A Spectrum Book

NUMERICAL ANALYSIS

WITH THE TI 99/4A™, COMMODORE 64™, APPLE II+/IIe™ AND TRS-80 model I/III™

Includes Annotated BASIC Program Listings

H. R. Meck

H. R. Meck received his B.S. from Lehigh University and a master's degree from Harvard. He first pursued a career in industry and government solving engineering problems in atomic energy, then went on to become a freelance writer. The author of many published journal articles in the field of solid mechanics, Mr. Meck also wrote the successful book *Scientific Analysis for Programmable Calculators.*

NUMERICAL ANALYSIS

With the TI-99/4A, Commodore 64, Apple II Plus/IIe, TRS-80 Model I/III

H. R. Meck

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Meck, H. R.
Numerical analysis with the TI 99/4A, Commodore 64, Apple II+/IIe, TRS-80 Model I/III

"A Spectrum Book."
Bibliography: p.
Includes index.
1. Numerical analysis—Data processing. 2. Microcomputers—Programming. 3. Basic (Computer program language) I. Title.
QA297.M43 1984 519.4'028'542 84–3425
ISBN 0–13–626649–5
ISBN 0–13–626631–2 (pbk.)

This book is available at a special discount when ordered in bulk quantities.
Contact Prentice-Hall, Inc., General Publishing Division,
Special Sales, Englewood Cliffs, N.J. 07632.

A SPECTRUM BOOK

1 2 3 4 5 6 7 8 9 10

Printed in the United States of America.

ISBN 0-13-626649-5

ISBN 0-13-626631-2 {PBK.}

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*
Editora Prentice-Hall do Brasil Ltda., *Rio de Janeiro*

Contents

Preface

This book introduces the reader to scientific programming in the BASIC language. It is suitable either as a self-study resource or as a textbook for a college course. The only prerequisite is a knowledge of calculus and linear algebra through the level reached in a typical undergraduate engineering curriculum; otherwise the book is essentially self-contained. Some prior knowledge of BASIC programming may be helpful, but it is not necessary. The required statements and operations are introduced as needed. Problems appear at the end of each chapter. Answers are given for all problems, either in the problem statements or at the back of the book. Suggested solutions to the more difficult problems are given at the back of the book.

The emphasis throughout the book is on writing computer programs to solve scientific problems, not on the theoretical foundations of numerical analysis. However, the numerical methods are explained as they are used. Derivations of a few of the more complicated algorithms are given in an appendix. Also, the book is not concerned with the BASIC language for its own sake; many techniques that might be useful in other contexts, such as business programming, are not considered.

The book can be used for convenient reference even by readers who

are not interested in writing programs, since the programs are useful in themselves and can be applied to practical problems.

The content of this book is similar to that of my earlier book on programmable calculators (reference 8), but more extensive. Chapter 1 is an introductory chapter that presents the most important BASIC statements and solves a number of representative problems. Chapter 2 is concerned with finding roots of equations, and Chapter 3 evaluates a number of commonly occurring higher transcendental functions. Chapter 4 is devoted to numerical integration, and Chapter 5 is concerned with differential equations. Chapter 6 covers matrices and simultaneous equations. If the book is used as a text, Chapter 1 and possibly the first section of Chapter 2 should be read first. The remaining five chapters are almost entirely independent of each other, and they may be read in any order. If the book is used for reference, programs of interest can be extracted from any part without studying the background material.

Details of the BASIC language vary from one model of computer to another. The programs in this book are written in a simple version of BASIC that works successfully with almost any microcomputer in common use, as well as many larger computers. Special statements and operations that work with only one or two models are avoided. The programs have been run on four models of microcomputer: the TRS-80 (Models I and III), the Apple (II Plus and IIe), the Commodore 64, and the TI-99/4. The first three machines use Microsoft BASIC and the last uses a dialect that is similar to standard BASIC. (The TI-99/4A uses the same BASIC as the TI-99/4.) Most of the programs will run as they stand on any of these models, but a few may require minor editing. Lines that may require editing are pointed out wherever they occur. While using this book, the reader should have the manufacturer's manual for his own model.

I wish to express my appreciation to Dr. J. T. Rice, Professor of Mechanical Engineering at Pratt Institute, for reviewing the manuscript and the programs from the standpoint of the TRS-80 and for many helpful comments. I am also indebted to Professor J. A. Liebreich and the Reading (PA) Area Community College for helpful advice and for allowing me to use the computer laboratory to check the programs on the Apple II Plus and the Apple IIe. Finally I wish to thank Mr. Terry Phelps for helpful advice on the operation of the TRS-80.

1

Introduction

This chapter introduces the reader to computer programming in BASIC (Beginner's All-purpose Symbolic Instruction Code). This language, developed by J. G. Kemeny and T. E. Kurtz at Dartmouth College in the 1960s, is used with almost all microcomputers and with many larger computers. No attempt is made in this book to give a comprehensive treatment of the BASIC language because the details of the language vary from one computer model to another. Instead, we use a simple version of BASIC that works successfully with almost any popular model of microcomputer that is suitable for scientific programming. Special statements and operations that work with only one or two models are avoided. The problems considered in this chapter have been selected both because they are of interest in themselves and because they illustrate important programming techniques.

1-1. Elements of the BASIC Language

As a very simple example of programming, we consider the problem of solving the equation

$$y = x + 3 \tag{1-1}$$

with $x = 2$. The BASIC program is

```
10 X=2
20 Y=X+3
30 PRINT Y
```

The lines are typed into the computer exactly as they are written. After each line is typed in, it is entered into the memory of the computer by pressing the ENTER key. (Some computers have a RETURN key instead of an ENTER key.) All characters are capitals. Procedures for correcting mistakes (editing) are not discussed here because they vary from one model to another. The best source of information on editing is the appropriate manufacturer's manual.

The lines at the left are line numbers. In BASIC programming, lines are not usually numbered consecutively; it is customary to choose line numbers that are multiples of 10. This system makes it easy to insert additional lines if it becomes necessary to amend a program later. One program line is not necessarily limited to one physical line on the screen; with most microcomputers, a program line may occupy several lines on the screen.

In this program, lines 10 and 20 are assignments. An equal sign in BASIC assigns the value of the expression on the right to the variable on the left. Although lines 10 and 20 of the sample program are valid algebraic equations, an assignment in general may or may not represent a valid algebraic equation. For example, the line J = J + 1 is a legitimate BASIC assignment, but it is meaningless as an algebraic equation.

To run the sample program, type RUN and press the ENTER key. The number 5 then appears on the screen. The calculation is made by lines 10 and 20; line 30 prints the result.

Any letter from A to Z may be used as a variable name. All of the commonly used versions of BASIC also allow variable names with two characters. In two-character variable names, the first character must be a letter; the second may be either a letter or a digit. Thus XN and X1 are legitimate variable names; these program variables correspond to x_n and x_1 in ordinary algebra. Longer names may be used, but, in many versions of BASIC for microcomputers, they are truncated to the first two characters. Thus, for example, ALPHA may be used as a variable name, but many microcomputers will abbreviate this internally to AL. In this book we use short names; long names are inconvenient because they necessitate a great deal of typing. Thus, for example, in programs we will use the Greek letters XI, NU, and PHI, but not LAMBDA or EPSILON.

In the original Dartmouth College BASIC of the 1960s, an assignment had to be introduced by the statement LET. Also, the statement END had to appear at the end of every program. These two requirements have disappeared from virtually all current versions of BASIC, at least for microcomputers, so we omit them from this book.

In the original Dartmouth College BASIC, only one statement could appear in each program line. Many and perhaps most current versions of BASIC allow multiple statements on a line, usually separated by colons. However, this flexibility is not universal. Throughout most of this book we shall follow the conservative practice of using only one statement on each line.

Line 30 of the sample program is a PRINT statement. This prints the value of the indicated variable on the screen. A simple algebraic expression may be used in a PRINT statement. For example, we could condense the program to

```
10  X=2
20  PRINT X+3
```

It is also possible to use words or the names of variables in the PRINT statement, by enclosing them in quotation marks. Anything that is enclosed in quotation marks will appear exactly as it is typed. When a variable or algebraic expression is typed without quotation marks, its numerical value is printed. Thus we may revise line 30 of the original program to

```
30  PRINT "Y=";Y
```

The result now reads

```
Y= 5
```

An expression that is enclosed in quotation marks is known as a *string.* Items in a PRINT statement may be separated by either semicolons or commas. A semicolon does not insert any space between items. However, with most microcomputers, any number that is not in quotation marks is automatically followed by a blank space. Also, a positive number is preceded by a blank space. (The Apple is an exception; this computer does not insert spaces automatically.) For a negative number, this space is occupied by the minus sign. When two items are separated by commas, the second item is printed in the next zone on the screen or printout.

There are six arithmetic operators in the BASIC language. These are listed in order of decreasing priority. (Operations on the same line have the same priority.)

( )	aggregation
∧	exponentiation
* /	multiplication, division
+ −	addition, subtraction

There are no brackets or braces in BASIC; multiple levels of parentheses are used. Operations in parentheses are performed first, starting with the expressions inside the innermost parentheses. Exponentiations are per-

formed next, followed by multiplications and divisions. Additions and subtractions are performed last. Evaluations proceed from left to right.

