5 - \frac{4}{2}} = 2^3 = 8, \text{ nor } 2^{\frac{5-4}{2}} = 2^{\frac{1}{2}} = 1.414.$$

Furthermore within a given level of the hierarchy, the expression is evaluated from left to right. Consequently,

$$8/3*6 \text{ means } \frac{8}{3} \times 6 \text{ and not } \frac{8}{3 \times 6}$$

$$6/4 - 2 \text{ means } \left(\frac{6}{4}\right) - 2 \text{ and not } 6/(4 - 2).$$

This leads to the use of parentheses. Suppose we wanted to calculate

$$\frac{50}{10.07 + 2.43} \text{ and } \frac{16 + 9}{6.25}.$$

We would type in

$$\begin{aligned} &:_50/(10.07+2.43) \\ &50/(10.07+2.43) = 4 \\ &:_{(16+9)}/6.25 \\ &(16+9)/6.25 = 4 \\ &:_ \end{aligned}$$

The use of parentheses allows an unambiguous definition of the order of arithmetic operations since Speakeasy first examines the innermost level of parentheses, evaluates that expression, and then proceeds to resolve any other level(s) of parentheses enclosing the inner expression(s). Thus

$$47 \times \frac{2^4 + 7}{16 \times 8.6} - \frac{83}{16}$$

would be entered as

$$47 * ((2**4)+7) / (16*8.6) - (83/16).$$

If you are sitting at a TSO console, now is an excellent time to experiment with the concepts that have been discussed. For starters, evaluate

$$\frac{7.5 + 2.5}{2}, \quad \frac{6^{\frac{1}{2} + \frac{5}{2}} - 2^4}{36 + 4}, \quad \text{and} \quad 6 + 7^{4-2} - 10 \times 5.$$

This illustrates the general method of resolving any doubts about the meaning Speakeasy attaches to any combination of operations. Just set up an appropriate numerical example and try different ways of typing it in to see which gives the right answer.

The Use of ANSWER

The result of evaluating a single expression is given the name ANSWER. This can be quite useful in evaluating complicated expressions since ANSWER is assigned the result of the last statement entered. In evaluating

$$(6^{(\frac{1}{2} + \frac{5}{2})} - 2^4) / (36 + 4),$$

ANSWER may be used as follows:

```
:_6**(1/2+5/2)
6**(1/2+5/2) = 216
:_answer-2**4
ANSWER-2**4 = 200
:_answer/(36+4)
ANSWER/(36+4) = 5
:_
```

The advantage of solving a problem in this manner is that Speakeasy prints the ANSWER at each intermediate step. This facilitates error detection.

Special Functions

Built into Speakeasy are many of the special functions that appear in handbooks of mathematics. Among these are the trigonometric functions

$\text{SIN}(X)$ = trigonometric sine of angle X ,

$\text{COS}(X)$ = trigonometric cosine of angle X ,

$\text{TAN}(X)$ = trigonometric tangent of angle X ,

$\text{COT}(X)$ = trigonometric cotangent of angle X ,

as well as the inverse trigonometric functions

$\text{ASIN}(X)$ = arcsine of X ,

$\text{ACOS}(X)$ = arccosine of X ,

$\text{ATAN}(X)$ = arctangent of X ,

and the hyperbolic functions

$\text{SINH}(X)$ = hyperbolic sine of X ,

$\text{COSH}(X)$ = hyperbolic cosine of X .

The calling of the trigonometric functions of angles expressed in radians is illustrated in the record

```
:_sin(3/4)
SIN(3/4)  =  .68163
:_
```

If the use of degrees is more convenient in the calculations with trigonometric functions, first enter

```
:_angles in degrees
```

then

```
:_sin(90)
SIN(90)  =  1
:_sin(45)-sqrt(2)*0.5
SIN(45)-SQRT(2)*0.5  =  0
:_tan(45)
TAN(45)  =  1
:_sin(45)/cos(45)
SIN(45)/COS(45)  =  1
:_
```

To revert back to radians, enter

```
:_angles in radians
:_
```

The constant π may be easily specified in terms of the trigonometric identity $\pi = 2 \arccosine 0$, as in the record

```
:_pi=2*acos(0)
:_print pi
PI = 3.1416
:_
```

The results have been printed to 5 significant figures. The number of significant figures printed may be controlled by use of the word SIGNIFICANCE followed by the number of significant figures desired. For example,

```
:_significance 15
:_print pi
PI = 3.14159265358979
:_significance 5
:_print pi
PI = 3.1416
:_
```

Other available functions are logarithms, exponentials (antilog), and square roots, for which the Speakeasy names are

LOG	Napierian logarithm (base e)
EXP	exponential function (Napierian antilog)
SQRT	square root

These, and many other special functions, are found by using the section called HELP. You will learn about more of these as you read this book.

All Speakeasy functions are very easy to use, for example,

```
:_log(2*8-6)
LOG(2*8-6) = 2.3026
:_exp(answer)
EXP(ANSWER) = 10
:_
```

Scientific Notation

Very large or very small numbers are most easily expressed in scientific notation. In Speakeasy, such numbers are expressed by terminating the numeric field by the letter E followed by the power of 10 associated with that number. Some examples are:

1.01E-3 means 1.01×10^{-3}
 234E+10 means 2.34×10^{12}
 6.2E21 means 6.2×10^{21} .

Note that there are no spaces in the numbers on the left.

Speakeasy will automatically choose scientific notation for results whenever such notation is deemed more readable. The number of significant figures printed remains correct in all notations. For example,

```
:_sqrt(2e-10)
SQRT(2E-10) = 1.4142E-5
:_exp(30)
EXP(30) = 1.0686E13
:_1.01e10/2.7e6
1.01E10/2.7E6 = 3740.7
:_
```

Variables

It is often useful to save portions of a calculation for use in a subsequent calculation or to break up a large calculation into a number of intermediate stages. Desk calculators frequently have a few memory locations for such intermediate information. Speakeasy allows you to define variables to save such information.

In Speakeasy, a variable is defined by putting its name on the left-hand side of an equal sign; the right-hand side contains the defining expression

```
:_x=3
:_y=4*x+6
:_print x,y
X = 3 Y = 18
:_
```

A variable may be given any name up to eight characters in length; the name must start with a letter and may consist of any combination of letters and numbers but no other symbols and no space. If more than eight characters are used, those beyond the eighth are ignored.

Examples of acceptable variable names are:

ALFIE X9 HELLO A1B2C3

Some variable names that are unacceptable, for the stated reasons are

2BY4	First character is not a letter
TWO/FOUR	Imbedded special character
TWO 4	Imbedded space.

In Speakeasy, the equal sign assigns a value to a variable rather than indicating an equation in a strict mathematical sense. For example,

$X = X+1$

is an impossible mathematical equality but is an entirely legitimate variable definition (actually a redefinition) in Speakeasy. Conversely,

$X+Y=3$ and $HEIGHT*WIDTH=AREA$

are meaningless Speakeasy expressions. Briefly, in Speakeasy, an equal sign says (1) compute the value of the expression on the right; then (2) assign that value to the (single) variable whose name appears on the left.

In context, $X=X+1$ might appear as

```
:_x=3
:_x=x+1
:_print x
X = 4
:_
```

Once defined, a variable may be used freely in subsequent expressions in place of its numerical value

```
:_x=2/3
:_sx=sin(x)
:_cx=cos(x)
:_tx=tan(x)
:_sinbycos=sx/cx
:_maybeone=sx**2 + cx**2
:_print sx, cx, maybeone
SX = .61837 CX = .78589 MAYBEONE = 1
:_print tx, sinbycos
TX = .78684 SINBYCOS = .78684
:_
```

Note that all variables that appear on the right-hand side of an equal sign must have previously appeared on the left-hand side in an earlier statement.

In the preceding example, variables have been printed by use of the PRINT command. A single variable may also be displayed by typing its name. This is an implied print facility. For example,

```

:_maybeone
MAYBEONE = 1
:_x
X = .66667
:_

```

The implied print facility makes it possible to have a discourse with Speakeasy. For example you may ask it questions and receive a legible response, as in the following session.

```

:_angles in degrees
:_sin(30)
SIN(30) = .5
:_cos(30)
COS(30) = .86603
:_sin(30)**2 + cos(30)**2
SIN(30)**2 + COS(30)**2 = 1
:_x=sin(30)
:_y=cos(30)
:_x**2 + y**2
X**2 + Y**2 = 1
:_

```

A fairly sophisticated desk calculator is now at your fingertips; you may define large numbers of variables and use them in evaluating lengthy expressions. You also have at your disposal the information of many mathematical tables in the form of logarithms, trigonometric functions, square roots, etc. Your skills in exploiting these capabilities should now be exercised; it should take you only a short time to learn to do real calculations.

Operations with Arrays of Numbers

The ability to operate on sets or arrays of numbers is a very powerful feature of Speakeasy. So far, only operations with single numbers or single-valued variables have been discussed. As will be seen, this array-processing capability provides the means for carrying out many calculations in parallel. For example,

```

:_x= 0,1,2,3,4

```

defines the variable X to be the integers 0 through 4. Operations on X may be carried out in the same manner as before. The statements

```

:_y=2*x + 1
:_z=x*x + 1

```

define two five-element objects Y and Z.

```

:_print x, y, z
  X (A 5 COMPONENT ARRAY)
    0 1 2 3 4
  Y (A 5 COMPONENT ARRAY)
    1 3 5 7 9
  Z (A 5 COMPONENT ARRAY)
    1 2 5 10 17
:_tabulate x,y,z
  X Y Z
  0 1 1
  1 3 2
  2 5 5
  3 7 10
  4 9 17
:_

```

The use of the TABULATE command, illustrated in the last input statement, provides an alternative form for printing results. It is particularly useful for displaying several arrays whose corresponding elements are related.

The ability to carry out calculations on sets of numbers is made even easier by providing you with a simple way of defining such sets. In the example above, the elements of the original array X were defined explicitly. In Speakeasy several alternative ways are provided for defining specific sets. For example,

```

:_i=integers(1,17)
:_print i
  I (A 17 COMPONENT ARRAY)
    1 2 3 4 5 6 7 8 9 10 11 12 13 14
    15 16 17
:_

```

defines I to be the integers from 1 through 17, and prints it. If you wished to have only the even integers from 4 through 40 you could obtain this with a statement either of the form

```

:_i=2*integers(2,20)
:_print i
  I (A 19 COMPONENT ARRAY)
    4 6 8 10 12 14 16 18 20 22 24 26 28 30
    32 34 36 38 40
:_

```

or of the form

```

:_i=integers(4,40,2)
:_print i
  1 (A 19 COMPONENT ARRAY)
  4   6   8   10  12  14  16  18  20  22  24  26  28  30
 32  34  36  38  40
:_

```

The word INTEGERS is used to define an array whose numbers are all integers. In using this you specify the limits of the values that you want. If you wish, you may specify the number of integers to be skipped between successive members of the array by supplying it in the definition.

In a similar way, Speakeasy allows you to specify an array of numbers starting at any value (not necessarily an integer) and going to another—normally with a specified step size between successive elements. Thus

```

:_x=grid(0.2,1.6,.1)
:_print x
  X (A 15 COMPONENT ARRAY)
  .2   .3   .4   .5   .6   .7   .8   .9   1   1.1  1.2
 1.3  1.4  1.5  1.6
:_

```

defines X to be the values starting at 0.2 and going to 1.6 in steps of 0.1. (If the step size is omitted in your definition of the grid, then 101 equally spaced values are generated. This provides enough points to represent most functions more than adequately for most calculations.[†])

At this stage you might assume that a larger number of other words would be provided to supply you with the means of defining arrays with a variety of spacings—e.g., logarithmic spacing, etc. However, this is unnecessary since the Speakeasy language itself provides those facilities. If, for example, you wished to have an array of points that consisted of all the square roots of the integers from 1 to 100, you could generate such an array by the statement

```

:_x=sqrt(integers(1,100))

```

At this time you might try generating a few arrays such as this to see how easily this is done.

By making use of these simple ways of defining sets and drawing on the simple functions already described, it is very easy to make a table of trigonometric functions with the following statements.