A remark about the symbol for exponentiation may be desirable. In the original Dartmouth College BASIC, the symbol ↑ is used for exponentiation. However, in some current versions of BASIC, the symbol ↑ is used to move the cursor or scroll the display on the screen. The caret ∧ seems to be the most popular and least confusing symbol for exponentiation, and it is used in this book. Thus, for example, x^5 is written as X∧5. Other symbols are also in use, and the nomenclature is not always uniform even with respect to different computers made by the same manufacturer. For example, some Radio Shack models use ↑, while others use [. Any reader whose computer uses a symbol other than ∧ can easily make the appropriate changes in the programs.

Not all calculations made with a computer necessarily have to be programmed. It is possible to obtain results immediately after the lines are entered. This is accomplished by entering the appropriate instructions without line numbers. Returning to the first program of this section, we enter

```
X=2
Y=X+3
PRINT Y
```

The result 5 is displayed immediately after the last line is entered. This mode of operation is known as the prompt mode, the immediate mode, the command mode, or the calculator mode. An instruction that is used in the program mode is known as a *statement*. An instruction that is used in the prompt mode is known as a *command*. Thus RUN is a command. PRINT is a command in the present example, but in the earlier examples it is a statement.

As a second example of programming, we consider the problem of evaluating the polynomial

$$y = 3 - 5x + 2x^2 + x^3 \tag{1-2}$$

The most efficient way to evaluate this is to start by writing it in nested form as

$$y = 3 - x(5 - x(2 + x)) \tag{1-3}$$

We have used only parentheses instead of parentheses and brackets in order to make the algebraic equation look as much as possible like the program equation. We again choose the value $x = 2$. The BASIC program is

```
10  X=2
20  Y=3-X*(5-X*(2+X))
30  PRINT "X=";X,"Y=";Y
```

Again, we run the program by typing RUN and pressing the ENTER key. The display then appears as follows:

```
X= 2                    Y= 9
```

The result 9 can easily be verified directly.

A program is typically used to obtain a number of results with different values of the independent variable. If this is done with the foregoing programs, it is necessary to type a new line 10 each time. The INPUT statement provides a more convenient way of handling this problem. We rewrite the last program as

```
10  INPUT X
20  Y=3-X*(5-X*(2+X))
30  PRINT "X=";X,"Y=";Y
```

When the execution reaches the INPUT statement, the computer stops and displays a question mark on the screen. The operator then enters the appropriate number, and the computer proceeds to execute the program. By running the program repeatedly with different values of x, we obtain the results shown in the following table:

x	-2	-1	0	1	2	3	4
y	13	9	3	1	9	33	79

The INPUT statement is less flexible than the PRINT statement. In most versions of BASIC, the input must be a number; algebraic expressions may not be used. However, it is permissible to include two or more items in the same input line. For example, the line

```
10  INPUT A,B,C
```

is acceptable. The numbers are entered together in the same order in which the variables appear, separated by commas.

The foregoing program is easy to use, but it is necessary to enter RUN for each desired result. This can be avoided by modifying the program as follows:

```
10  INPUT X
20  Y=3-X*(5-X*(2+X))
30  PRINT "X=";X,"Y=";Y
40  GOTO 10
```

The statement GOTO followed by a line number transfers the execution of the program to the beginning of the line indicated. Each time an evaluation is completed, execution returns to line 10 to call for further input.

Hence we type RUN only the first time the program is run. Thereafter results are obtained by simply entering each value of x.

A difficulty arises when we have obtained all of the desired results and are ready to proceed to some other problem; the computer is still waiting for the next value of x. One way to break a perpetual cycle is to turn off the computer. However, it is usually more convenient to press the BREAK key. Every computer has a key or combination of keys that performs this function. On most computers, it is called the BREAK key. On the Apple IIe, the same thing is accomplished by simultaneously pressing the CTRL (control) and RESET keys. On the Commodore 64, the procedure is to hold down the RUN/STOP key and hit the RESTORE key.

Programs are often organized to print a blank line between successive items of output. A blank line is generated by using a PRINT statement, followed by nothing. With this modification, the program becomes

```
10 PRINT
20 INPUT X
30 Y=3-X*(5-X*(2+X))
40 PRINT "X=";X,"Y=";Y
50 GOTO 10
```

It is possible to include a prompting message with the INPUT statement. The format is similar to that of the PRINT statement. With this modification, the program becomes

```
10 PRINT
20 INPUT "X=";X
30 Y=3-X*(5-X*(2+X))
40 PRINT "Y=";Y
50 GOTO 10
```

When the execution reaches line 20, the line X= or X=? appears on the screen. (Some, but not all, computers omit the question mark when a prompting message is used.) The operator enters a number, say 2. Line 30 then calculates the result y=9 and line 40 prints it. The final display looks like this:

```
X= 2
Y= 9
```

The first line is due to the INPUT statement, and the second is due to the PRINT statement. (With some computers, the first line may read X=? 2.) Although the primary purpose of a prompting message is to remind the operator what data to enter, it also serves to display the input data on the screen. We now need only y in the PRINT statement instead of x and y.

The INPUT prompting message varies among computers. A few computers require a colon instead of a semicolon; also, a few computers do not allow prompting messages. We also point out that the present discussion applies only to the screen display, like everything in this book. Instructions for printers vary from model to model, and the best source of information is the appropriate manufacturer's manual.

The statement REM (remark) at the beginning of a program line instructs the computer that the line is not used in the calculations. This makes it possible to insert an explanatory remark. For example, we might choose to add a title to the last program. Thus

```
 1  REM: EVALUATION OF NESTED POLYNOMIAL
10  PRINT                      Generates blank line.
20  INPUT "X=";X               Calls for value of x.
30  Y=3-X*(5-X*(2+X))          Calculates y.
40  PRINT "Y=";Y               Prints result.
50  GOTO 10                    Returns for new input.
```

Remarks increase the length of a program, thus affecting both the amount of typing and the space that the program occupies in the computer memory. In this book we include very few remarks in the programs, with the exception of titles. Explanatory notes are given directly to the right of the programs, as shown above. Notes have not been necessary for the simple programs of this section, but they will be helpful for the more complicated programs considered later. Any reader who wishes to do so may integrate the notes into the programs as remarks. By using line numbers that are not multiples of 10 for remarks, it is possible to do so without reworking the entire programs.

Strings (quotations) may be assigned names and handled like ordinary variables. The name of a string must end with a dollar sign. A string may be referenced in a PRINT statement or in an INPUT statement. Consider the following simple program:

```
10  A$="STRING"
20  PRINT A$
```

When the program is run, the word STRING appears on the screen.

A string must always be enclosed in quotation marks when used in an assignment statement. However, most versions of BASIC allow a simple string to be used without quotation marks in response to an INPUT statement. Restrictions on punctuation vary among different versions of BASIC. The reader is advised to consult his manual for details.

The BASIC language includes a number of scientific functions that are built into the computer. Ten built-in functions are provided with any model suitable for scientific programming. Each function is denoted by a three-letter name, followed by the argument in parentheses.

There are three trigonometric functions, namely

The argument is in radians. Most computers provide only one inverse trigonometric function: the arc tangent, denoted as

ATN(X)

There are also an exponential function e^x and a natural logarithm. These are

EXP(X) LOG(X)

The absolute value function

ABS(X)

returns the absolute value of the argument. The signum function

SGN(X)

has the values

$$\begin{aligned} \text{sgn } x = 1 \quad &\text{if} \quad x > 0 \\ = 0 \quad &\quad = 0 \\ = -1 \quad &\quad < 0 \end{aligned}$$

The square root function

SQR(X)

returns the square root of a nonnegative argument. The integer function

INT(X)

returns the value of the largest integer that does not exceed the argument.

The argument of any of these functions may be a number, a variable, a simple algebraic expression, or another function. Thus an expression such as EXP(SIN(X)) is legitimate.

1-2. Discontinuous Functions: The IF-THEN Statement

It is often necessary to evaluate a function that is given by one formula over one part of an interval and by a different formula over another part of the interval. The IF-THEN statement is very useful for problems of this type. The statement consists of the words IF and THEN, separated by an equation or inequality and followed by a line number. If the relation

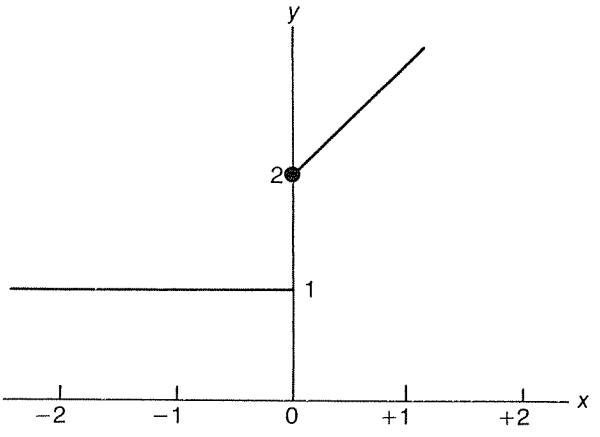


FIG. 1-1

is satisfied, the execution of the program is transferred to the beginning of the line indicated. If it is not, execution continues with the next program line. Consider the function

$$y = 1,\ x < 0 \qquad y = x + 2,\ x \geq 0 \tag{1-4}$$

which is sketched in Fig. 1-1. The program is

```
10  PRINT                      Generates blank line.
20  INPUT X                    Calls for value of x.
30  IF X<0 THEN 60             Transfers execution if x<0.
40  Y=X+2                      Calculates y if x≥0.
50  GOTO 70                    Transfers to PRINT statement.
60  Y=1                        Calculates y if x<0.
70  PRINT "X=";X,"Y=";Y        Prints x and y.
80  GOTO 10                    Returns for further input.
```

Line 10 creates a blank line between successive sets of output. Line 20 is an INPUT statement that calls for the value of x. Line 30 is an IF-THEN statement. If $x < 0$, execution is transferred to line 60, where y is assigned the value 1. Otherwise execution continues with line 40, which sets y equal to $x + 2$. In either case, execution then proceeds to lines 70 and 80. Line 70 prints the results and line 80 sends the execution back to the beginning to call for new input. Some numerical results follow:

x	−2	−1	0	1	2	3
y	1	1	2	3	4	5

This program can be rewritten a little more concisely, as follows:

```
10  PRINT
20  INPUT X
```

```
30  Y=1
40  IF X<0 THEN 60
50  Y=X+2
60  PRINT "X=";X,"Y=";Y
70  GOTO 10
```

The evaluation starts by letting $y = 1$. If $x < 0$, this is the final result. If it happens that $x \geq 0$, this result is overwritten by line 50. The results are identical to those given by the first program.

Some versions of BASIC allow an equation to be used after THEN instead of a line number. The statement ELSE may also be added to cover the case that is excluded by the relation between IF and THEN. For a computer that uses this type of BASIC, a third program is

```
10  PRINT                                  Generates blank line.
20  INPUT X                                Calls for value of x.
30  IF X<0 THEN Y=1 ELSE Y=X+2             Calculates y.
40  PRINT "X=";X,"Y=";Y                    Prints x and y.
50  GOTO 10                                Returns for further
                                           input.
```

This type of programming is not used in this book, because many versions of BASIC do not allow it. Actually there is very little difference in length between the last two programs. The last program combines three short lines into one long line.

We have used the operators = and < without comment. There are six relational operators in BASIC. Each appears directly below its corresponding algebraic operator, as follows:

```
=    ≠   >   ≥   <   ≤
=   <>   >   >=   <   <=
```

Relational expressions are sometimes useful. An expression such as (A = B) has the value 0 if the relation is false. If the relation is true, its value is 1 or −1, depending on the version of BASIC used by the computer. We now write a fourth version of the program using a relational expression:

```
10  PRINT
20  INPUT X
30  Y=1+(X+1)*ABS(X>=0)
40  PRINT "X=";X,"Y=";Y
50  GOTO 10
```

Whenever a relational expression appears in this book, we use the absolute value in order to eliminate the ambiguity in sign. Programs written in this way can be used on almost any popular model of microcomputer

without editing. If a program is to be used on one model only, any user who wishes to do so may delete the ABS statement and insert the appropriate sign.

The foregoing methods work well when the transition point is on one of the branches of the curve. If the transition point is elsewhere, as sketched in Fig. 1-2, three equations are necessary to define y as a function of x, and the program becomes a little more complicated. We modify Eq. 1-4 to let $y = 1.5$ at $x = 0$. A program follows:

```
10  PRINT                       Generates blank line.
20  INPUT X                     Calls for value of x.
30  Y=1                         Calculates y.
40  IF X<0 THEN 90              Transfers execution if x<0.
50  IF X=0 THEN 80              Transfers execution if x=0.
60  Y=X+2                       Recalculates y if x>0.
70  GOTO 90                     Transfers to PRINT statement.
80  Y=1.5                       Recalculates y if x=0.
90  PRINT "X=";X,"Y=";Y         Prints x and y.
100 GOTO 10                     Returns for further input.
```

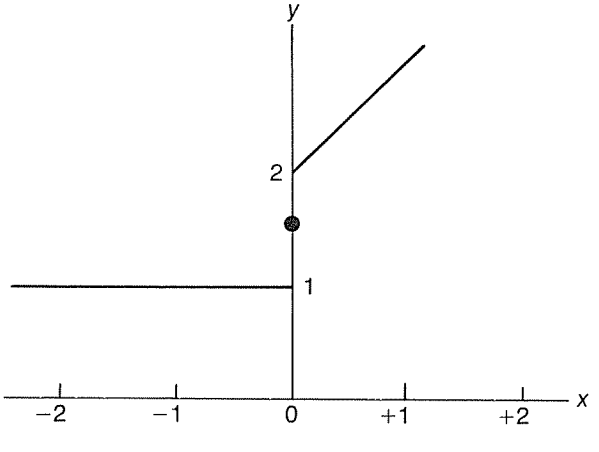
The results are identical to those found previously except that $y = 1.5$ when $x = 0$.

With relational expressions, the program becomes

```
10 PRINT
20 INPUT X
30 Y=1+(X+1)*ABS(X>0)+ABS(X=0)/2
40 PRINT "X=";X,"Y=";Y
50 GOTO 10
```

The six lines 30 through 80 are now combined into the single line 30.

FIG. 1-2



The last two methods work for any value of y at the transition point. When the transition point is midway between the two branches, the simplest possible program is obtained by using the signum function. With this function, the equation for y becomes

$$y = \tfrac{1}{2}\,[x + 3 + (x + 1)\text{sgn } x]$$

The program is

```
10  PRINT
20  INPUT X
30  Y=(X+3+(X+1)*SGN(X))/2
40  PRINT "X=";X,"Y=";Y
50  GOTO 10
```

The results are identical to those found with the preceding program.

The ON-GOTO statement is occasionally used instead of multiple IF-THEN statements. This consists of the words ON and GOTO, separated by a variable or algebraic expression and followed by a group of line numbers. The expression is evaluated and truncated to an integer. Let the result be n. Then the execution is transferred to the nth designated line number. With this statement, the program becomes

```
10  PRINT                          Generates blank line.
20  INPUT X                        Calls for value of x.
30  ON SGN(X)+2 GOTO 40,60,80      Transfers execution to appropriate line.
40  Y=1                            Calculates y if x<0.
50  GOTO 90                        Transfers to PRINT statement.
60  Y=1.5                          Calculates y if x=0.
70  GOTO 90                        Transfers to PRINT statement.
80  Y=X+2                          Calculates y if x>0.
90  PRINT "X=";X,"Y=";Y            Prints x and y.
100 GOTO 10                        Returns for further input.
```

Results are again identical to those found with the preceding programs.

It is sometimes necessary to write a program for a periodic function. The INT (integer) statement is often useful for problems of this type. Consider the periodic function sketched in Fig. 1-3, which represents the equation

$$y = x + 1 \qquad 0 \le x < 2 \tag{1-5}$$

and is repeated indefinitely in the direction of positive (or negative) x. The program is

```
10  PRINT
20  INPUT X
```

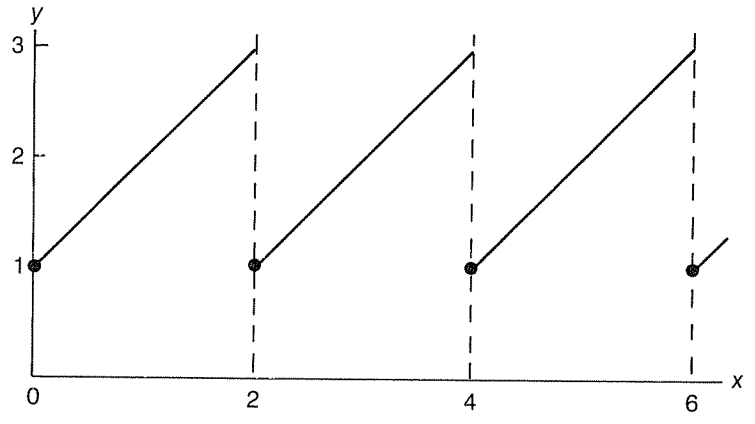


FIG. 1-3

```
30  Y=X-2*INT(X/2)+1
40  PRINT "X=";X,"Y=";Y
50  GOTO 10
```

If the transition points are not on the branches of the curve to the right of the discontinuities as shown in Fig. 1-3, the foregoing program must be modified. Let $y = 2$ at the transition points. Then the program becomes

```
10  PRINT
20  INPUT X
30  U=X-2*INT(X/2)
40  Y=U+1+ABS(U=0)
50  PRINT "X=";X,"Y=";Y
60  GOTO 10
```

The program is self-explanatory. We have inserted an intermediate step in the evaluation of y.