[†] Notice that the print statement did not contain parentheses. Speakeasy has been designed to be tolerant of such syntactical oversights.

```

:_angles in degrees
:_theta=grid(0,90,10)
:_sine=sin(theta)
:_cosine=cos(theta)
:_tangent=tan(theta)
:_tabulate theta sine cosine tangent
  THETA SINE      COSINE  TANGENT
    0     0        1       0
   10   .17365   .98481   .17633
   20   .34202   .93969   .36397
   30    .5     .86603   .57735
   40   .64279   .76604   .8391
   50   .76604   .64279   1.1918
   60   .86603    .5     1.7321
   70   .93969   .34202   2.7475
   80   .98481   .17365   5.6713
   90    1       0       INF.
:_

```

In defining arrays of numbers, one can use the general defining word ARRAY. Thus,

```
X=ARRAY(N:list)
```

defines X to be an array with N elements. The colon marks the end of this definition of the structure of X; the elements of the "list" that follows are the successive components of X. If an array is used in this list, then the elements of that array are used as successive elements in the new array. One should be careful that the number of elements contained in the list fills the array being defined. If the array has more elements than are supplied in the list, the ones left unspecified are set equal to zero. The use of arrays in the list of elements of a new array is illustrated by

```

:_i1=1,2,3,4
:_i2=6,7,8,9
:_i=array(10:i1,5,i2,17)
:_print i
  1 (A 10 COMPONENT ARRAY)
  1  2  3  4  5  6  7  8  9  17
:_

```

Speakeasy provides a number of operations that are designed specifically to act on arrays. Among these are

SUM(X)	to obtain the sum of the elements of X
SUMSQ(X)	to obtain the sum of the squares of the elements of X
MIN(X)	to find the minimum element of X
MAX(X)	to find the maximum element of X

NOELS(X) to find the number of elements of X

AVERAGE(X) to obtain the average value of the elements of X:

these operations are illustrated in the following Speakeasy statements.

```

:_x=integers(-2,4)
:_x
X (A 7 COMPONENT ARRAY)
-2 -1 0 1 2 3 4
:_sum(x)
SUM(X) = 7
:_sumsq(x)
SUMSQ(X) = 35
:_min(x)
MIN(X) = -2
:_max(x)
MAX(X) = 4
:_noels(x)
NOELS(X) = 7
:_average(x)
AVERAGE(X) = 1
:_

```

These operations may be used in a longer expression. For example, the root-mean-square (RMS) deviation from the average value may be calculated as

```

:_rms=sqrt(sumsq(x-average(x))/noels(x))
:_rms
RMS = 2
:_

```

Two-Dimensional Arrays

In the previous discussion of variables that are sets of numbers, we dealt with structures called one-dimensional arrays. In some circumstances it is easier conceptually, or even necessary, to deal with two-dimensional arrays, those that have both rows and columns. The statement

```

:_atwo=array(2,3: 11,12,13,21,22,23)

```

defines a two-dimensional array with two rows and three columns.

```

:_atwo
ATWO (A 2 BY 3 ARRAY)
11 12 13
21 22 23
:_

```

The two numbers preceding the colon define the number of rows and the numbers of columns, respectively. Following the colon, the elements of the array are specified in order, reading along successive rows. Thus,

```
:_apart=array(2,2:11,12,21)
:_apart
  APART (A 2 BY 2 ARRAY)
    11  12
    21   0
:_
```

As in the case of one-dimensional arrays, structured objects may be used in defining the elements of a two-dimensional array. The elements of the structured object are used as elements in the array being defined. In the case of two-dimensional objects, these elements are extracted and used row by row.

```
:_one=array(3:1.5,2,2.5)
:_two=array(3:3,3.5,4)
:_a=array(2,3:one,two)
:_print a
  A (A 2 BY 3 ARRAY)
    1.5  2   2.5
    3    3.5  4
:_
```

Two-dimensional arrays may be used in mathematical expressions in the same manner as any other variable. Operations on arrays take place element by element. All two-dimensional arrays that appear in a given arithmetic expression must have the same structure, i.e., the same number of rows and the same number of columns. Several examples dealing with two-dimensional arrays are shown below.

```
:_x=array(2,2:8,6,4,2)
:_x
  X (A 2 BY 2 ARRAY)
    8  6
    4  2
:_z=x/2
:_z
  Z (A 2 BY 2 ARRAY)
    4  3
    2  1
:_1/z
1/Z (A 2 BY 2 ARRAY)
    .25  .33333
    .5   1
:_
```


Operations on Two-Dimensional Arrays

The operators SUM, SUMSQ, MIN, MAX, NOELS, and AVERAGE that were discussed in the context of one-dimensional arrays may also operate on two-dimensional arrays. In addition, there are several operators that are designed specifically to operate on two-dimensional objects. Some of these are described below.

SUMROWS(X) adds the elements in each row of the two-dimensional array X. The result is a one-dimensional array, each of whose elements is the sum of the elements in the corresponding row of X. Symbolically SUMROWS(X) is $\sum_j X_{ij}$.

SUMSQROWS(X) adds the squares of the elements in each row of two-dimensional array X. The result is a one-dimensional array. Symbolically SUMSQROWS(X) is $\sum_j (X_{ij})^2$.

PRODROWS(X) forms the product of the elements in each row of the two-dimensional array X. The result is a one-dimensional array. Symbolically PRODROWS(X) is $\prod_j X_{ij}$.

SUMCOLS(X) adds the elements in each column of the two-dimensional array X. The result is a one-dimensional array, each of whose elements is the sum of the elements in the corresponding columns of X. Symbolically SUMCOLS(X) is $\sum_i X_{ij}$.

SUMSQCOLS(X) adds the sum of the squares of the elements in each column of the two-dimensional array X. The result is a one-dimensional array, each of whose elements is the sum of the square of the elements in the corresponding column of X. Symbolically SUMSQCOLS(X) is $\sum_i (X_{ij})^2$.

PRODCOLS(X) forms the product of the elements of each column of the two-dimensional array X. The result is a one-dimensional array. Symbolically PRODCOLS(X) is $\prod_i X_{ij}$.

NOROWS(X)	returns the number of rows of X.
NOCOLS(X)	returns the number of columns of X.
COLMAX(X)	specifies the column with the largest element.
COLMIN(X)	specifies the column with the smallest element.
ROWMAX(X)	specifies the row with the largest element.
ROWMIN(X)	specifies the row with the smallest element.
MAXOFCOL(X)	creates a one-dimensional array whose elements are the largest elements of the corresponding columns of X.
MINOFCOL(X)	creates a one-dimensional array whose elements are the smallest elements of the corresponding columns of X.
MAXOFROW(X)	creates a one-dimensional array whose elements are the largest elements of the corresponding rows of X.
MINOFROW(X)	creates a one-dimensional array whose elements are the smallest elements of the corresponding rows of X.

Examples of some of these operations are given in the following Speakeasy statements.

```

:_x=array(2,3: integers(1,6))
:_x
  X (A 2 BY 3 ARRAY)
  1  2  3
  4  5  6
:_sumrows(x)
SUMROWS(X) (A 2 COMPONENT ARRAY)
  6  15
:_sumcols(x)
SUMCOLS(X) (A 3 COMPONENT ARRAY)
  5  7  9
:_sumsqrows(x)
SUMSQROWS(X) (A 2 COMPONENT ARRAY)
  14  77
:_prodcols(x)
PRODCOLS(X) (A 3 COMPONENT ARRAY)
  4  10  18
:_noels(x)
NOELS(X) = 6
:_average(x)
AVERAGE(X) = 3.5
:_norows(x)
NOROWS(X) = 2
:_

```

Operations on Parts of Objects

So far, we have learned how to perform parallel operations on sets of numbers by use of the array features of Speakeasy. It is often useful to operate selectively on parts of such sets. This section will illustrate a variety of ways of performing such operations.

Let us define a set of numbers

```
:_set=array(7:3,5,6,3,4,2,4)
:_
```

The *I*th element of the array SET is referenced by the statement SET(*I*). The quantity *I* is called an index. For the one-dimensional array SET, the first element is SET(1), the second is SET(2), etc. Thus,

```
:_set(1)
SET(1) = 3
:_set(5)
SET(5) = 4
:_
```

Here SET(5) refers to the fifth element in the array SET. The fifth element could equally well be chosen by specifying a new variable. Thus,

```
:_i=5; set(i)
SET(i) = 4
:_
```

selects the same element. (Note the use of a semicolon to separate two statements entered on the same line.)

Several elements of an array may be selected by use of an array-valued index. The sequence of operations

```
:_i=1,3,7,5
:_subset=set(i)
:_
```

defines a new array SUBSET that consists of the 1st, 3rd, 5th, and 7th elements of SET.

The ability to work with structured indices, those that refer collectively to several elements of an object, enables you to operate on parts of defined objects in much the same way as you work on the entire object. All of the elements of SET may be summed by

```
:_sum(set)
SUM(SET) = 27
:_
```

The 1st, 3rd, 5th, and 7th elements may be summed by

```
:_sum(subset)
SUM(SUBSET) = 17
:_
```

Structured indices are usually generated on the basis of some condition involving the elements of an object. This condition may involve concurrent relationships among the elements of several objects. The manner in which interrelations among objects may be tested and utilized is the subject of the next chapter.

3. RELATIONAL, LOGICAL, AND CONDITIONAL OPERATORS

Relational Operators

In many applications, it is essential to know the relationship between two variables. Is one variable equal to another? Is it larger or smaller? The answers to such questions are provided by Speakeasy's relational operators. They connect two variables of identical structures or any variable and a scalar. They operate on an element-by-element basis and result in a true or false indication signaled by a "1" or a "0", respectively. In Speakeasy, the relational operators are represented by keywords enclosed between periods which are listed below.

.EQ.	equal to
.NE.	not equal to
.GT.	greater than
.LT.	less than
.GE.	greater than or equal to
.LE.	less than or equal to

A comparison of two scalar variables is illustrated by

```
:_x=2; y=3; x.eq.y
X.EQ.Y = 0
:_x.lt.y
X.LT.Y = 1
:_x.gt.1
X.GT.1 = 1
:_
```

A set of numbers (an array) may be compared with a scalar. For example,

```

:_a=0,1,2,3
:_a.eq.x
A.EQ.X (A 4 COMPONENT ARRAY)
  0  0  1  0
:_a.gt.1.25
A.GT.1.25 (A 4 COMPONENT ARRAY)
  0  0  1  1
:_

```

The relationships between the elements of two sets may also be determined in a straightforward manner. For example,

```

:_b=2,1,0,-1
:_aeqb=a.eq.b; aneb=a.ne.b; altb=a.lt.b
:_agtb=a.gt.b; ageb=a.ge.b; aleb=a.le.b
:_tabulate a,b,aeqb,aneb,agtb,altb,ageb,aleb
  A  B  AEOB  ANEB  AGTB  ALTB  AGEB  ALEB
  0  2  0    1    0    1    0    1
  1  1  1    0    0    0    1    1
  2  0  0    1    1    0    1    0
  3 -1  0    1    1    0    1    0
:_

```

Logical Operators

Three logical operators are available in Speakeasy, namely

```

.NOT.      logical not
.AND.      logical sum
.OR.       inclusive or

```

These operators consider any nonzero value to be "true" and a zero value to signify "false." Using the variables defined above, we find

```

:_not.a
.NOT.A (A 4 COMPONENT ARRAY)
  1  0  0  0
:_a.or.b
A.OR.B (A 4 COMPONENT ARRAY)
  1  1  1  1
:_a.and.b
A.AND.B (A 4 COMPONENT ARRAY)
  0  1  0  1
:_a.gt.1.and.b.le.0
A.GT.1.AND.B.LE.0 (A 4 COMPONENT ARRAY)
  0  0  1  1
:_

```

It is also permissible to mix logical, relational, and arithmetic expressions in a single statement. In such cases, the order of operations is

- (1) arithmetic operations,
- (2) relational operations,
- (3) logical .NOT. operations, and
- (4) logical .OR. and .AND. operations.

Within each step of this hierarchy, the evaluation proceeds from left to right. The expression

.NOT. A. AND. B+C. LT. D. OR. E. EQ. F*G+H

is equivalent to

((.NOT. A) .AND. ((B+C). LT. D)) .OR. (E. EQ. (F*G+H)).

In keeping with the Speakeasy approach of dealing with objects as a whole, the LOCS function enables the conditional selection of elements of a set. LOCS operates on a one-dimensional object and provides the element numbers (indices) corresponding to the nonzero ("true") elements of the array in question. The result of the operation is normally used as a structured index.

If X is the set of numbers

```
:_x=-1,-1,.5,0,3,1.27,-2,5,10,3.0,-5.2,8.7
:_
```

LOCS(X) will return the indices of the nonzero elements of the set. Thus,

```
:_locs(x)
LOCS(X) (A 11 COMPONENT ARRAY)
 1   2   3   5   6   7   8   9   10  11  12
:_
```

This could be used in eliminating all zero values in a set of numbers. For example, the following statements removes all zero elements from the array X.

```
:_x=x(locs(x))
:_x
X (A 11 COMPONENT ARRAY)
-1   -1   .5   3   1.27 -2   5   10   3
-5.2   8.7
:_
```

Similarly, we may filter those elements whose absolute value exceeds 1 by the statement

```
:_x=x(locs(abs(x).le.1))
:_x
X (A 3 COMPONENT ARRAY)
-1  -1  .5
:_
```

Conditional Statements

Often, you will want certain operations performed only if specific variables or relations among several variables satisfy certain conditions. Speakeasy's conditional statements consist of two parts: (1) a test that utilizes relational operators, and (2) a statement that is carried out only when certain test criteria are met. The conditional statements in Speakeasy are IF and WHERE.