1-3. The FOR-NEXT Loop

Repetitive calculations occur very frequently in scientific applications. The FOR-NEXT loop is very useful for problems of this type. To introduce this technique, we consider the problem of finding the sum of the first n integers, that is, we evaluate the sum

$$S = 1 + 2 + 3 + \ldots + n \tag{1-6}$$

The program is

```
    10  PRINT                    Generates blank line.
13  20  INPUT N                  Calls for value of n.
```

```
30  S=0                          Initializes S.
40  FOR J=1 TO N
50  S=S+J                        } Calculates S.
60  NEXT J
70  PRINT "N=";N,"S=";S          Prints n and S.
80  GOTO 10                      Returns for further input.
```

The program is straightforward. Line 10 generates a blank space between successive items of output. Line 20 calls for the input n. Line 30 assigns the initial value 0 to S, the partial sum of the series on the right side of Eq. 1-6. Lines 40 through 60 constitute the FOR-NEXT loop. When the FOR statement is reached in line 40, the value of j is set equal to 1. Execution then continues to line 60. The NEXT statement adds 1 to the value of j and compares the result with the upper limit n of line 40. If the incremented value of j is not greater than n, execution returns to the line immediately following the FOR-TO statement (line 50), and the cycle is repeated. After the last (nth) cycle, execution continues with the next program line (line 70). The n cycles constitute a loop. Line 70 prints the results, and line 80 returns the execution of the program to the beginning in preparation for further input.

There is a slightly more general version of the FOR-TO statement than the one given in line 40. The increment does not necessarily have to be 1. An optional STEP statement may be added. Thus, for example, we might have

```
40  FOR J=1 TO 10 STEP 3
```

The running variable j now assumes the values 1, 4, 7, 10. The step size may be either positive or negative. We also point out that the upper limit does not have to coincide exactly with one of the values of the running variable. The running variable takes on whatever values are possible without exceeding the upper limit. Thus, for the case

```
40  FOR J=2 TO 15 STEP 4
```

the running variable j assumes the values 2, 6, 10, 14.

In exactly the same way we write a program to evaluate the factorial

$$n! = 1 \cdot 2 \cdot 3 \ldots n \tag{1-7}$$

The program is

```
10  PRINT
20  INPUT N
30  P=1
40  FOR J=1 TO N
50  P=J*P
```

```
60 NEXT J
70 PRINT "N=";N,"N!=";P
80 GOTO 10
```

This program works in the same way as the first program of this section. The parameter P is the partial product on the right side of Eq. 1-7.

When the running variable runs from 1 to n, where n is a positive integer and the step size is 1, the cycle runs n times. However, the case $n = 0$ sometimes presents a problem. In standard BASIC the upper limit is checked before the cycle runs. If this is less than the initial value of the running variable, the cycle does not run (unless the step size is negative). Hence the loop operates correctly in the case $n = 0$; the cycle runs zero times. Most large computers and some microcomputers operate in this way. However, in the versions of BASIC used by most microcomputers, no check is made until the NEXT statement is reached at the end of the loop. Hence the cycle always runs at least once, even if $n = 0$. The result obtained from the first program when $n = 0$ should be 0; actually it may be either 0 or 1, depending on the computer used. Usually this does not matter much; the correct result for $n = 0$ is obvious by inspection, and a computer evaluation is not necessary. However, a program of the present type sometimes appears as a segment in a more complicated program in which the case $n = 0$ may not be trivial. In this case the evaluation should be valid for any integral value of $n \geq 0$. This is accomplished by inserting an IF-THEN statement to bypass the loop when $n = 0$. Programs written in this way will work with either type of computer. An even simpler remedy for the first program is to change the lower limit in line 40 from 1 to 0.

No matter what type of computer is used, the second program leads to the correct result $0! = 1$. Since the first cycle consists of a multiplication by 1, it does not matter whether it runs.

The FOR-NEXT loop is very useful in summing series. Consider

$$S = \sum_{j=1}^{\infty} \frac{1}{j^2 + 1} = \frac{1}{2} + \frac{1}{5} + \frac{1}{10} + \frac{1}{17} + \cdots \tag{1-8}$$

Some preliminary transformations are necessary, since the series converges so slowly that it would be impractical to evaluate it as it stands. After the first few terms, the general term is essentially $1/j^2$, and the error is of order $1/n$, where n is the number of terms considered. It would be necessary to consider many thousands of terms to obtain a reasonably accurate value of the sum. The convergence can be greatly improved by using the known fact that

$$\sum_{j=1}^{\infty} \frac{1}{j^2} = \frac{\pi^2}{6}$$

Then the desired result becomes

$$S=\frac{\pi^2}{6}-\sum_{j=1}^{\infty}\frac{1}{j^2}+\sum_{j=1}^{\infty}\frac{1}{j^2+1}$$

$$=\frac{\pi^2}{6}-\sum_{j=1}^{\infty}\frac{1}{j^2(j^2+1)}$$

The general term is now essentially $1/j^4$, and the error is of order $1/n^3$. A further improvement is obtained by using the formula

$$\sum_{j=1}^{\infty}\frac{1}{j^4}=\frac{\pi^4}{90}$$

We now have

$$S=\pi^2\left[\frac{1}{6}-\frac{\pi^2}{90}\right]+\sum_{j=1}^{\infty}\frac{1}{j^4(j^2+1)} \tag{1-9}$$

The general term is essentially $1/j^6$, and the error is of order $1/n^5$. The program follows:

```
10  PI=4*ATN(1)                    Calculates π.
20  S=PI*PI*(1/6-PI*PI/90)         Evaluates constant in equation 1-9.
30  INPUT "N=";N                   Calls for value of n.
40  FOR J=1 TO N                   }
50  S=S+1/J^4/(J*J+1)              } Calculates S.
60  NEXT J                         }
70  PRINT "S=";S                   Prints result.
```

Line 10 calculates π. (With any computer such as the Commodore 64 that has a built-in constant for π, this may be used instead, although the present program can be used as it stands.) Line 20 evaluates the constant on the right side of Eq. 1-9. Line 30 calls for the number of terms n. Lines 40 through 60 constitute a FOR-NEXT loop that sums the series on the right side of Eq. 1-9. Line 70 prints the result. Results found by the program with several values of n are

n	10	20	30	40
S	1.0766725	1.07667399	1.07667404	1.07667405

In a problem of this type it is highly desirable to make several approximations, because the easiest way to estimate the accuracy of the result is to compare successive approximations. It is not necessary to repeat the entire calculation each time. The following program is set up so that each evaluation begins with the result of the next lower approximation.

```
10  PI=4*ATN(1)                      Calculates π.
20  S=PI*PI*(1/6 - PI*PI/90)         Evaluates constant in Eq. 1-10.
30  J=1                              Initializes j.
40  PRINT                            Generates blank line.
50  INPUT "N=";N                     Calls for value of n.
60  FOR J=J to N                     }
70  S=S+1/J^4/(J*J+1)                } Calculates S.
80  NEXT J                           }
90  PRINT "S=";S                     Prints result.
100 GOTO 40                          Returns for further input.
```

Numerical results are identical to those given by the previous program.

The nested format of Sec. 1-1 is very useful for summing certain types of series. Consider

$$S = 1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \cdots \tag{1-10}$$

In nested form, this becomes

$$S = 1 + \frac{1}{3}\left[1 + \frac{2}{5}\left(1 + \frac{3}{7}(1 + \cdots)\right)\right]$$

The program follows:

```
10  PRINT                            Generates blank line.
20  INPUT "N=";N                     Calls for value of n.
30  S=1                              Initializes S.
40  FOR J=N-1 TO 1 STEP -1           }
50  S=1+J/(2*J+1)*S                  } Calculates S
60  NEXT J                           }
70  PRINT "S=";S                     Prints result.
80  GOTO 10                          Returns for further input.
```

The program is set up to obtain repeated approximations. Line 10 is an optional line that generates a blank space between successive sets of data. Line 20 calls for the desired number of terms. The starting value 1 in line 30 is the 1 at the extreme right of the nested equation. Lines 40 through 60 constitute a FOR-NEXT loop that sums the nested series. The calculation proceeds from right to left, ending with the first 1 on the right side of the equation. The $n - 1$ cycles give the sum of n terms of the original series. Line 70 prints the result. Line 80 is an optional line that makes it possible to obtain further approximations without entering RUN each time. (However, with the nested format it is not possible to reuse previous approximations.) We obtain the following results:

n	10	20	30
S	1.570289	1.57079 5964	1.57079 6327

The last result is the exact value $\pi/2$, correct to ten significant figures.

We have used the FOR-NEXT loop to deal with repetitive steps within a calculation. A loop is also useful for a case in which an entire calculation must be repeated a number of times. We return to Eq. 1-3 of Sec. 1-1, which is

$$y = 3 - x(5 - x(2 + x))$$

Again, we desire the values of y corresponding to integral values of x from -2 to 4. Instead of running the program of Sec. 1-1 repeatedly, we now use a FOR-NEXT loop. The revised program is

```
10  PRINT " X"," Y"
20  FOR X=-2 TO 4
30  Y=3-X*(5-X*(2+X))
40  PRINT X,Y
50  NEXT X
```

The program is set up so that the results appear as the following table:

X	Y
−2	13
−1	9
0	3
1	1
2	9
3	33
4	79

The leading spaces inside the quotation marks in line 10 align the characters X and Y with the first digits of the numbers that appear below them. For any computer such as the Apple that does not print leading spaces with numeric output, these may be omitted.