The form of the IF statement is

IF (scalar-valued expression) STATEMENT.

If the scalar-valued expression is nonzero (true), the STATEMENT is carried out; if the expression is zero (false), the statement is ignored. For example,

```
:_x=2
:_y=3
:_if(x .gt. 3) y=y+1
:_y
Y = 3
:_if(x .eq. 2) y=y+2
:_y
Y = 5
```

The second type of conditional statement, the WHERE statement, operates on structured objects. In particular, it provides for masking operations. Its form is

WHERE (logical expression) assignment statement.

The logical expression and the result of the assignment statement must be of identical structure. The logical expression serves as a mask in the sense that an element of the assignment statement is replaced only at the locations where the corresponding element of the mask is true (nonzero).

```
:_autoprint
:_aa=array(2,3:-1,3,1,0,4,-2)
AA (A 2 BY 3 ARRAY)
-1 3 1
0 4 -2
:_where(aa.gt.0)cc=2*aa
CC (A 2 BY 3 ARRAY)
0 6 2
0 8 0
:_where(aa.gt.-1 .and. aa.le.3) bb=aa+1
BB (A 2 BY 3 ARRAY)
0 4 2
1 0 0
:_
```

In this last example, we used the automatic print feature of Speakeasy. AUTOPRINT asks that all new or redefined items be displayed. This is a useful option when you are interested in examining intermediate steps in a calculation. This feature is turned off by entering the command ENDAUTOPRINT.

4. DIAGNOSTICS AND ERROR DETECTION

The consequences of possible mistakes such as misspellings, etc. have not been mentioned thus far. If, as was originally suggested, you have been using TSO Speakeasy as a learning aid while reading this book, then you are no doubt already familiar with some of the error-detection features built into Speakeasy. This section will describe some of these features and others that you may not have encountered yet.

Several types of errors can occur while using Speakeasy. Many of these can be detected by Speakeasy and will result in an error message informing you of the detected error. These error messages indicate that Speakeasy was not able to carry out your request. The messages are informative, in the sense that they attempt to explain the reason why the request could not be satisfied. Since Speakeasy does not know the problem you are solving, such guesses at the cause of the error may be wrong. In any event, the specific request causing an error response is impossible to process and must be re-entered correctly.

Syntactical errors are usually the result of mistyping. If you type in a sentence of symbols that are not a meaningful instruction, Speakeasy will respond by telling you that it was unable to carry out your request and will indicate why. You should retype the proper statement to proceed with the calculation. For example,

```
:_2+3.5*/8
```

is meaningless and will elicit the response

```
IN LINE " 2+3.5*/8 " DOUBLE OP.
:_
```

This means that two operators * and / occurred together—an illegal syntactical form in Speakeasy.

If you notice an error before pressing the RETURN key or while typing, you may backspace over the part that is wrong and begin typing the correct information from that point on. If you had noticed the mistake in the previous example you could have corrected it as follows (← indicates backspace)


```

:_2+3.5*88
      ^^^
      /8

```

which would be equivalent to

```
:_2+3.5/8
```

If the entire line is in error, you may cancel it; i. e., the entire line will be ignored if you press the attention key. You should then re-enter the proper input.

Some errors are due to logical inconsistencies. The use of undefined variables, carrying out specific operations of variables for which such operations are meaningless, and operations between variables that are of incompatible structure are all logical errors that can be detected by Speakeasy. Speakeasy is designed to test for logical consistency whenever it is doing your calculations. It also tells you about any logical errors that it finds. If, for example, you enter

```
:_2*house+27.5
```

and HOUSE had not previously been defined, Speakeasy will respond

```
HOUSE IS NOT DEFINED IN LINE " 2*HOUSE+27.5  "
:_
```

It is therefore impossible to use undefined objects in calculations. You are informed of such an attempt immediately and you can make the necessary corrections and re-enter your request.

Similarly if X is an array and you attempt to use elements of X not contained within X you will be informed of this. For example,

```
:_x=1,2,3,4,5
:_x(7)*5
X IN LINE " X(7)*5  "  INDEX OUTSIDE BOUNDS.
:_
```

Error messages such as these are so-called execution time messages. They represent errors detected by Speakeasy in attempting to carry out requests that appear meaningful but that are in fact logically unsound. As you proceed through a calculation, you will often carry out operations involving several defined variables. Each such operation results in many checks concerned with internal logic consistency. If for instance you type

```
:_a=1,2,3,4
:_b=1,2,3
:_c=a+b
```

you will be informed that

```
IN LINE " C=A+B " A AND B ARE INCOMPATIBLE
: _
```

You should realize that the monitoring of the logic of your request provides assurances as to the integrity of your calculations. Any result that you obtain from Speakeasy is a correct answer to a problem that has undergone extensive tests for logical consistency. Of course, it is still possible to make mistakes while using Speakeasy. A properly formulated description of the wrong problem will produce results. The results will be correct for the stated problem but will probably be of little use to you.

However, Speakeasy does provide many tools to help you in correctly formulating your problem. AUTOPRINT is a facility (described earlier) that enables you to watch the calculation being carried out as it proceeds. All of the intermediate quantities developed during the calculation are available to you. While using Speakeasy, you can easily carry out auxiliary calculations as a means of checking your steps. This approach to carrying out a calculation is one that will become natural to you as you begin to feel confident about Speakeasy and as you learn about the powerful operations available to you while using Speakeasy.

5. MATRIX AND VECTOR OPERATIONS

Simultaneous Equations

In the preceding sections we have dealt with scalar objects, i. e., objects with one element and with one- and two-dimensional arrays of elements. Operations carried out with such objects provide the capabilities for producing tables of specific functions quickly. In all of the algebraic rules demonstrated so far, the elements of objects were correlated with each other in a one-for-one relationship in a rather direct way. There is an alternative mathematical scheme for dealing with arrays of numbers that is also extensively used in many fields. If you are already familiar with matrix algebra, you will find the following sections a rather rapid review of some applications. It also serves as a brief introduction to the subject for those of you unacquainted with the field.

Matrices and vectors are two- and one-dimensional sets of numbers that have special rules for multiplication and division. One very useful application of matrices and vectors is the solution of simultaneous linear equations. The set of equations

$$2x_1 + 3x_2 = 8$$

$$5x_1 - 2x_2 = 1$$

is written in matrix-vector notation as

$$Mx = y$$

and has the "obvious" solution

$$x = (1/M)y,$$

where the matrix M is the object

$$M = \begin{bmatrix} 2 & 3 \\ 5 & -2 \end{bmatrix}$$

and the vector y is

$$y = \begin{bmatrix} 8 \\ 1 \end{bmatrix}.$$

In Speakeasy, M and Y are defined by the statements

```
:_m=matrix(2,2:2,3,5,-2)
:_y=vector(2:8,1)
```

The first statement defines M as a matrix with 2 rows and 2 columns whose elements, entered row-wise, are 2, 3, 5, and -2. The colon separates the size specification from the list of elements. The next statement defines the variable Y as a vector with two components, 8 and 1, with the colon serving a similar purpose. The result $X = (1/M)Y$ is obtained by the simple expression

```
:_x=(1/m)*y
:_x
X (A VECTOR WITH 2 COMPONENTS)
1 2
:_
```

This says the answer is $x_1 = 1$ and $x_2 = 2$. You can probably solve two simultaneous equations in two unknowns as quickly as the computer. With three or more equations, the computer formulation is no more complicated yet is almost certainly quicker. For example, a set of four equations in

four unknowns would take you at least five minutes by hand, but the Speakeasy result is almost immediate. To solve for the four unknowns in the four equations

$$1x_1 + 2x_2 + 3x_3 + 4x_4 = 20$$

$$4x_1 + 3x_2 + 2x_3 + 1x_4 = 10$$

$$1x_1 + 3x_2 + 5x_3 + 7x_4 = 34$$

$$7x_1 + 5x_2 + 3x_3 + 1x_4 = 14$$

the procedure is to enter

```
:_m=matrix(4,4: 1,2,3,4, 4,3,2,1, 1,3,5,7, 7,5,3,1)
:_y=vector(4: 20,10,34,14)
:_x=(1/m)*y
```

and asking for X

```
:_x
X (A VECTOR WITH 4 COMPONENTS)
-1      1     -13     12.25
:_
```

That is, the result is $x_1 = -1$, $x_2 = 1$, $x_3 = -13$, and $x_4 = 12.25$.

Sometimes the specification of elements in a matrix will exceed the number of characters available in a single line of your console. This apparent difficulty is easily circumvented. Speakeasy allows structured objects to be used in defining the elements of an object. In such cases, the elements of the structured objects are used as successive elements in the definition. The 4×4 matrix in the previous problem may also be defined by writing

```
:_row1=1,2,3,4; row2=4,3,2,1; row3=1,3,5,7;row4=7,5,3,1
:_otherm=matrix(4,4:row1,row2,row3,row4)
:_print otherm
OTHERM (A 4 BY 4 MATRIX)
 1  2  3  4
 4  3  2  1
 1  3  5  7
 7  5  3  1
:_
```

There is nothing special about the designations ROW1, ROW2, ...; these are merely one-dimensional arrays of the type discussed earlier. It is not even necessary to break the element list into rows. All that matters is the order of the elements, in whatever form they may appear.

Algebraic Details

This section deals with formalities. First there is a brief review of the definitions and conventions of matrix algebra and their representation in Speakeasy. These matrix-vector operations and vector-vector operations are defined with examples of their formulation in Speakeasy.

Matrices

A matrix M is a two-dimensional set of numbers. In Speakeasy, a matrix is defined by the statement

$M = \text{MATRIX}(\text{NROWS}, \text{NCOLS}; \text{element list}).$

The numbers of rows and columns precede the colon and the element list follows it. This element list defines the matrix row-wise; it may consist of scalars or some collection of structured objects such as one-dimensional arrays. If there are fewer numbers in the element list than are required to fill the matrix, the remaining elements are set equal to zero.

The element M_{ij} —i. e., the element in the i th row and j th column—is represented as $M(I, J)$ in Speakeasy. The product C of two matrices A and B is given by

$$C_{ij} = \sum_k A_{ik} B_{kj}.$$

In Speakeasy, this product is written as

$$C = A * B.$$

The inverse of a square matrix M is that matrix N such that

$$MN = 1,$$

where 1 represents the appropriate identity or unit matrix. In Speakeasy, this matrix inverse is easily found by the statement

$$N = 1/M.$$

The most common matrix operations, their formal definitions, and their Speakeasy representations are summarized in tabular form as follows.

<u>Operation</u>	<u>Formal Definition</u>	<u>Speakeasy</u>
Matrix product $C=AB$	$C_{ij} = \sum_k A_{ik} B_{kj}$	$C=A*B$
Matrix sum $C=A+B$	$C_{ij} = A_{ij} + B_{ij}$	$C=A+B$
Matrix inverse $AB=1$ $B=1/A$	$B_{ij} = F_{ij}/\det A $ where F_{ij} is the transposed matrix of cofactors of A	$B=1/A$
Matrix scalar sum $A=B+S$	$A_{ij} = B_{ij} + S\delta_{ij}$	$A=B+S$
Matrix-scalar product	$A_{ij} = SB_{ij}$	$A=S*B$

Some of these matrix operations are illustrated in the examples that follow.

We may define a matrix easily with the statement

```
:_m=matrix(3,3: integers(4,12) )
:_m
  M (A 3 BY 3 MATRIX)
    4    5    6
    7    8    9
   10   11   12
:_
```

To add a scalar to a matrix, the scalar is added to each diagonal element. Thus,

```
:_m+100
M+100 (A 3 BY 3 MATRIX)
  104    5    6
    7   108    9
   10   11   112
:_
```

Predefined sets of numbers may be inserted as elements of a matrix.

```
:_x=3,44,4
:_n=matrix(3,3:30,1,1/2,x,5,6,50)
:_n
  N (A 3 BY 3 MATRIX)
    30    1    .5
    3   44    4
    5    6   50
:_
```

A scalar times a matrix results in the scalar multiplying every element of the matrix. Thus $2n$ would be

```
:_nn=2*n
:_nn
  NN (A 3 BY 3 MATRIX)
    60    2    1
    6    88    8
    10   12   100
:_
```

The matrix sum is illustrated by

```
:_mn=m-nn
:_mn
  MN (A 3 BY 3 MATRIX)
   -56    3    5
    1  -80    1
    0   -1  -88
:_
```

A matrix C which is the product of two matrices is defined by the statements

```
:_m1=matrix(3,2:1/2,2,3,1,-1/2,2)
:_m1
  M1 (A 3 BY 2 MATRIX)
    .5    2
    3     1
   -.5    2
:_c=mn*m1
:_c
  C (A 3 BY 2 MATRIX)
  -21.5  -99
  -240   -76
    41   -177
:_
```

Vectors

A vector V is a set of numbers that acts as a row matrix, a column matrix, or a diagonal matrix according to context. In Speakeasy, a vector is defined by the statement

```
V = VECTOR(NOELS: element list).
```

NOELS is the number of elements. It precedes the colon. The element list may consist of scalars or structured objects. If there are fewer elements in the element list than the number specified, the remaining elements

are set equal to zero. If desired, the number of elements may be omitted and only the element list given. In this case, a vector is defined by the statement

V = VECTOR(: element list).

The i th element V_i of a vector is written in Speakeasy as V(I). Some of the vector-vector and matrix-vector operations are summarized as follows.

<u>Operation</u>	<u>Formal Definition</u>	<u>Speakeasy</u>
Inner product (scalar product)	$S = \sum_i X_i Y_i$	S=X*Y
Outer product (tensor product)	$M_{ij} = X_i Y_j$	M=X**Y
Vector sum	$Z_i = X_i + Y_i$	Z=X+Y
Matrix-vector product	$Z_i = \sum_k M_{ik} X_k$	Z=M*X
Vector-matrix product	$Z_j = \sum_k X_k M_{kj}$	Z=X*M

The simplicity with which matrix-vector operations are formulated in Speakeasy is illustrated in the following examples. (Here we begin by issuing the command AUTOPRINT. This command instructs Speakeasy to print every variable as it is defined. A restricted form AUTOPRINT(A,B,C,...) asks Speakeasy to print the designated variables as they are defined or changed.) For example,

```
:_autoprnt
:_v=vector(3:1,2,3)
  V (A VECTOR WITH 3 COMPONENTS)
  1  2  3
:_v0=vector(3:)
  V0 (A VECTOR WITH 3 COMPONENTS)
  0  0  0
:_w=vector(3:2,4)
  W (A VECTOR WITH 3 COMPONENTS)
  2  4  0
:_
```

The inner and outer products are


```

:_inner=v*w
INNER = 10
:_outer=v**w
OUTER (A 3 BY 3 MATRIX)
  2   4   0
  4   8   0
  6  12   0
:_

```

Taking the product (or sum) of a scalar and a vector is an element-by-element operation. Thus

```

:_2*v + 10
2*v + 10 (A VECTOR WITH 3 COMPONENTS)
 12  14  16
:_

```

The product of a matrix M and a vector V or W depends on whether the vector is multiplied from the left or the right of the matrix. If the vector appears on the right-hand side of a matrix, as in M*V, it behaves as a row vector. Thus

```

:_m=matrix(3,3: integers(4,12))
M (A 3 BY 3 MATRIX)
  4   5   6
  7   8   9
 10  11  12
:_m*v
M*v (A VECTOR WITH 3 COMPONENTS)
 32  50  68
:_

```

and

```

:_w*m
W*M (A VECTOR WITH 3 COMPONENTS)
 36  42  48
:_

```

A matrix N is created by taking the outer product of two vectors. As this results in two matrices, M and N, being currently defined, the product of two matrices is illustrated at this time.

```

:_n=v**w
N (A 3 BY 3 MATRIX)
  2   4   0
  4   8   0
  6  12   0

:_m*n
M*N (A 3 BY 3 MATRIX)
 64  128  0
100  200  0
136  272  0

```

Operations on Matrices

In addition to its matrix-algebra capabilities, Speakeasy provides a full complement of standard operations on matrices. Among these are operators that find transposes, determinants, traces, and inverses or that select the diagonal elements of a matrix. With the exception of the transpose operation, all are restricted to operate on square matrices.

```
:_m=matrix(3,3:1,3,5,4,2)
:_transpose(m)
TRANPOSE(M) (A 3 BY 3 MATRIX)
  1  4  0
  3  2  0
  5  0  0
:_m=m+transpose(m)
:_m
M (A 3 BY 3 MATRIX)
  2  7  5
  7  4  0
  5  0  0
:_
```

The determinant of M, its trace, and its inverse are found.

```
:_det(m)
DET(M) = -100
:_trace(m)
TRACE(M) = 6
:_inverse(m)
INVERSE(M) (A 3 BY 3 MATRIX)
-1.3878E-17  1.3878E-17  .2
  0          .25      -.35
  .2         -.35      .41
:_
```

Note that INVERSE(M) is synonymous with 1/M. Thus,

```
:_1/m
1/M (A 3 BY 3 MATRIX)
-1.3878E-17  1.3878E-17  .2
  0          .25      -.35
  .2         -.35      .41
:_
```

The diagonal elements are given by

```
:_diagels(m)
DIAGELS(M) (A 3 COMPONENT ARRAY)
  2  4  0
:_
```

The operations discussed earlier for two-dimensional arrays may also be applied to matrices. For example,

```
:_sumrows(m)
SUMROWS(M) (A VECTOR WITH 3 COMPONENTS)
  14  11  5
:_
```

The eigenvalues and eigenvectors of a matrix may be easily computed. The eigenvalues of matrix M, in descending order, are

```
:_eigenvalues(m)
EIGENVALUES(M) (A VECTOR WITH 3 COMPONENTS)
-6.5063  1.3816  11.125
:_
```

The set of eigenvectors of M is a matrix whose columns are the eigenvectors of M. The ith column is the eigenvector corresponding to the ith eigenvalue returned by the EIGENVALUES(M) operation. Thus,

```
:_eigenvectors(m)
EIGENVECTORS(M) (A 3 BY 3 MATRIX)
-.70109  .21696  .67927
 .46711 -.58001  .66738
 .53878  .78518  .3053
:_
```

A given eigenvector of this matrix may be isolated as follows.

```
:_evecs=answer
:_vec1 = evecs(,1)
:_vec2 = evecs(,2)
:_print vec1 vec2
  VEC1 (A VECTOR WITH 3 COMPONENTS)
-.70109  .46711  .53878
  VEC2 (A VECTOR WITH 3 COMPONENTS)
 .21696 -.58001  .78518
:_
```

6. CALCULUS AND SPEAKEASY

Speakeasy provides many of the common numerical approximations to operations in differential and integral calculus. Before detailing the features available, it is important to stress that there are inherent risks in such numerical approximations and that you cannot blindly use such approximations with assurance of correct results. Even so, numerical procedures such as those available in Speakeasy are extremely valuable

and, when properly understood, will lead to meaningful answers quickly. TSO Speakeasy is particularly useful when carrying out this type of calculation because you are able to see results immediately and therefore can examine and test the validity of the approximations.

Derivatives

In your introductory course in calculus, you probably learned about the derivative of a function in terms of a limiting process. The derivative is defined as the limit of the ratio of the change of the function to the changes of its independent variable. In Speakeasy, we mimic this original definition and provide the ratio of the change in a function to the change in its associated variable. The increments involved in the limiting process are controlled by the values that you supply for the variable and the function. For example,

```
:_pi=2*acos(0)
:_x=grid(0,pi/2,pi/100)
```

defines X to be a set of points

0, $\pi/100$, $2\pi/100$, ..., $\pi/2$.

Then the function $Y = (X^2 - 1) \sin(X)$ for these values of X is defined by

```
:_y=(x**2-1)*sin(x)
```

This is a fairly sizable number of points (51) so that one can expect reasonably good values for the derivative by numerical techniques. In Speakeasy, the command DERIVATIVE(F:X) means calculate the numerical approximation to dF/dX by using the supplied tabulated values for F and X. (There must be the same number of values for F and X and F is assumed to be the value of the function at the corresponding value of X.) Thus

```
:_dydx=derivative(y:x)
```

supplied the desired derivative. Actually this function can be differentiated analytically and its derivative is

$$\frac{dy}{dx} = 2x \cdot \sin(x) + (x^2 - 1) \cos(x).$$

We can evaluate this in Speakeasy and compare the results, as follows.

```
:_exact=2*x*sin(x)+(x**2-1)*cos(x)
```

```
:_tabulate (x,dydx,exact)
```

X	DYDX	EXACT	X	DYDX	EXACT
0	-1.0023	-1	.81681	.96316	.96304
.031416	-.9954	-.99655	.84823	1.0871	1.087
.062832	-.98505	-.9862	.87965	1.2113	1.2114
.094248	-.96784	-.96898	.91106	1.3355	1.3356
.12566	-.94382	-.94495	.94248	1.4591	1.4593
.15708	-.91306	-.91417	.97389	1.5818	1.582
.1885	-.87565	-.87674	1.0053	1.703	1.7033
.21991	-.83171	-.83278	1.0367	1.8224	1.8228
.25133	-.78135	-.7824	1.0681	1.9395	1.9399
.28274	-.72474	-.72576	1.0996	2.0538	2.0543
.31416	-.66205	-.66303	1.131	2.1649	2.1655
.34558	-.59345	-.5944	1.1624	2.2724	2.273
.37699	-.51916	-.52008	1.1938	2.3757	2.3765
.40841	-.43941	-.44028	1.2252	2.4745	2.4753
.43982	-.35443	-.35526	1.2566	2.5683	2.5692
.47124	-.26449	-.26527	1.2881	2.6567	2.6577
.50265	-.16985	-.17059	1.3195	2.7393	2.7403
.53407	-.070821	-.071503	1.3509	2.8155	2.8166
.56549	.032302	.031673	1.3823	2.8851	2.8863
.5969	.13919	.13862	1.4137	2.9476	2.9488
.62832	.24952	.249	1.4451	3.0026	3.0039
.65973	.36293	.36247	1.4765	3.0497	3.0511
.69115	.47906	.47866	1.508	3.0885	3.09
.72257	.59753	.59721	1.5394	3.1188	3.1203
.75398	.71798	.71771	1.5708	3.1446	3.1416
.7854	.83999	.83979			

```
:_
```

Note that there are errors and that the numerical derivative is not equally reliable at all places. Errors in numerical derivatives will be largest, in general, when the derivative itself is changing most rapidly.

Try carrying out the series of steps shown above with different grids for the initial variable and see if you can improve the approximation for specific ranges of the variable.

Integration

The integral of a function is simply the area under the curve defined by the function. As in the case of the derivative, the simplest definition of an integral is given in a limiting process. It is usually defined as the sum of the areas of trapezoids made by joining successive points along the function, in the limit in which the step size between these points becomes infinitesimal. This limiting process is illustrated graphically in Figs. 1, 2, and 3.

Let us evaluate $\int_{-2}^2 [\frac{1}{2}x^2 + 1] dx$. In other words, compute the area under the curve $\frac{1}{2}x^2 + 1$ between $x = -2$ and $x = +2$. The exact answer is $6 - 2/3$. Figure 1 illustrates the nature of the approximation if two trapezoids

$$I_2 = 8$$

$$I_4 = 7$$

$$I_8 = 6 \frac{3}{4}$$

Figs. 1, 2 and 3

grid X . Quite simply, the value of the INTEGRAL operation $G(I)$ at a grid point $X(I)$ is just the area under the curve F up to the point $X(I)$. Let us illustrate this operation with the following simple example. We will define the independent variable to be a grid from 0 to 10 and evaluate the integral of x^2 over this range.

```
:_x=grid(0,10,.25)
:_g=integral(x**2:x)
```

Let us pick a few points and tabulate the values of G and its correct values $x^3/3$.

```
:_i=locs(fracpart(x).eq.0)
:_tabulate( x(i),g(i),x(i)**3/3)
```

0	0	0
1	.34375	.33333
2	2.6875	2.6667

(3 points) are used. The area of the two trapezoids is 8. Figure 2 shows four trapezoids (5 points). Their areas sum to 7, which is a distinctly better approximation. Figure 3 uses eight trapezoids (9 points) and the sum of the areas of the trapezoids is $6\frac{3}{4}$. It is evident that we are converging on the correct result.

The Speakeasy word TOTALINTEGRAL provides the results of the trapezoidal integration rule. You have to provide TOTALINTEGRAL with two things:

- (1) A suitable grid of points and
- (2) the function evaluated on that grid. Thus the Speakeasy attack on our problem would be

```
:_x=grid(-2,2,0.1)
:_y=(x**2)/2+1
:_totalintegral(y:x)
TOTALINTEGRAL(Y:X) = 6.67
:_
```

Speakeasy is also capable of calculating numerical indefinite integrals. The Speakeasy operation $G=\text{INTEGRAL}(F:X)$ returns an indefinite integral of the function F which has been evaluated on the "suitable"

3	9.0312	9
4	21.375	21.333
5	41.719	41.667
6	72.062	72
7	114.41	114.33
8	170.75	170.67
9	243.09	243
10	333.44	333.33

In performing numerical integration by the trapezoidal rule, it is not necessary to evaluate the integrand at equal spaces in the independent variable. In fact, better accuracy can be achieved by concentrating the points of evaluation in regions where the integrand is varying in size most rapidly.