The foregoing solution works because the values of the argument x are uniformly spaced; therefore the running variable in the FOR-NEXT loop can be used as the argument x. Now suppose that we require the values of y corresponding to a number of irregularly spaced values of x—say $x = -2$, 0, .5, 2.3. The READ and DATA statements are useful for this problem. These statements provide the third method of introducing data into a computer. (The first two are the assignment and the INPUT statement.) The numerical values are inserted into a DATA statement and assigned to the appropriate variable or variables by a READ statement. With these statements, the program becomes

```
10  PRINT " X"," Y"
20  FOR J=1 TO 4
30  READ X
40  Y=3-X*(5-X*(2+X))
50  PRINT X,Y
60  NEXT J
70  DATA -2,0,.5,2.3
```

The results appear as the following table:

X	Y
−2	13
0	3
.5	1.125
2.3	14.247

On the first cycle, the READ statement reads the first entry in the data line. On each successive cycle, it moves one step to the right to read a subsequent data entry. Two or more data entries may be read by one READ statement; for example, in line 30 we might have READ X,Y,Z. However, the total number of items read may not exceed the total number of data entries, unless the data are restored. The RESTORE statement, which is used later, causes the READ statement to start over and read the data from the beginning. The data may be spread over two or more DATA statements; the READ statement starts at the beginning and proceeds through all the data lines. Also, data lines may appear anywhere in a program, but it is preferable to place all of them together, either at the beginning or at the end.

A very important practical example of this programming technique occurs in the problem of fitting a straight line through a set of experimental points, as shown in Fig. 1-4. This problem occurs repeatedly in science and engineering. Suppose that we have experimental data for a set of points as follows,

$x_1 \quad x_2 \quad x_3 \quad \ldots \quad x_n$
$y_1 \quad y_2 \quad y_3 \quad \ldots \quad x_n$

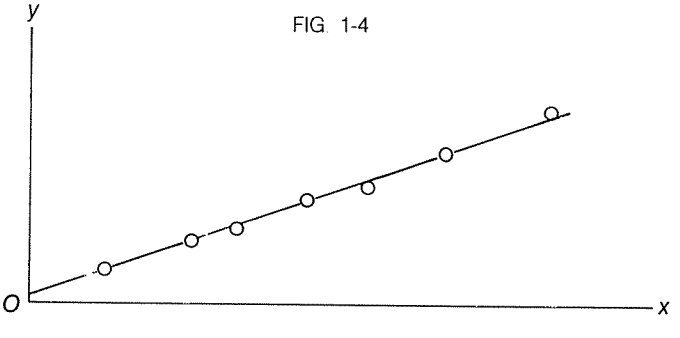


FIG. 1-4

and we wish to find the best values of the coefficients a and b in the equation

$$y = ax + b$$

The most commonly used procedure is the method of least squares. We find the values of a and b that make the sum of the squares of the errors a minimum. In other words, we minimize the expression

$$\Sigma(y_j - ax_j - b)^2 = (y_1 - ax_1 - b)^2 + (y_2 - ax_2 - b)^2 + \ldots$$

(All the summations in this analysis run from $j = 1$ to n; we do not write the limits each time.) Setting the partial derivatives of this expression with respect to a and b equal to zero leads to the equations

$$\Sigma x_j(y_j - ax_j - b) = 0$$
$$\Sigma(y_j - ax_j - b) = 0$$

which can be rewritten as

$$a\Sigma x_j^2 + b\Sigma x_j = \Sigma x_j y_j$$
$$a\Sigma x_j + nb = \Sigma y_j$$

By solving for a and b, we find that

$$a = \frac{n\Sigma x_j y - \Sigma x_j \Sigma y_j}{n\Sigma x_j^2 - (\Sigma x_j)^2} = \frac{nv - st}{nu - s^2}$$
$$b = \frac{1}{n}(\Sigma y_j - a\Sigma x_j) = \frac{t - sa}{n}$$

The following substitutions have been made:

$$s = \Sigma x_j \qquad t = \Sigma y_j$$
$$u = \Sigma x_j^2 \qquad v = \Sigma x_j y_j$$

A program follows for the data:

x_j	1.0	2.1	2.7	3.8	4.6	5.6	7.0
y_j	.43	.85	1.03	1.42	1.59	2.04	2.53

```
 1  REM: ANALYSIS OF EXPERIMENTAL DATA
10  N=7                                  Assigns value of n.
20  FOR J=1 TO N                       ⎫
30  READ X,Y                           ⎪
40  S=S+X                              ⎪
50  T=T+Y                              } Calculates S,T,U,V.
60  U=U+X*X                            ⎭
```

```
 70  V=V+X*Y                                  ┐
 80  NEXT J                                   ┘
 90  A=(N*V-S*T)/(N*U-S*S)                    ┐ Calculates a and b.
100  B=(T-S*A)/N                              ┘
110  PRINT "A=";A                             ┐ Prints results.
120  PRINT "B=";B                             ┘
130  DATA 1.0,.43,2.1,.85,2.7,1.03,           ┐
     3.8,1.42,4.6,1.59,5.6,2.04,7.0,2.53      ┘ Data line.
```

Line 1 is the title. Line 10 assigns the value of n. Lines 20 through 80 constitute a FOR-NEXT loop that reads the values of the x_js and y_js from the data line and calculates s, t, u, and v. Lines 90 and 100 calculate a and b, and lines 110 and 120 print the results. Line 130 is the data line. Lines 10 and 130 are filled in by the user each time the program is run. With the present sequence, we obtain the results

$$a = .344 \qquad b = .094$$

One further comment is needed. We have not assigned the initial values of zero to the variables S, T, U, and V. With most computers this is not necessary because the RUN command automatically sets all variables equal to zero. For any computer that does not have this feature, the variables S, T, U, and V must be initialized. This can be done by inserting assignment lines between lines 10 and 20.

1-4. The Subroutine: The User-Defined Function

A program often performs the same action in several places. Instead of writing the same line repeatedly, it is usually more convenient to write it only in one place, as a subroutine. This is accomplished by using the GOSUB and RETURN statements. The GOSUB statement, followed by a line number, transfers the execution to the line specified, which is the subroutine. Execution then proceeds until the RETURN statement is reached, when it returns to the original point following the GOSUB statement. To illustrate the use of a subroutine, we consider the problem of evaluating

$$y = f(x_1) - 3f(x_2) + 2f(x_3) \tag{1-11}$$

where

$$f(x) = (3 - 5x + 2x^2 + x^3)^{1/2} \tag{1-12}$$

The program is

```
10  INPUT X1,X2,X3                    Calls for values of xs.
20  X=X1
30  GOSUB 140                         } Calculates f(x1).
40  F1=F
50  X=X2
60  GOSUB 140                         } Calculates f(x2).
70  F2=F
80  X=X3
90  GOSUB 140                         } Calculates f(x3).
100 F3=F
110 Y=F1-3*F2+2*F3                    Calculates y.
120 PRINT Y                           Prints y.
130 END                               END statement.
140 F=SQR(3-X*(5-X*(2+X)))            Subroutine.
150 RETURN                            RETURN statement.
```

Line 10 calls for the values of x_1, x_2, and x_3. Lines 20 through 40 calculate $f(x_1)$ by using the subroutine of line 140, which represents Eq. (1-12). (The nested format is used.) Lines 50 through 70 calculate $f(x_2)$, and lines 80 through 100 calculate $f(x_3)$. Line 110 represents Eq. 1-11. Line 120 prints the result. Line 140 is the subroutine and line 150 is the RETURN statement. The END statement of line 130 is necessary to prevent the execution from running into line 140 after line 120 has been executed. In most versions of BASIC, either STOP or END may be used in line 130. The END statement is preferable because, with most microcomputers, the STOP statement generates a BREAK message. If we expect to continue with other sets of input data, GOTO 10 may be preferable to either STOP or END in line 130.

As a numerical example, we let $x_1 = 1$, $x_2 = 2$, $x_3 = 3$. The result is $y = 3.489125293$.

A user-defined function can often be used as an alternative to a subroutine. The rules for defining and using a user-defined function vary considerably among different versions of BASIC. We shall adopt a narrow version that works on almost any popular model of computer or microcomputer with this feature. First the function is defined by using the statement DEF. The name of the function follows; it consists of three letters, the first two of which are FN. This is followed by the argument in parentheses and then an equal sign, after which the function is written out. After a function has been defined in this way, it may be used later in the program in exactly the same way as a built-in function. A program follows for the problem of Eqs. 1-11 and 1-12, this time employing a user-defined function.

```
10  INPUT X1,X2,X3
20  DEF FNF(X)=SQR(3-X*(5-X*(2+X)))
30  Y=FNF(X1)-3*FNF(X2)+2*FNF(X3)
40  PRINT Y
```

Results given by this program are identical to those given by the first program.

The subroutine and the user-defined function are often but not always interchangeable. The user-defined function is suitable only for a simple function that can be defined in a single line. The subroutine can be used for a function of any degree of complexity, since any number of lines may be used in a subroutine. In this book, we do not make much use of the user-defined function, because some microcomputers, such as the TRS-80 cassette models, do not have this feature.