The selection of an optimal set of coordinates for the evaluation of numerical integrals is a nontrivial problem. Some fairly specific knowledge of the behavior of the function of interest will permit an increased accuracy for a specific number of function evaluations. In the absence of such specific knowledge, the use of an equally spaced grid is perfectly satisfactory, particularly in an interactive environment such as TSO Speakeasy which allows the results of a given choice to be seen immediately and lets the validity of the choice be tested quickly by changing the grid specifications.

Other Techniques of Evaluating Definite Integrals: Gaussian Quadrature

There are special integration schemes designed to optimize the accuracy of a given integral for a specific number of evaluations of the integrand. These schemes generally rely on assumptions regarding the analytic behavior of the function to be integrated. A technique called Gaussian quadrature is particularly well suited to a class of integrals of the form

$$I = \int_a^b w(x)f(x) dx,$$

for which the so-called weight function $w(x)$ does not change sign and $f(x)$ is representable by a polynomial on the interval (a, b) . For such a situation, the integral can be approximated by

$$I = \sum_{n=1}^N w_n f(x_n),$$

where the weights w_n and coordinates x_n are specified by the Gaussian-quadrature rule. Furthermore, the integral is evaluated exactly for N function evaluations if $f(x)$ is representable by a polynomial of degree

$2N - 1$ or less. These values of w_n and x_n are related to the derivatives and roots of polynomials that, on the interval (a, b) , are orthogonal to the weight function $w(x)$. Formulas for the appropriate w_n and x_n for several choices of the weight function $w(x)$ are available in the literature. An excellent reference on this subject is a book by Stroud and Sechrest entitled Gaussian Quadrature Formulas. Speakeasy incorporates the four Gaussian formulas summarized as follows.

Gaussian Rule	Interval		$w(x)$	Available N	Speakeasy Call
	a	b			
Legendre	-1	+1	1	1—50	SETGAUSS(N,X,W,A,B) [†]
Jacobi	-1	+1	$(1-x)^a(1+x)^b$	1—50	SETJACOBI(N,X,W,ALPHA,BETA)
Laguerre	0	$+\infty$	$x^a e^{-x}$	1—32	SETLAGUERRE(N,X,W,ALPHA)
Hermite	$-\infty$	$+\infty$	$x^a e^{-x^2}$	1—24	SETHERMITE(N,X,W,ALPHA)

[†]The parameters A and B are optional; they need not be specified unless the integration range is other than -1 to +1. Specifying them results in a linear rescaling of the Gaussian coordinates and weights.

Operation with any one of these Gaussian rules returns arrays of the coordinates x and weights w for the specified values of N and the other parameters. The integral is then $\text{SUM}(W*F)$, where the integrand F has been evaluated at the points X . Further details on the Speakeasy operations SETGAUSS, etc. above, can be found in the Speakeasy HELP documents (discussed in Section 10 of this book).

The most widely used of the Gaussian rules is the Gauss-Legendre rule. We will close this section on integration by applying this rule to our original problem,

$$\int_{-2}^2 \left[\frac{1}{2}x^2 + 1 \right] dx.$$

Recall the statement that the Gaussian quadrature rule is exact for N evaluations of the integrand if this integrand is a polynomial of degree $2N - 1$ or less. This means we need only two points to evaluate

$$\int_{-2}^2 \left[\frac{1}{2}x^2 + 1 \right] dx.$$

We proceed as follows

```
:_setgauss(2,x,w,-2,2)
:_
```


This operation defines x and w . The integral is evaluated by

```
:_sum(w*(x**2/2+1))
SUM(W*(X**2/2+1)) = 6.6667
:_
```

The exact answer was $6-2/3$. As a check on the accuracy of the Gaussian-quadrature method, let us call for an evaluation to 15 significant digits.

```
:_significance(15)
:_answer
ANSWER = 6.666666666666667
:_
```

7. COMPLEX NUMBERS

In our discussions and examples thus far, we have restricted ourselves to real numbers. In the process of testing Speakeasy, you may have attempted a calculation that could have yielded a result that was imaginary or complex. If you did, then you were informed of this by an error message. For example

```
:_sqrt(-5)
IN LINE " SQRT(-5) " ENTERED COMPLEX DOMAIN.
:_
```

For many calculations it is desirable to be able to work in the complex domain. In Speakeasy you are free to select the allowed domain for calculations at any time. The command

```
:_domain complex
```

means that calculations with complex and imaginary numbers are now allowed. The command

```
:_domain real
```

means that calculations that involve complex or imaginary numbers will result in error messages. You can alternate between these allowed domains as you wish.

An imaginary number is expressed in Speakeasy by terminating it with the letter i . A complex number is expressed by combining a real and an imaginary number. Some examples of imaginary numbers are

```
:_x=2.14i,3i,-1i
```

Examples of complex numbers are

```
:_x=6+2i,2-9i,5.76+2i
```

A word of caution. The expression $6+I$ is not a representation of a complex number. I appearing alone refers to the variable named I , the expression above would indicate that you wished to add 6 to the variable I . The proper representation of the unit imaginary numbers is $1I$.

By this time you already are familiar with many of the available features in Speakeasy and you have explored their capabilities in the real domain. All of these are equally easily used in the complex domain.[†] You may define complex arrays and complex matrices, and you may carry out the standard algebraic operations with these objects. All of the special functions such as $\sin$, $\cos$, etc. are usable with either real or complex values. It might be useful to review the earlier parts of this book with this in mind and carry out calculations with complex numbers.

Some Speakeasy functions designed particularly for complex numbers are

CONJUGATE	which produces the complex conjugate
IMAGPART	which selects the imaginary part
REALPART	which selects the real part.

They are used as follows.

```

:_domain complex
:_x=matrix(2,2:3+4i,2-3i,6i,3)
:_x
  X (A 2 BY 2 MATRIX)
    3+4i  2-3i
    6i    3
:_conjugate(x)
  CONJUGATE(X) (A 2 BY 2 MATRIX)
    3-4i  2+3i
    -6i    3
:_imagpart(x)
  IMAGPART(X) (A 2 BY 2 MATRIX)
    4 -3
    6  0
:_realpart(x)
  REALPART(X) (A 2 BY 2 MATRIX)
    3  2
    0  3

```

[†] Most of the logical and relational operations described earlier are the sole exception to this generalization. It is not meaningful to make comparisons other than equality tests between complex objects and this is therefore not allowed.

8. PROGRAM MODE

Thus far in the examples, we have been using Speakeasy in a responsive mode of operation—i. e., much as if it were a desk calculator. Each of your requests has been processed immediately upon receipt by Speakeasy. If the request is properly formulated, Speakeasy performs the indicated operation and then prompts you to enter your next instruction.

This mode of operation, referred to as the MANUAL MODE, is a very powerful method for carrying out most straightforward nonrepetitious types of calculations. In combination with the almost immediate response in exploratory calculations, the large number of operators available in the existing Speakeasy vocabulary provides you with the means of obtaining answers to problems that are either straightforward and clearly defined or ones that require that subjective decisions be made on generated results at each step of the calculation.

For many problems the MANUAL MODE of operation will provide sufficient power to satisfy all your requirements. However, it is more likely that you will find that portions of calculations that you perform are similar in form to ones that you have done before. To avoid the repetitious typing that would be involved in such calculations, Speakeasy provides a means to make use of a previously defined set of requests. This is the so-called PROGRAM MODE. As you will see, this provides capabilities far above those of desk calculators and provides the means of tailoring Speakeasy to your specific needs.

In this chapter we will describe how a program (a set of saved requests) is constructed, edited, and used. In addition, we will describe some of the special facilities that are available when using such programs.

A Speakeasy program is constructed in a special mode of operation called the EDIT MODE. While in the edit mode, every statement that you type is saved. The operations indicated in the statements are not immediately performed, and no attempt is made to see if your requests are sensible or consistent. The edit mode is used to construct and tailor a set of requests so that they may be used at a later time; it offers you facilities to examine and modify your program as you are constructing it.

When your program is completed, it is examined by Speakeasy for obvious errors. All of the individual statements in your program are tested, but these tests are restricted to syntax and do not check their logic. If errors are detected, you are informed of this fact and you may then re-enter the EDIT MODE to correct these or other statements.

A program can be thought of as a series of statements that are being remembered and that are available as an alternative to retyping them.

Creating a Program

If you are in the manual mode (i. e., if the :_ prompt character is being displayed), you begin the construction of a program by typing in the word PROGRAM followed by the name that you wish to give that program. The processor responds by typing out

```
:_program table
EDIT INPUT MODE
%1      PROGRAM TABLE
2
```

You are now in the EDIT INPUT MODE and are prompted by a number. These numbers represent the order in which the statements that you are entering are to be processed. The prompting numbers are automatically incremented by one each time you enter a statement. You may now enter your statements one by one until your program is complete. The last statement in your program should be the single word END. This indicates to Speakeasy that the construction of the program is complete and that the processor is returned to the MANUAL MODE. This sequence of operations is illustrated below for a simple program.

```
:_program table
EDIT INPUT MODE
%1      PROGRAM TABLE
2 y=x*x+4
3 z=x*y
4 tabulate x,y,z
5 end
MANUAL MODE
:_
```

Using a Defined Program

Once a series of statements such as those illustrated above are defined, they can be used at any time during the Speakeasy session by typing EXECUTE followed by the name of the program. For example, we can use the program TABLE as follows.

```
:_x=1,2,3,4
:_execute table
EXECUTION STARTED
  X   Y   Z
  1   5   5
  2   8  16
  3  13  39
  4  20  80
MANUAL MODE
:_
```

You may define any number of programs in the manner illustrated above and you may assign each of them any name. A program is a defined object, just as a variable is, and its name has meaning to Speakeasy. You should therefore be careful not to use the same name for a program and for a variable and you should not use a Speakeasy keyword such as QUIT or END as a name of the program.

Modifying and Correcting a Program

In the preceding description it was assumed that the statements constituting your program were entered without error. Needless to say, this will not always be the case and it is necessary that you be able to correct and revise a previously defined program. To do this type the word EDIT followed by the name of the program. Speakeasy will respond by informing you that you are in the EDIT COMMAND MODE.

The Speakeasy editor was used previously to construct the program in the INPUT MODE. In the input mode statements defining your program are being entered. In the command mode you are directing the editor in its functions. The commands available enable you to list, delete, move and copy selected statements. There are also commands provided to allow you to insert statements at selected points between previously defined statements, and commands to change text within lines. Only a few of the commands are described here.

The commands available in the editor are:

LIST:

LIST	which lists the entire edit file
LIST N	which lists the statement with line number N
LIST N, M	which lists the statements with line numbers N through M

INSERT:

INSERT N	which inserts the statements that follow this command at locations N, N+1, N+2, ...
INSERT N,I	which inserts the statements that follow this statement at the locations N, N+I, N+2I, ...

A null line stops the insert.
INSERT may be abbreviated to I.

DELETE:

DELETE N	which deletes the statement numbered N
DELETE N, M	which deletes statements N through M

DELETE may be abbreviated to D.

MOVE:

MOVE N TO bottom	moves statement N to the last position
MOVE N M TO bottom	moves statements N through M to the last positions
MOVE N TO K	moves line N to position K
MOVE N M TO K	moves statements N through M to positions K+i, K+2i, where i is calculated so that the lines will not cover the one following K
MOVE N M TO K BY I	moves statements N through M to positions K, K+I, K+2I, ...

COPY:

COPY N TO bottom	copies statement N to the last position
COPY N M TO bottom	copies statements N through M to the last position
COPY N M TO K	copies statements N through M to positions K, K+i, K+2i, ... where i is calculated so that the lines will not cover the one following K
COPY N M TO K BY I	copies statements N through M to positions K, K+I, K+2I, ...

Note if a statement is being copied to or inserted at a statement number at which a previous statement exists, the previous statement is replaced. You should be careful while editing programs that such overlays do not occur unintentionally.

You may change from one mode of operation of the editor to the other at any time by entering a blank or null line.

After you have modified your program to your liking you return to the manual mode by typing the single word END.