1-5. Recurrence Formulas; Legendre Polynomials

Recurrence formulas occur in many practical problems, such as the numerical solutions of differential equations. As a simple example, consider

$$x_{j+1} = 3x_j - 2x_{j-1} + 1 \tag{1-13}$$

Two initial values are specified, say $x_1 = 1$ and $x_2 = 2$. It is required to find the subsequent terms of the sequence of x_js through x_n. A program follows:

```
  1 REM: SECOND ORDER RECURRENCE FORMULA
 10 READ X1,X2                 Reads initial values.
 20 INPUT N                    Calls for value of n.
 30 PRINT X1                   }
 40 PRINT X2                   } Prints initial values.
 50 FOR J=3 to N               }
 60 X3=3*X2-2*X1+1             }
 70 X1=X2                      } Calculates and prints
 80 X2=X3                      } subsequent xj s.
 90 PRINT X3                   }
100 NEXT J                     }
110 DATA 1,2                   Data line for initial values.
```

Line 1 is the title. Line 10 reads the initial values x_1 and x_2 from the data line 110. Line 20 calls for the value of n, the desired number of terms. Lines 30 and 40 print the initial values x_1 and x_2. Lines 50 through 100 constitute a FOR-NEXT loop that calculates subsequent terms of the sequence. Line 60 represents Eq. 1-13. Lines 70 and 80 reassign the values of the x_js so that the last two values become x_1 and x_2 in preparation for the next cycle. Line 90 prints the result of each cycle. With $n = 8$, we obtain the sequence 1, 2, 5, 12, 27, 58, 121, 248. (It can be shown that the analytical solution is $x_j = 2^j - j$.) The same program can be used for any other second-order recurrence formula; only the equation line 60 and the data line 110 have to be changed.

The same method can be used to write a program for the Legendre

polynomials, which are considered in advanced calculus. The first few are

$$P_0(x) = 1 \tag{1-14a}$$

$$P_1(x) = x \tag{1-14b}$$

$$P_2(x) = \frac{1}{2}(3x^2 - 1) \tag{1-14c}$$

$$P_3(x) = \frac{x}{2}(5x^2 - 3) \tag{1-14d}$$

$$P_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3) \tag{1-14e}$$

$$P_5(x) = \frac{x}{8}(63x^4 - 70x^2 + 15) \tag{1-14f}$$

The general recurrence relation is

$$P_{n+1}(x) = \frac{1}{n+1}[(2n+1)xP_n(x) - nP_{n-1}(x)] \tag{1-15}$$

A program follows:

```
 1 REM: LEGENDRE POLYNOMIALS
10 PRINT                                  Generates blank line.
20 INPUT "ENTER N,X";N,X                  Calls for values of n and x.
30 IF N>1 THEN 60                         } Considers case n=0
40 P2=1+N*(X-1)                           } or n=1.
50 GOTO 130
60 P0=1                                   } Initializes variables.
70 P1=X
80 FOR J=1 TO N-1                         }
90 P2=((2*J+1)*X*P1-J*P0)/(J+1)           } Calculates Pn(x).
100 P0=P1                                 }
110 P1=P2                                 }
120 NEXT J
130 PRINT "N=";N,"X=";X                   Prints n and x.
140 PRINT "PN(X)=";P2                     Prints Pn(x).
150 GOTO 10                               Returns for further input.
```

Line 1 is the title. Line 10 skips a line between successive results. Line 20 calls for the values of n and x. Lines 30 through 50 take care of the special cases $n = 0$ and $n = 1$. The remainder of the program deals with the case $n \geq 2$. Lines 60 and 70 assign the appropriate values to $P_0(x)$ and $P_1(x)$. Lines 80 through 120 constitute a FOR-NEXT loop that calculates $P_n(x)$. Line 90 represents Eq. 1-15, and lines 100 and 110 reassign the values of the Ps in preparation for the next cycle. Lines

130 and 140 print the results, and line 150 sends the execution back to the beginning in preparation for further input.

1-6. Subscripted Variables

We have used subscripted variables in a number of algebraic equations. However, we have not yet used subscripted variables in the programs. Variables with one, two, three, or more subscripts may be used in BASIC. Subscripts in the program variables are indicated by parentheses; thus x_3 and $x_{1,2}$ become X(3) and X(1,2), respectively. The subscripts must be positive integers or zero. Variables may be used as subscripts. Subscripted variables are handled in exactly the same way as ordinary variables. The two applications that follow illustrate the use of subscripted variables.

At times it is necessary to obtain a value of a function from a table by interpolation. A good table usually makes it possible to obtain satisfactory accuracy for most engineering calculations by linear interpolation. This can easily be done in the prompt mode, and a program is not necessary. However, if high accuracy is required or if the available table gives results only for widely spaced values of the argument, higher-order interpolation is needed unless the argument coincides with one of the tabulated values. One commonly used procedure is known as Lagrange interpolation, in which the desired function is approximated by a polynomial of order $n - 1$, using data from n points. For linear interpolation ($n = 2$), the formula is

$$y = \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_1}{x_2 - x_1} y_2 \tag{1-16a}$$

For quadratic interpolation ($n = 3$), the formula is

$$y = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3 \tag{1-16b}$$

For cubic interpolation ($n = 4$), the formula is

$$y = \frac{(x - x_2)(x - x_3)(x - x_4)}{(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)} y_1 + \frac{(x - x_1)(x - x_3)(x - x_4)}{(x_2 - x_1)(x_2 - x_3)(x_2 - x_4)} y_2$$
$$+ \frac{(x - x_1)(x - x_2)(x - x_4)}{(x_3 - x_1)(x_3 - x_2)(x_3 - x_4)} y_3 + \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_4 - x_1)(x_4 - x_2)(x_4 - x_3)} y_4 \tag{1-16c}$$

The foregoing equations are clearly exact at the base points $x = x_j$. The base points do not have to be uniformly spaced, although they usually are when the method is used to interpolate in values from a table. The general formula for n base points is

$$y = \sum_{j=1}^{n} \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i} y_j \tag{1-17}$$

The program follows. It is set up to evaluate the Bessel function $J_0(x)$ at the point $x = 1.15$, using known values of $J_0(x)$ at the points $x = .9$, 1.0, 1.1, 1.2, 1.3, 1.4. These are taken from a table on page 390 of reference 1.

```
  1 REM: LAGRANGE INTERPOLATION
 10 INPUT "ENTER N,X";N,X
 20 FOR J=1 to N
 30 READ X(J),Y(J)
 40 NEXT J
 50 S=0
 60 FOR J=1 TO N
 70 U(J)=Y(J)
 80 FOR I=1 TO N
 90 IF I=J THEN 110
100 U(J)=U(J)*(X-X(I))/(X(J)-X(I))
110 NEXT I
120 S=S+U(J)
130 NEXT J
140 PRINT S
150 DATA 1.1,.7196220185,1.2,.6711327443,
    1,.7651976866,1.3,.6200859896,
    .9,.8075237981,1.4,.5668551204
```

Calls for values of n and x.

Reads tabular data.

Initializes S.

Calculates u_j

Calculates y.

Prints result.

Data lines.

Line 1 is the title. Line 10 calls for the values of n and x. Lines 20 through 40 read the values of the x_js and y_js from the data line. Line 50 assigns the initial value 0 to S, the partial sum on the right side of Eq. 1-17. The remainder of the program consists essentially of a nested FOR-NEXT loop. The inner loop of lines 80 through 110 evaluates the product u_j on the right side of Eq. 1-17, and the outer loop of lines 60 through 130 evaluates the sum. Line 140 prints the result, and line 150 contains the values of the x_js and y_js.

The correct result to ten significant figures is .6957197635. With $n = 2$, (linear interpolation), we obtain .6953773814, which is correct to three significant figures. With $n = 4$, the result is .6957193243, which is correct to six significant figures. With $n = 6$, we obtain .6957197628, which is correct to nine significant figures.

This program runs as it stands on almost any microcomputer in common use; only the data line 150 must be filled in for each application. However, a few computers, such as the TI-99/4, do not allow the same character to be used as both an ordinary variable and a subscripted variable. On any machine that has this limitation, a different name, such as xi, must be used for the unsubscripted x in lines 10 and 100.

It is sometimes necessary to use inverse interpolation, that is, to find the value of an independent variable corresponding to some specified value of the dependent variable. Lagrange interpolation is well suited to this problem, since the base points do not have to be equally spaced. We use exactly the same procedure as that just given, but we call the dependent variable x and the independent variable y.

The practical utility of higher-order interpolation is rather limited. Results can usually be obtained with better accuracy and less labor by evaluating the function directly. Programs for Legendre polynomials have been given in Sec. 1-5; programs for a number of other higher mathematical functions are given in Chapter 3. If it is desired to use a table, the best procedure is to find a good table with reasonably closely spaced values of the argument. Most of the tables of reference 1 give results only for very widely spaced values of the argument. They are very good for checking a program or for examples in which the argument can be chosen to fit the table, but they are not suitable for practical problems.