The following section of a session shows a program being constructed, reedited and executed.

```

:_program try
EDIT INPUT MODE
%1      PROGRAM TRY
 2 y=x*2-/
 3 z=y-x+x*x
 4 tabulate x y z
 5 end
IN LINE " Y=X*2-/ : "  DOUBLE OP.
:_edit try
EDIT COMMAND MODE
:%%2 y=x*2-1
:%compile
MANUAL MODE
:_x=1,2,3
:_execute try
EXECUTION STARTED
  X  Y  Z
  1  1  1
  2  3  5
  3  5  11
MANUAL MODE
:_edit try
EDIT COMMAND MODE
:%list
%1      PROGRAM TRY
%2      Y=X*2-1
%3      Z=Y-X+X*X
%4      TABULATE X Y Z
%5      END
:%insert 3(.1)
 3 xsq=x*x
 3.1 z=y-x+xsq
 3.2 tabulate x y xsq z
 3.3
:%delete 4
:%list
%1      PROGRAM TRY
%2      Y=X*2-1
%3      XSQ=X*X
%3.1    Z=Y-X+XSQ
%3.2    TABULATE X Y XSQ Z
%5      END
:%run
EXECUTION STARTED
  X  Y  XSQ  Z
  1  1  1    1
  2  3  4    5
  3  5  9   11
MANUAL MODE
:_

```

Allowable Speakeasy Statements in Program Mode

All Speakeasy statements that have previously been described as available for use in the MANUAL MODE, are allowable in the PROGRAM MODE. This includes statements such as the implied print using a variable name, and Speakeasy commands such as NAMES and FREE. However, there are certain Speakeasy statements that can be used only in a program, for reasons that should be obvious. These statements, their syntax, and their operation are given as follows.

Labeled Statements

Any statement within a program may be labeled. A labeled statement is a standard Speakeasy statement preceded by a Speakeasy name followed by a colon, as follows.

```
A5:A=MATRX(3, 3.).
```

Thus, in the above statement, A5 is the label for a Speakeasy statement defining the 3 x 3 matrix called A with zero (0) values.

GO TO Statements

A GO TO statement alters the sequence of instructions normally executed by the program. The normal sequence is the order in which the program statements were numbered. GO TO statements may be either unconditional or conditional. An example of an unconditional GO TO statement is

```
GO TO A5.
```

The next statement executed will be the one with the label A5. If no statement with label A5 exists in the program, an error message will be printed and the processor returns to the manual mode. A conditional GO TO is created by combining a logical IF and a GO TO as follows.

```
IF (A. LT. B) GO TO A5.
```

In this case the next statement executed will be that with label A5 only if A is less than B; if not, the next statement is executed. Below is a program illustrating the use of GO TO and statement labels.


```

:_program xyz
EDIT INPUT MODE
%1      PROGRAM XYZ
 2 y=x*2-4
 3 if(x.gt.y) go to a
 4 z=y-x
 5 go to p
 6 a: z=x-y
 7 p: print x,y,z
 8 end
MANUAL MODE
:_x=5
:_execute xyz
EXECUTION STARTED
X =  5 Y =  6 Z =  1
MANUAL MODE
:_x=-1
:_execute xyz
EXECUTION STARTED
X = -1 Y = -6 Z =  5
MANUAL MODE
:_

```

RETURN

A RETURN statement terminates execution of a program and transfers control of the statement following the EXECUTE statement to the one that started the program or the program mode. Otherwise the processor returns to the MANUAL MODE. A RETURN statement is implied before the END statement of any program. The following example illustrates a program in which program TWO is called by another program ONE. Program TWO makes use of a RETURN statement.

```

:_program one
EDIT INPUT MODE
%1      PROGRAM ONE
 2 x=y/2
 3 xx=abs(x)
 4 if(xx.lt.20) go to t
 5 z=xx/2
 6 go to p
 7 t:execute two
 8 p:print y x z
 9 end
MANUAL MODE

```

```

:_program two
EDIT INPUT MODE
%1      PROGRAM TWO
  2 if(y.ne.0) go to no
  3 z=-1
  4 return
  5 no:if(y.gt.0) go to pos
  6 z=x-y+4
  7 return
  8 pos: z=y-x/2
  9 end
MANUAL MODE
:_y=10
:_execute one
EXECUTION STARTED
Y = 10 X = 5 Z = 7.5
MANUAL MODE
:_y=-5
:_execute one
EXECUTION STARTED
Y = -5 X = -2.5 Z = 6.5
MANUAL MODE
:_y=50
:_execute one
EXECUTION STARTED
Y = 50 X = 25 Z = 12.5
MANUAL MODE
:_y=0
:_execute one
EXECUTION STARTED
Y = 0 X = 0 Z = -1
MANUAL MODE
:_

```

CONTINUE

The CONTINUE statement performs no operations but can be used to provide a statement to which a label can be attached.

FOR LOOPS

The FOR LOOP is provided so that a section of a program may be executed repeatedly. The statements to be repeated start with the FOR statement and continue until a matching ENDLOOP statement is reached. The FOR statement specifies how many times the statements are to be executed, and its form is either

FOR N=START, STOP

or

FOR N=START, STOP, INCREMENT

where

N is a name of a scalar that may be used within the FOR loop

START, STOP, and INCREMENT are scalar expressions (not involving N) that respectively specify the initial value of N, its final value (its boundary), and the value to be added to N every-time the loop is repeated. START, STOP, and INCREMENT may be any real numbers. If INCREMENT is not specified, it is assumed to be +1 for the case in which START is less than STOP and -1 for the case in which START is greater than STOP.

An example of a program involving a FOR loop is given below.

```
:_program loop
EDIT INPUT MODE
%1      PROGRAM LOOP
 2 for i=2,n,2
 3 x=integers(1,i)
 4 f=cumprod(x)
 5 ii=x**x
 6 tabulate x, f, ii
 7 endloop i
 8 end
```

MANUAL MODE

:_n=6

:_execute loop

EXECUTION STARTED

X	F	ii
1	1	1
2	2	4
X	F	ii
1	1	1
2	2	4
3	6	27
4	24	256
X	F	ii
1	1	1
2	2	4
3	6	27
4	24	256
5	120	3125
6	720	46656

MANUAL MODE

:_

The range of a FOR loop is all statements starting with the FOR statement and continuing up to and including the ENDLOOP statement. FOR loops may be nested (one loop within another loop), provided that the range of the outer loop includes all statements within the range of the inner loop. In that case, the inner loop is completely executed (N=START TO N=STOP) for every value of N for the outer loop. Nested loops may

not have scalar names (N) that are the same. Nested loops may be nested to a depth of 10 loops.

Operations that require looping in other programming languages may often be done in Speakeasy with a single command by the use of array or matrix operations. In such cases, it is more efficient to use the available Speakeasy operation rather than a FOR loop. Some examples follow.

Example 1:

$$\sum_{i=1}^5 A_L B_L$$

Without FOR Loop

```
:_program one
EDIT INPUT MODE
%1      PROGRAM ONE
2 a=integers(1,5)
3 b=integers(21,25)
4 value=sum(a*b)
5 value
6 end
```

Example 2:

With FOR Loop

```
:_program two
EDIT INPUT MODE
%1      PROGRAM TWO
2 a=integers(1,5)
3 b=integers(21,25)
4 val=0
5 for n=1,5
6 val=a(n)*b(n)+val
7 endloop n
8 val
9 end
```

B is an array of 20 numbers. Find the values and positions of the minimum and maximum elements.

Without FOR Loop

```
:_program one
EDIT INPUT MODE
%1      PROGRAM ONE
2 max(b)
3 locmax(b)
4 locmin(b)
5 min(b)
6 end
```

With FOR Loop

```
:_program two
EDIT INPUT MODE
%1      PROGRAM TWO
2 maxb=b(1)
3 minb=b(1)
4 lmax=1
5 lmin=1
6 for i=2,20
7 if(b(i).le.maxb) go to s
8 maxb=b(i)
9 lmax=i
10 s:if(b(i).ge.minb) go to e
11 minb=b(i)
12 lmin=i
13 e:endloop i
14 maxb
15 lmax
16 minb
17 lmin
18 end
```

INPUT/PAUSE/STOP

It is often desirable to interrupt a program at a specific place but to retain the capability of resuming the execution of the program at the point of interruption. The statements INPUT, PAUSE, and STOP are words that enable you to interrupt a program. If the first word of a statement encountered during the execution of a program is INPUT, PAUSE, or STOP, then the entire statement is printed out and the execution of the program is interrupted. You are now in a HOLDING MODE. This mode is indicated by the prompt character :>. The HOLDING MODE is in essence the MANUAL MODE with the added capability of being able to resume the execution of the interrupted program at a later time. All the facilities of the MANUAL MODE are available for your use in the HOLDING MODE. However, you should not execute another program because doing so terminates the ability to continue execution of the interrupted program.

RESUME/CONTINUE/GO

The execution of the interrupted program may be resumed at the point of interruption by entering either RESUME, CONTINUE, GO, or a blank line.

The following sequence of statements illustrates the use of the HOLDING MODE. The program that is being used is listed first.

```
:_program hold
EDIT INPUT MODE
%1      PROGRAM HOLD
 2 input x
 3 y=x*2-5
 4 pause for input of a and b
 5 z=a*x-b*y
 6 tabulate x, y, z
 7 end
```

MANUAL MODE

```
:_execute hold
EXECUTION STARTED
INPUT X
:>x=integers(1,5)
:>resume
PAUSE FOR INPUT OF A AND B
:>a1=max(y)
:>a2=min(y)
:>a=a1-a2
:>b=a/2+4
:>continue
```

```
  X  Y  Z
 1 -3 32
 2 -1 24
 3  1 16
 4  3  8
 5  5  0
```

```
MANUAL MODE
```

\$ Comments

It is often desirable to include explanatory comments in a program as it is being written in order to make later understanding easier. Such comments play no role in the calculation but are an important part of proper program documentation. In Speakeasy the symbol \$ is used to indicate the beginning of a comment. An optional second \$ is used to terminate a comment if only part of a statement is a comment.

```

:_program minmax
EDIT INPUT MODE
%1      PROGRAM MINMAX
 2 $this program illustrates functions on 2-dim. arrays.
 3 $x is a previously defined 2-dim. array.
 4 rowmin(x) $gives the row of the minimum element.
 5 rowmax(x) $gives the row of the maximum element.
 6 colmin(x) $gives the column of the minimum element.
 7 colmax(x) $gives the column of the maximum element.
 8 minx=min(x)
 9 print 'the value of the smallest element is' minx
10 maxx=max(x)
11 print 'the value of the largest element is' maxx
12 $maxofcol and maxofrow define 1-dim. arrays containing$
13 $the maximum elements of each column and row respectively.$
14 cmax=maxofcol(x)
15 rmax=maxofrow(x)
16 print 'the maximum elements in each column are' cmax
17 print 'the maximum elements in each row are' rmax
18 end
MANUAL MODE
:_x1=1,9,5
:_x2=6,2,8
:_x3=3,6,5
:_x=array(3,3:x1,x2,x3)
:_x
  X (A 3 BY 3 ARRAY)
  1  9  5
  6  2  8
  3  6  5
:_execute minmax
EXECUTION STARTED
ROWMIN(X)           : = 1
ROWMAX(X)           : = 1
COLMIN(X)            : = 1
COLMAX(X)            : = 2
THE VALUE OF THE SMALLEST ELEMENT IS MINX = 1
THE VALUE OF THE LARGEST ELEMENT IS MAXX = 9
THE MAXIMUM ELEMENTS IN EACH COLUMN ARE
  CMAX (A 3 COMPONENT ARRAY)
  6  9  8
THE MAXIMUM ELEMENTS IN EACH ROW ARE
  RMAX (A 3 COMPONENT ARRAY)
  9  8  6
MANUAL MODE

```

9. CONTROLLING THE FORM OF OUTPUT

Each user of Speakeasy has special requirements for the form of his results. Many of these are met by the choices already built into Speakeasy. You can exercise considerable individual control over them by the use of special words to be described here. None of these words control the actual results of calculations; they are used solely to control the form used in typing the results.

SIGNIFICANCE

Normally Speakeasy prints the first 5 significant figures of results. You may select alternative numbers of figures (up to 14) by entering the line

SIGNIFICANCE N

where N is the desired number of figures.

SETNULL

Very small numbers generated during calculations are often in fact insignificant. You may set the value of what you consider this limit. It is normally set to 10^{-16} , i.e., .000000000001. If you would like to make it 10^{-8} , enter

SETNULL 1E-8.

SETINFINITY

Similarly you can set the largest number you are willing to have printed. If a larger number is encountered, the word INF is printed instead.

SETINFINITY 1E40.

MARGINS

You may control the margins for printed output by typing

MARGINS(N, M)

where N is the column number of the left margin and M the corresponding number for the right. All printout will be restricted to lie between those columns.