When a subscripted variable is used, the BASIC language sets aside eleven spaces in the data memory for subscripts 0 through 10. This is usually more than enough for the Lagrange interpolation program; in the example considered, the approximation $n = 4$ led to a result that is good enough for most practical applications. If large values of n are required, the DIM (dimension) statement is used. This consists of the statement DIM followed by the name of the variable, followed by the value of the highest subscript in parentheses. Thus, for example, to use the Lagrange interpolation program with $n = 20$, we could insert the line

```
5 DIM X(20)
```

A program for sorting numbers further illustrates of the use of subscripted variables. Suppose that we have a sequence of numbers arranged in random order. We want to write a program to arrange them in correct numerical order, increasing from left to right. Before writing the program, it may be helpful to consider how the sorting process will be organized. We need a nested loop. The inner loop compares successive pairs of data and arranges each pair in correct order, running from left to right through the sequence. The outer loop then repeats this operation. If there are n input numbers, the inner loop initially runs $n - 1$ times; the outer loop also runs $n - 1$ times. We trace the following input data through $n - 1 = 4$ cycles of the inner loop (one cycle of the outer loop):

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

The highest number is now at the right. For the next cycle of the outer loop, we need only $n - 2 = 3$ cycles of the inner loop. The result is

3 2 1 4 5

The third and fourth cycles of the outer loop lead to

2 1 3 4 5
1 2 3 4 5

which is the desired result. The required number of cycles of the inner loop decreases by 1 with each successive cycle of the outer loop.

```
  1 REM: SORTING NUMBERS
 10 N=10                                  Assigns value of n.
 20 FOR J=1 TO N                        }
 30 READ X(J)                           } Reads numbers to be sorted.
 40 NEXT J                              }
 50 FOR J=1 TO N-1                      }
 60 FOR I=1 TO N-J                      }
 70 IF X(I)<=X(I+1) THEN 110            }
 80 T=X(I)                              } Sorts numbers.
 90 X(I)=X(I+1)                         }
100 X(I+1)=T                            }
110 NEXT I                              }
120 NEXT J                              }
130 FOR J=1 TO N-1                      }
140 PRINT X(J);",";                     } Prints sorted sequence.
150 NEXT J                              }
160 PRINT X(N)                          }
170 DATA 7,3,5,9,4,6,2,10,8,1           Data line.
```

In the preceding program, line 1 is the title. Line 20 assigns the value of n, the number of terms to be sorted. Lines 20 through 40 constitute a FOR-NEXT loop that reads the values of the numbers to be sorted. Lines 50 through 120 constitute a nested FOR-NEXT loop, which sorts the numbers according to the scheme already discussed. The inner loop runs from line 60 through line 110. Line 70 compares two successive numbers x_i and x_{i+1} to see whether they are in the correct order (with the larger number at the right). If they are not, they are interchanged in lines 80 through 100. During the exchange operation, x_i is given a temporary label t so that its value is not lost when it is replaced by x_{i+1}. If the terms are in correct order, the exchange operation is skipped. The FOR-NEXT loop of lines 130 through 150 prints the first $n - 1$ terms of the sorted sequence, followed by commas. Line 160 prints the last term. Line 170 is the data line. This contains the sequence

7,3,5,9,4,6,2,10,8,1

By running the program, we obtain the sorted sequence

1,2,3,4,5,6,7,8,9,10

The same program can easily be applied to other sequences. Lines 10 and 170 are filled in by the user each time the program is run.*

If more than ten terms are to be sorted, a DIMension statement must be used. If there are 20 terms, we may insert the following line:

```
5 DIM X(20)
```

The same thing can be accomplished in a slightly different way by inserting the line

```
15 DIM X(N)
```

The second method is by far the more convenient way of using a DIMension statement, because the line can be left as a permanent part of the program and does not have to be adjusted each time the program is used. However, some computers, such as the TI-99/4 and some Apples, do not allow a variable to be used as the index in a DIMension statement. With any machine that has this limitation, the most convenient procedure is to insert a large number as the index in line 5, then leave it as a permanent part of the program.

Either the INPUT statement or the DATA statement can be used in a program for sorting numbers; the choice is a matter of individual preference. If it is necessary to correct an error or make a change in the sequence after the program is run, the DATA statement is a little more convenient. All the numbers are still in the program, so it is not necessary to reenter the entire sequence. We shall now give a slightly different version of the program using the INPUT statement. At the same time we shall modify the program so that it is not necessary to count the numbers and enter *n*. The following revision accomplishes this:

```
 5 DIM X(100)
10 INPUT Y$
20 If Y$="E" THEN 50
30 N=N+1
40 X(N)=VAL(Y$)
45 GOTO 10
```

* This method of sorting numbers is known as *bubble sort.* It is one of the simplest methods, but it is not one of the most efficient. If it is necessary to sort many sequences containing more than 20 to 30 terms, it may be worthwhile to consider one of the more sophisticated methods that can be found in the literature.

Also, line 170 may be deleted.

Line 5 is a DIMension statement. (Since we do not count the entries in using this program, we simply choose a large number and leave it as a permanent part of the program.) Line 10 is an INPUT statement. A string variable is used, so the input may be either a number or a letter. The numbers of the sequence are inserted one by one in response to the question marks that appear on the screen. After each numeric entry, line 45 returns the execution to line 10 in preparation for further input. Line 30 keeps a running count of the number of entries. After all the numbers have been entered, the operator enters E (for "end") to indicate that the input is complete. Line 20 then transfers the execution to line 50, where the sorting process begins. The only thing new is the VALue function in line 40. This function returns the value of a numerical string; for example, if Y$="7", then the expression VAL(Y$) returns the value 7. Using the same example as before, we enter the sequence

7 3 5 9 4 6 2 10 8 1 E

and we obtain the same result.

This technique can also be used with the DATA statement; the DATA statement handles strings in the same way as the INPUT statement.

Problems

1-1. Write programs to evaluate the following functions:

a. $y = 5 - 3x + 2x^2 + 3x^3 - x^4$
b. $y = 2x^5 - x^4 + 3x^3 + 2x^2 - x - 1$
c. $y = e^{3x} - x^2 + 5x^3 - \cos x$
d. $y = x^3 \ln x + x^2 - 3x + 2 \sin x$
e. $y = e^x \cos x + e^{-x} \sin x$

Check the programs by obtaining numerical results with $x = 2$.

Ans. a. 15 b. 77 c. 439.84494 d. 5.3637723 e. −2.9518723

1-2. Write programs for the following functions:

a. $y = 2x - 3,\ x \leq -1;\quad y = x^2 + 6x,\ x \geq -1$
b. $y = (x-2)^2,\ x \leq 2;\quad y = 0,\ x \geq 2$
c. $y = e^x \quad x \leq 2;\quad\quad y = 1,\ x > 2$

1-3. A bank has the following service charges for checks:

$0.10 per check for the first five checks (1-5)
.09 per check for the next five (6-10)
.08 per check for the next five (11-15)
.07 per check for each check over 15

Write a program to calculate the total service charge.

1-4. Write programs to evaluate the following finite sums. (The analytical expressions for the sums are given to make it easy to check the programs.)

a. $1^2+2^2+3^2+\ldots+n^2=\frac{n}{6}(n+1)(2n+1)$

b. $1^2+3^2+5^2+\ldots+(2n-1)^2=\frac{n}{3}(4n^2-1)$

c. $1^3+2^3+3^3+\ldots+n^3=\left[\frac{n}{2}(n+1)\right]^2$

d. $1^3+3^3+5^3+\ldots+(2n-1)^3=n^2(2n^2-1)$

1-5. Write programs to evaluate the following infinite series. (The analytical expressions for the sums are given to make it easy to check the programs.)

a. $\left(\frac{1}{1\cdot2\cdot3}\right)^2+\left(\frac{1}{2\cdot3\cdot4}\right)^2+\left(\frac{1}{3\cdot4\cdot5}\right)^2+\ldots=\frac{\pi^2}{4}-\frac{39}{16}$

b. $\frac{1}{1\cdot2\cdot3}+\frac{1}{5\cdot6\cdot7}+\frac{1}{9\cdot10\cdot11}+\ldots=\frac{1}{4}\ln 2$

c. $1-\frac{1}{2\cdot2}+\frac{1}{3\cdot2^2}-\frac{1}{4\cdot2^3}+\ldots=2\ln\frac{3}{2}$

d. $1-\frac{1}{3\cdot3}+\frac{1}{5\cdot3^2}-\frac{1}{7\cdot3^3}+\ldots=\frac{\pi}{2\sqrt{3}}$

e. $1-\frac{1}{3}+\frac{1\cdot2}{3\cdot5}-\frac{1\cdot2\cdot3}{3\cdot5\cdot7}+\ldots=\frac{2}{\sqrt{3}}\ln\frac{\sqrt{3}+1}{\sqrt{2}}$

1-6. Write a program to evaluate the binomial coefficient

$$\binom{p}{q}=\frac{\mathrm{p!}}{q!(p-q)!}$$

Numerical results to check the program can be found in almost any mathematics handbook.

1-7. Modify the program for Eq. 1-13 in Sec. 1-5 so that the original sequence can be extended without repeating the prior calculations.