RATIONALIZE

This command requests that all results be printed in fractional form if this looks reasonable. Therefore .5 will appear as 1/2, .75 as 3/4, and .3333333 as 1/3. This particularly neat form of printing is useful for displaying the simplicity of many results. It is turned off by the command

NORATIONALIZE.

The use of this command is illustrated by the following example.

```
:_angles in degrees
:_x=grid(0,180,15)
:_sinx=sin(x)
:_cosx=cos(x)
:_tanx=tan(x)
:_setinfinity(1.e15)
:_setnull(1.e-15)
:_rational
:_tabulate x,sinx,cosx,tanx
  X      SINX      COSX      TANX
  0       0         1         0
  15     .25882     .96593     .26795
  30     1/2        .86603     .57735
  45     .70711     .70711     1
  60     .86603     1/2        1.7321
  75     .96593     .25882     3.7321
  90     1          0         -INF.
  105    .96593    -.25882    -3.7321
  120    .86603    -1/2       -1.7321
  135    .70711    -.70711    -1
  150    1/2       -.86603    -.57735
  165    .25882    -.96593    -.26795
  180    0         -1         0
:_norational
:_y=30
:_sin(y)
SIN(Y) = .5
:_
```

Output from Speakeasy is designed in part to minimize the amount of paper used (and also the typing time). When printing many objects, it is, therefore, difficult to compare corresponding elements. The command AUTOTAB causes printout to have fixed column widths that will help make such printout more legible. The command

ENDAUTOTAB

is used to restore the normal compact print form.

AUTOPRINT

Sometimes it is desirable to print a variable each time it is changed (redefined). For example, `AUTOPRINT(X, Y, Z)` prints each variable X, Y, or Z each time it appears to the left of an equal sign. The command

`AUTOPRINT`

without any arguments prints all variables as they are defined or redefined. Automatic printing of variables may be stopped by the command

`ENDAUTOPRINT.`

This is illustrated in the following example.

```
:_autoprint
:_halfpi=acos(0)
HALFPI = 1.5708
:_pi=2*halfpi
PI = 3.1416
:_twopi=2*pi
TWOPI = 6.2832
:_endautoprint
:_significance 14
:_print pi, halfpi, twopi
PI = 3.1415926535898 HALFPI = 1.5707963267949
TWOPI = 6.2831853071796
:_
```

TABULATE

It is often convenient to obtain output of values of related elements of arrays in tabular form. The Speakeasy command `TABULATE` produces tabulated output of one-dimensional arrays. The command `TABULATE` has been illustrated in the section on one-dimensional arrays. Here is another example.

```
:_a=integers(-4,4,2)
:_b=(a-1)*2
:_c=abs(b)-abs(a)
:_tabulate a b c
  A  B  C
-4 -10 6
-2 -6  4
 0 -2  2
 2  2  0
 4  6  2
:_
```

10. OTHER USEFUL COMMANDS

QUIT

To end a Speakeasy session enter the command

QUIT

This command gets you out of Speakeasy and tells you how much space you were using when you terminated the Speakeasy session, your maximum space used, and the amount of space that was allocated for the session.

SIZE

You may increase or decrease the amount of space allocated for data by use of the SIZE command. It must be your first Speakeasy statement. The default space allocated for data is generally 10K but varies with the installation. You may learn what the default size is at your installation from the response when you end a Speakeasy session with the command QUIT for a session in which you have not used the SIZE command. The value of core allocated is the default size. There is an upper limit to the amount of space you can allocate. This value is also unique to each TSO installation.

NAMES, FREE, CLEARDATA, CLEAR

During a Speakeasy session, you may get the response

TOO MUCH ACTIVE DATA

and your previous statement will not be carried out. In order to define further variables, structured objects, or programs, you must get rid of some defined objects which you longer need. To learn the names of currently defined objects enter the command

NAMES

The response gives the names of all currently defined variables, objects, and programs. The definitions of objects may be removed from your space for data by the command

FREE(NAME1, NAME2, NAME3, ..., NAMEN)

where NAME1, NAME2, NAME3, ..., NAMEN are names of defined objects. Any number of variables, structured objects, and programs may be freed in one such statement. If you no longer have need of any previously defined data, you may free it all at once by the command

CLEARDATA

The command CLEARDATA frees all previously defined numerical data, but does not free programs. However, the command

CLEAR

erases the entire active data-storage area. After the execution of the statement CLEAR, there are no user-defined objects, numerical data or programs.

The following sequence of statements illustrates the differences between these statements. After each statement the command NAMES is used so that you may see what objects are still defined.

```

:_names
  CURRENTLY DEFINED NAMES
  A , B , HALFPI , TABLE (A PROGRAM) , C , PI ,
  TWOPI , ETSO (A PROGRAM) , X , SINX , COSX , TANX
  , Y
:_free x,sinx,cosx,tanx
:_names
  CURRENTLY DEFINED NAMES
  A , B , HALFPI , TABLE (A PROGRAM) , C , PI ,
  TWOPI , ETSO (A PROGRAM) , Y
:_cleardata
:_names
  CURRENTLY DEFINED NAMES
  TABLE (A PROGRAM) , ETSO (A PROGRAM)
:_clear
:_names
  NOTHING DEFINED
:_

```

HENCEFORTH

HENCEFORTH enables you to define synonyms for Speakeasy words. The statement

HENCEFORTH X is Y

means that the word X will be treated as Y in all following statements. The word IS in the expression is arbitrary. Any word may be used in place of IS. The use of HENCEFORTH to redefine a Speakeasy word is illustrated in the following example.

```

:_henceforth m is matrix
:_henceforth p is print
:_y=m(3,3:integer(2,18,2))
:_x=m(3,3:1,3,2,-2,-4,-2,-10,-15,-14)
:_p x y
X (A 3 BY 3 MATRIX)
  1  3  2
-2 -4 -2
-10 -15 -14
Y (A 3 BY 3 MATRIX)
  2  4  6
  8 10 12
 14 16 18
:_henceforth d is diagels
:_d(x)
D(X) (A 3 COMPONENT ARRAY)
  1 -4 -14
:_d(y)
D(Y) (A 3 COMPONENT ARRAY)
  2 10 18
:_w=d(x) + d(y)
:_p w
W (A 3 COMPONENT APRAY)
  3 6 4
:_

```

HENCEFORTH itself may be given a synonym by a HENCEFORTH definition. Note that the word HENCEFORTH is still defined and may be used in later expressions.

```

:_henceforth let is henceforth
:_let p be print
:_henceforth m is matrix
:_n=m(2,2:3,6,1,4)
:_let ti be transpose inverse
:_ti(n)
TI(N) (A 2 BY 2 MATRIX)
  .66667 -.16667
 -1      .5
:_p n
N (A 2 BY 2 MATRIX)
  3 6
  1 4
:_

```

A few very special Speakeasy words may not be given synonyms by this method. These are PROGRAM, PROCEDURE, GOTO, FOR, ENDLOOP, END, DATA, LOADDATA, QUIT.

11. LEARNING MORE ABOUT SPEAKEASY

This book has described only a selected set of the basic facilities of TSO Speakeasy. A large amount of additional information is available to help you learn more about Speakeasy and to aid you in the use of the many features available. The user consultant at your computer center has additional documentation and can refer you to other sources of information. However, many of the more useful documents and learning aids are available to you through Speakeasy itself. These are the Speakeasy HELP documents and the Speakeasy TUTORIAL.

The HELP documents are a very large set of short documents designed to explain how each of the available features of Speakeasy is used. Since there are a very large number of words in the language, it is necessary to provide you with some sensible way of locating a feature of interest to you. The command VOCABULARY is interpreted by Speakeasy as a request for an index to the available HELP documents. Speakeasy will respond by typing out a complete alphabetical listing of the names of the documents. (These names are truncated to the first eight characters for very long words.)

If you are looking for a particular operation you might locate it immediately in this list. If not, a quick search of the list may provide you with a likely alternative to that name. Once you have located such a member of this list you type

```
:_HELP name
```

(where name is the name in the index) to obtain a copy of the document describing that particular operation.

The command HELP also provides a means for helping you find specific operations even if you cannot easily locate them by name. A tree structure of classification is available to you so that you can quickly locate a document by the general type of operation involved. To use the HELP command in this way, first type the word HELP by itself to find the general topics available. Then type HELP followed by the topical name to find information about this topic. Several successive uses of the HELP command will lead you to the specific document of interest.

The following example shows the use of the word VOCABULARY and of the HELP processor.

:_vocabulary

ABS	COT	FRACPART	LIBRARIE	NAMES	READ	SUMS
ACCURACY	CREATE	FREE	LIBRARYN	NE	REAL	SUMSQ
ACOS	CREATEME	GAMMA	LINEPLOT	NEUMANN	REALPART	SUMSQCOL
ACOT	CUMPROD	GE	LINKLIB	NEWGRAPH	REAL4	SUMSOROW
ADDGRAPH	CUMSUM	GEIGEN	LINKULES	NEWPAGE	REAL8	SYMBOLS
ADJOINT	DATA	GO	LIST	NEWS	RECLASS	SYMMAT
AFAM	DEBUGGIN	GOTO	LISTHEAD	NOCOLS	RESTRICT	TABULATE
AMAT	DEC	GRAPH	LISTMEMB	NOECHO	RESUME	TAN
AND	DEFINEA1	GRAPHICS	LISTPROG	NOELS	RETURN	TIME
ANGLES	DEFINEA2	GRID	LOADDATA	NORATION	ROOTS	TOTALINT
ARRAY	DELETE	GT	LOC	NORROOTS	ROWARRAY	TOTINT
ARRAYS1	DERIV	HELP	LOCMAX	NOROWS	ROWMAT	TRACE
ARRAYS2	DERIVATI	HENCEFOR	LOCMIN	NOT	ROWMAX	TRANSFAM
ARRAY2D	DET	HIERARCH	LOCS	NOZEROS	ROWMIN	TRANSP
ASIN	DIAGELS	HIGHWIDE	LOG	NUMBERS	RUN	TRANSPOS
ASYMMAT	DIAGMAT	HIWIDE	LOGGAMMA	OBJECT	SELECT	TRIG
ATAN	DMAT	HLABEL	LOGIC	OBJECTS	SETGAUSS	TSO
AUTOCORE	DOCUMENT	HSCALE	LOWERTRI	OMITCLAS	SETINFIN	TUTORIAL
AUTOPRIN	DOMAIN	HSIZE	LT	ONEDIMFU	SETJACOB	TWODIMFU
AUTOTAB	DONTLIST	IF	MACHINE	ONERROR	SETLAGUE	UMAT
AVERAGE	DOUBLEFA	IMAG	MARGINS	OR	SETLEGEN	UNITMAT
AID	DUMP	IMAGPART	MAT	ORDERED	SETLIB	UPPERTRI
A2D	ECHO	INOUT	MATH	ORDERER	SETNULL	USE
BESSEL	EDIT	INPUT	MATRICES	OTHERS	SETPLOT	USEMEMBE
BESSELK	EDITMODE	INPUTS	MATRIX	OUTPUT	SIGN	VARIABLE
BUGS	EIGENSYS	INSERT	MATRIXDE	PAUSE	SIGNIFIC	VEC
CGAMMA	EIGENVAL	INTEG	MATRIXOP	PLOTSYMB	SIMEQ	VECTOR
CLEAR	EIGENVEC	INTEGERS	MAX	PLOTTITL	SIN	VECTORDE
CLEARDAT	ELEMENTA	INTEGRAL	MAXOFCOL	PRINT	SINGLEVA	VECTORS
COLARRAY	ELLIPE	INTEGRAT	MAXOFROW	PRINTCLA	SINH	VERSIONS
COLMAT	ELLIPK	INTERP	MELD	PROCLIB	SIZE	VFAM
COLMAX	END	INTERPOL	MFAM	PROD	SMAT	VLABEL
COLMIN	ENDAUTOP	INTPART	MIN	PRODCOLS	SORT	VOCABULA
COLWIDTH	ENDLOOP	INTS	MINOFCOL	PRODRROWS	SPACE	VSCALE
COMMANDS	EONE	INT2	MINOFROW	PRODUCTS	SPECIAL	VSIZE
COMPILE	EQ	INT4	MISCELLA	PROGRAM	SPHRES	WHATIS
CONJ	ERF	INVERSE	MONEY	PROGRAMM	SPHBESN	WHERE
CONJUGAT	ERFC	KEEP	MOVE	PROGRAMS	SQRT	WHEREVER
CONSTRAI	ERRORS	KEPT	MYDOCS	PUNCH	STOP	WHOLE
CONTINUE	EXECUTE	LABEL	MYHELP	QUIT	STRUCTUR	WRITE
CONVERT	EXP	LE	MYKEEP	RANDOM	SUM	ZEROS
COPY	FIN	LENGTH	MYKEPT	RANKED	SUMCOLS	
COS	FOR	LIBINDEX	MYLINKS	RANKER	SUMPROD	
COSH	FRAC	LIBNAMES	MYPROCS	RATIONAL	SUMROWS	

:_

```

:_help arrays1
  ARRAYS1 are words dealing with 1-dimensional arrays.
DEFINEAL      (Functions defining 1-dim. arrays)
ONEDIMFUNCTIONS (Functions of 1-dim. objects)
STRUCTURE     (Fuctions which give information about the
               structure of structured objects)
SUMPROD       (Sums and products of elements of
               structured objects)

:
  To obtain the words in a given subclass SC, enter
  HELP SC
:_help defineal
  DEFINEAL are words used in defining 1-dimensional arrays.
AFAM          defines a member of the array family.
ARRAY         defines an array.
AID           defines a 1-dim. array.
DIAGELS       returns the diagonal elements of a matrix.
GRID          defines a 1-dim. array of equally spaced points.
INTEGERS      defines a 1-dim. array of integers.
INTS          defines a 1-dim. array of integers.
LOWERTRI      gives the lower triangular part of a 2-dim. array.
UPPERTRI      gives the upper triangular part of a 2-dim. array.
VARIABLE      defines a 1-dim. array of equally spaced points.

:
  To obtain a description of a given word XXX, enter
  HELP XXX
:_

```

A brief explanation of the meaning and use of any Speakeasy word is obtained by the command HELP followed by the desired word.

```

:_help diagels
  DIAGELS(X) selects the diagonal elements of X.
The diagonal elements are given in the order in which they
occur in X. X must be a square matrix or array. The result
is a 1-dimensional array.
:_

```

Note that it is not necessary to go through the complete tree structure to find out about a particular word. The tree structure is merely available to assist the new user who is not aware of the features available. If you just want to learn how to use a particular word, enter HELP followed by the desired word. For example, the meaning of the Speakeasy word MATRIX is found by entering HELP MATRIX.

In most computer centers, copies of the complete sets of Speakeasy HELP documents are available in printed form. Such listings can

be of great use for quickly examining many documents. If you make frequent use of the HELP facility, you might ask to have a complete set of the documents made available near your terminal.

The Tutorial

A large number of tutorial sessions designed to teach you how to use Speakeasy are included in the facilities available at every TSO-Speakeasy installation. The sessions are in the form of short paragraphs describing specific features of the language with examples and suggested tests that you can perform. The tutorial is constantly being augmented by contributed sessions. You will find the tutorial a valuable aid both for learning about other features of Speakeasy, and for greater detailed explanations for many of the topics covered here.

An index to the available tutorial sessions is obtained by entering the command

```
:_tutorial
INDEX PAGE = 0
  INDEX TO THE SPEAKEASY TUTORIAL SESSIONS  July 8,1973
    SESSION      SUBJECT
    START        THE INTRODUCTORY SESSION
    ARRAYS        INTRODUCTION TO ARRAY OPERATIONS
    MATRIX        MATRIX DEFINITIONS AND OPERATIONS
    VECTOR        VECTOR DEFINITIONS AND USES
    VECMAT        VECTOR-MATRIX OPERATIONS
    TEK           USE OF THE TEKTRONIX DISPLAY
    KEEP          SAVING INFORMATION BETWEEN RUNS
Type  TUTORIAL XXX to begin the tutorial session called XXX.
Type  TUTORIAL XXX n to display page n of that session.
Type  MORE to continue a session
:_
```

The response to this command will vary depending on the sessions available at your installation. To begin a particular session type the command TUTORIAL followed by the name of the session.

```
:_tutorial start
START PAGE = 0
START
  Table of Contents
  1 How to use the Tutorials
  2 Introduction
  3 Arithmetic Operators
  4 Answer
  5 Representation of Real Numbers
  6 Representation of Complex Numbers
  7 Significance
  8 Defining a Variable
  9 Built-in Functions
 10 Conclusion
```


Type the word MORE to obtain the next page of the tutorial session.

```
:_more
START PAGE = 1
HOW TO USE THE TUTORIALS
```

This tutorial is a series of short paragraphs written to help you learn about Speakeasy. It is written for use with the TSO-Speakeasy processor. To use this tutorial, read the information printed, and try to carry out some examples of the topic being discussed. When you feel that you understand the subject, enter the command MORE. This will request the next part of the tutorial to be displayed. If you wish to restart the tutorial at some later time, at a particular place, type in the statement TUTORIAL XXX N where XXX is the heading of the section, and N is the page number.

Remember that typing SPEAKEZ will allow you to use the TSO-Speakeasy processor and typing QUIT will return you to TSO.

Now, enter the command MORE.

12. CONCLUSION

This book has described only the very basic elements of the Speakeasy language. The vocabulary of words and operations available to you is large and is growing. As users of the system explore Speakeasy's capabilities and apply it to new problems there is a steady increase in capabilities available to you.

In your use of Speakeasy you may find a particular operation or set of operations are lacking. If this is the case, discuss your needs with your local consultant. You may be surprised to find that they already exist or that it is not very difficult to add them yourself. Then you will understand what we mean when we say, "Speakeasy Is the People's Language," Speakeasy is your language.

INDEX

- Addition, 8-10
- ACOS (arccosine), 11
- ACOT (Arccotangent), 11
- Algebra
 - Hierarchy of operations, 8
 - for vectors and matrices, 33-37
- AND (logical sum), 25
- ANGLES (degrees or radians), 11
- ANSWER, 10
- ARRAY, 15-22
 - definition of 1-dim, 15-18
 - definition of 2-dim, 19-20
 - operations on 1-dim, 18-19
 - operations on 2-dim, 21-22
- Arithmetic
 - expressions, 9-10
 - operators, 8
- AUTOPRINT, 27-28, 61
- AUTOTAB, 60
- AVERAGE, 19
- Branching within a program, 54, 64
- Calculus, 39-45
- Cancel an input line, 29
- CLEAR, 63
- CLEARDATA, 63
- COLMAX, 22
- COLMIN, 22
- Commands, 62-64
- COMMENTS (in a program), 58
- Complex arithmetic, 45-46
- Complex conjugate, 46
- Complex numbers, 45-46
- Conditional statements, 27
- CONJUGATE, 46
- CONTINUE, 54, 57
- Control over printed output, 12, 16, 59-61
- COPY, 50
- COS (cosine), 11
- COSH (hyperbolic cosine), 11
- COT (cotangent), 11
- DATA, 64
- Degrees, 11
- Defining
 - arrays, 15-18, 19-20
 - matrices, 33
 - objects, 13-14
 - pi, 12
 - programs, 48-49
 - vectors, 35
- DELETE (an edit command), 49
- DERIVATIVE, 40-41
 - definition, 40
- DET, 38
- Determinant (of a matrix), 38
- DIAGELS, 38
- Diagonal elements of a matrix, 38
- Division, 8-10
- EDIT, 49-50
 - EDIT COMMANDS, 49-50
 - EDIT COMMAND MODE, 49
 - EDIT INPUT MODE, 49
- Eigenvalues of a matrix, 39
- Eigenvectors of a matrix, 39
- END, 50, 64
- End of a Speakeasy session, 62
- ENDAUTOPRINT, 28, 61
- ENDAUTOTAB, 60
- ENDLOOP, 54, 55
- EQ, 24
- Equal sign, 14
- Error detection facilities, 28-30
- EXP, 12
- Exponentiation, 8-9, 12
- FOR, 54-56, 64
- FREE, 62
- Functions
 - for complex numbers, 46
 - trigonometric, 11
 - special functions, 12
- Gauss integration
 - Gaussian quadrature, 43-45
 - example using Gauss-Legendre Quadrature formula, 44-45
- GE (greater than or equal to), 24
- GO, 57
- GOTO (GO TO), 52, 64
- GRID, 17
- GT (greater than), 24
- HELP, 65-68
- HENCEFORTH, 63
- Hierarchy of arithmetic operations, 9
 - with logical and relational operations, 25-26
- HOLDING MODE, 57
- IF, 27
- Imaginary numbers, 45-46
- IMAGPART, 46
- Implied print, 14-15
- Indices, 23-24
- INPUT, 57
- Input and output, 12, 16, 29, 59-61
- INSERT (an edit command), 49
- INTEGERS, 16
- INTEGRAL, 42
- Integration techniques, 41-45
 - Gaussian quadrature, 43-45
 - INTEGRAL, 42
 - SETGAUSS, 44
 - SETHERMITE, 44
 - SETJACOBI, 44
 - SETLAGUERRE, 44
 - TOTALINTEGRAL, 42
- INVERSE, 38
- Labels for statements, 52
- LE (less than or equal to), 24
- LIST (edit command), 49
- LOADDATA, 64
- LOC, 26
- LOG, 12
- Logic
 - logical and relational statements, 24-26
 - logical operators, 25
 - logical functions, 26
- Loops, 54-56
- LT (less than), 24
- MANUAL MODE, 6, 47
- MARGINS, 59
- Mathematical
 - operators, 8

- expressions, 9-10
- replacement statement, 14
- MATRIX, 33-39
 - definition, 33
 - eigenvalues and eigenvectors, 39
 - matrix algebra, 33-34
 - matrix operators, 38-39
- MAX, 18
- MAXOFCOL, 22
- MAXOFROW, 22
- MIN, 18
- MINOFCOL, 22
- MINOFROW, 22
- MODES
 - EDIT, 47, 49
 - HOLDING, 57
 - INPUT, 49
 - MANUAL, 6, 47
 - PROGRAM, 47
- MOVE (an edit command), 50
- Multiplication, 8-10
 - of matrices, 33-34
- NAMES, 62
- Names of objects, 13-14
- Names of programs, 49
- Napierian logarithm, 12
- NE (logical inequality), 24
- NOCOLS, 22
- NOELS, 19
- NORATIONALIZE, 60
- NOROWS, 22
- NOT (logical not), 25
- Notation
 - complex numbers, 45-46
 - names, 13-14, 49
 - numbers, 8-10, 12-13
 - elements of objects, 23-24
- One-dimensional arrays, 15-18
- One-dimensional functions, 18-19
- Operations
 - on matrices, 38
 - on parts of objects, 23-24
 - on one-dimensional arrays, 18-19
 - on two-dimensional arrays, 21-22
- OR (inclusive or), 25
- OUTPUT
 - AUTOPRINT, 27-28
 - AUTOTAB, 60
 - control over form, 59-61
 - implied print, 14-15
 - MARGINS, 59
 - RATIONALIZE, 60
 - SETINFINITY, 59
 - SETNULL, 59
 - SIGNIFICANCE, 12
 - TABULATE, 16
- Parentheses (use of), 9-10
- PAUSE, 57
- PROCEDURE, 64
- PRODCOLS, 21
- PRODROWS, 21
- Program
 - construction of, 48
 - continuation after interrupt, 57

- execution of, 48
- interrupting a program, 57
- modification of existing, 49
- names, 49
- statements, 52-58
- PROGRAM, 64
- QUIT, 62, 64
- Radians, 11
- RATIONALIZE, 60
- READY, 7
- REALPART, 46
- Redefining object names, 63
- Relational operators, 24
- RESUME, 57
- RETURN, 53
- ROWMAX, 22
- ROWMIN, 22
- Scientific notation, 12-13
- Selection of elements of an array, 26
- Sets of numbers, 15-19
- SETGAUSS, 44
- SEThERMITE, 44
- SETINFINITY, 59
- SETJACOBI, 44
- SETLAGUERRE, 44
- SETNULL, 59
- SIGNIFICANCE, 12, 59
- Simultaneous equations, 30-32
- SIN (sine), 11
- SINH (hyperbolic sine), 11
- SIZE, 62
- Space allocation for data, 62
- SPEAKEZ, 7
- Special functions, 10-12
- SQRT (square root), 12
- Statement labels, 52
- STOP
 - (with FOR LOOPS), 54-55
 - (to interrupt a program), 57
- Structured indices, 23-24
- Subtraction, 8-10
- SUM, 18
- SUMCOLS, 21
- SUMROWS, 21
- SUMSQ, 18
- SUMSQCOLS, 21
- SUMSQROWS, 21
- Syntax errors, 28
- TABULATE, 16, 61
- TAN (tangent), 11
- TOO MUCH ACTIVE DATA, 62
- TOTALINTEGRAL, 42
- TRACE, 38
- Trigonometric functions, 11, 18
- TSO Speakeasy, 7
- TUTORIAL, 65, 68-69
- Two-dimensional arrays, 19-20
- Two-dimensional functions, 21-22
- Variables, 13-14
- VECTOR, 35-37
 - definition, 35
 - matrix-vector algebra, 36-37
- VOCABULARY, 65-66
- WHERE, 27