1-8. Write programs for the following recurrence formulas, and use them to find the first few values of y_j.

a. $y_{j+1}-5y_j+6y_{j-1}=0 \qquad y_1=1,\ y_2=2$

Ans. $y_j=1, 2, 4, 8, 16, 32, \ldots$

b. $y_{j+1}-2y_j+2y_{j-1}=0 \qquad y_1=3,\ y_2=5$

Ans. $y_j=3, 5, 4, -2, -12, -20, \ldots$

1-9. a. Show that the integral

$$I_n = \int_0^{\pi/2} x^n \sin dx \qquad n = 0,1,2, \ldots$$

satisfies the recurrence formula

$$I_{n+1} = (n+1)\left[\left(\frac{\pi}{2}\right)^n - nI_{n-1}\right] \qquad n \geq 1$$

b. Also show by elementary integration that $I_0 = I_1 = 1$. Write a program to evaluate $I_2, I_3, \ldots I_n$

Ans. 1.1415927, 1.4022033, 1.8040265, 2.3962749, . . .

1-10. The Hermite polynomials are considered in advanced calculus. The first few are

$$H_0(x) = 1$$
$$H_1(x) = 2x$$
$$H_2(x) = 4x^2 - 2$$
$$H_3(x) = 3x^3 - 12x$$
$$H_4(x) = 16x^4 - 48x^2 + 12$$

By using the recurrence formula

$$H_{n+1}(x) - 2xH_n(x) + 2nH_{n-1}(x) = 0$$

or otherwise, write a program to evaluate the Hermite polynomials. Numerical results to check the program for values of n from 0 through 4 can easily be obtained from the basic formulas given.

1-11. Write a program for the problem of Sec. 1-4 using subscripted variables instead of a subroutine.

1-12. Using the INPUT statement, write a program that sorts numbers and prints the intermediate sorted sequences after each input entry.

2

Roots of Equations

2-1. The Method of Iteration

It is often necessary to find the roots of various types of equations. In this chapter we consider several methods of solving equations. We start with the method of iteration. To solve the equation

$$y = f(x) = 0 \tag{2-1}$$

We write it in the form

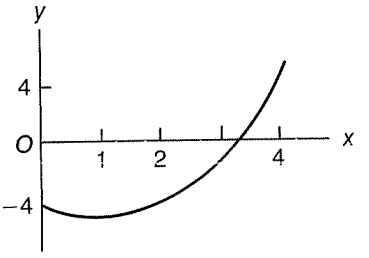
$$x = \phi(x) \tag{2-2}$$

It is sometimes possible to solve the equation very easily by first obtaining a preliminary estimate of x from a rough plot of the function, then substituting this into the right side of Eq. 2-2. If the procedure is successful, the resulting value on the left is closer to the true value than the original estimate. This procedure is repeated as many times as necessary until the desired accuracy is obtained.

Consider the equation

$$y = x^2 - 2x - 4 = 0 \tag{2-3}$$

FIG. 2-1



A plot of y against x appears in Fig. 2-1. There are various ways in which Eq. 2-3 can be converted to the form of Eq. 2-2. We write $x^2 = 4 + 2x$. Then

$$x = (4 + 2x)^{1/2} \tag{2-4}$$

The sketch of Fig. 2-1 suggests $x = 3.2$ for a rough estimate of the root. There are several possible ways of programming an iterative evaluation, and we shall consider a few. The simplest possibility is

```
10  X=3.2
20  X=SQR(4+2*X)
30  PRINT X
```

To operate the program we enter RUN. The number 3.225 appears on the screen. At this point we must decide how to proceed to the next higher approximation. Each subsequent cycle must begin with the last prior value of x. Therefore we want to return to line 20—not to line 10. We can do so by entering RUN 20. With a few computers, this leads to the next higher approximation. However, most computers automatically set all variables equal to zero whenever the RUN statement is entered. Hence we may expect to obtain the result 2, which is incorrect. Another approach is to enter GOTO 20. GOTO is a statement normally used in a program, whereas RUN is a command used in the prompt mode. Most microcomputers, such as the Apple IIe, the TRS-80, and the Commodore 64, employ a broad form of BASIC in which statements and commands may be used interchangeably. On any machine of this type, we can solve the equation by repeatedly entering GOTO 20. Then we obtain the sequence

3.2 3.225 3.2326 3.2350 3.23574 3.23597 3.23604 3.236058
3.236065 3.2360670 3.2360677 3.2360679

The convergence is not very rapid, but we eventually obtain a high degree of accuracy. The last result agrees to the full number of digits shown with the exact solution $\sqrt{5} + 1$.

Some computers and microcomputers such as the TI-99/4 maintain a sharp separation between statements and commands. On a machine of

this type, GOTO cannot be used in the prompt mode, and the foregoing solution does not work.

To avoid having to type GOTO 20 repeatedly, we incorporate the GOTO statement into the program as follows:

```
10  INPUT X
20  X=SQR(4+2*X)
30  PRINT X
40  GOTO 10
```

The INPUT statement serves to interrupt the execution; otherwise the computer would rush ahead grinding out further iterations indefinitely without waiting for a signal to begin each new cycle. The first time a question mark appears on the screen, enter the initial estimate 3.2. For further iterations, press the ENTER key without entering anything. Execution then proceeds to the next cycle, using the current value of x. This procedure works with the TRS-80 (all models) and the Commodore 64, but it does not work with most older Commodores or most other computers. Most computers require a definite entry in response to a request for numeric input. To use the foregoing program, the operator has to type in the last value of x on the screen each time so that it can be used as the starting value for the next cycle.

The best procedure is to choose some character that does not appear in the calculations as a dummy input variable, preferably a string. With this technique, the program becomes

```
 1  REM: ROOTS OF EQUATIONS BY ITERATION
10  X=3.2              Assigns initial estimate of x.
20  PRINT X            Prints x.
30  INPUT Q$           Interrupts execution.
40  X=SQR(4+2*X)       Calculates new value of x.
50  GOTO 20            Starts next cycle.
```

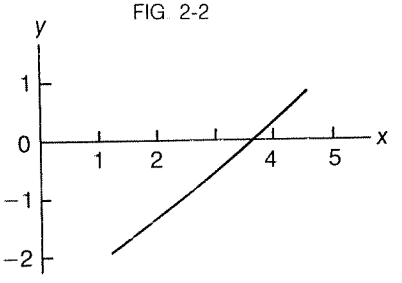
We have also added a title and changed the order of the steps so the initial estimate is printed as the first item of output. Each time a question mark appears on the screen, the operator proceeds to the next iteration by pressing the ENTER key without entering anything. Although most computers do not allow this when numeric input is requested, a null string (a string consisting of nothing) is a legitimate entry in any version of BASIC. After satisfactory convergence has been obtained, the execution is terminated by pressing the BREAK key.

One minor change may make the program a little more convenient to use with the Commodore 64. When execution is terminated by a BREAK operation (RUN/STOP and RESTORE), this computer automatically clears the screen. However, it is possible to terminate execution without losing the data by entering the line

```
35 IF Q$="END" THEN END
```

The execution is now terminated by entering END.

FIG. 2-2



As a second example, consider the equation

$$y = x - \frac{1}{2}\ln x - 3 = 0 \tag{2-5}$$

A plot of y against x appears in Fig. 2-2. We write

$$x = \frac{1}{2}\ln x + 3 \tag{2-6}$$

From the sketch of Fig. 2-2, we obtain $x = 3.6$ for a rough estimate of the root. Lines 10 and 40 of the program become

```
10 X=3.6
40 X=LOG(X)/2+3
```

The operation of the program is the same as in the first example. We enter RUN to start the execution and then press ENTER for each iteration. This leads to the sequence

3.6 3.640 3.6461 3.6468 3.64693 3.646943 3.6469446
3.64694486 3.64694490

As a third example, consider the equation

$$y = \tan x - x = 0 \tag{2-7}$$

A plot of y against x appears in Fig. 2-3. We write

$$x = \arctan x \tag{2-8}$$

In the present example it is essential to look carefully at Fig. 2-3 before writing the program. The desired root is approximately 4.4, which is in

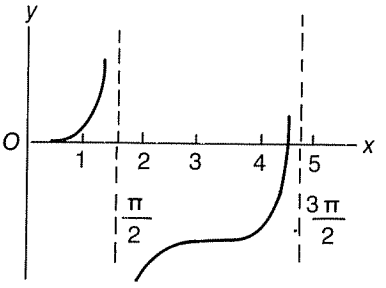


FIG. 2-3

the third quadrant. Since the inverse trigonometric functions given by the computer are principal values, it is necessary to increase the arc tangent by π. Lines 10 and 40 of the program become

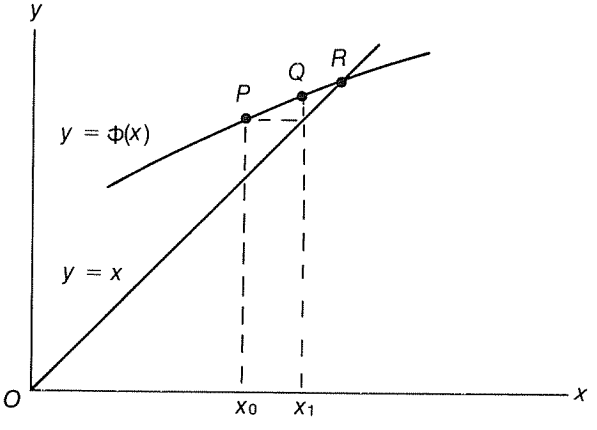
```
10  X=4.4
40  X=ATN(X)+4*ATN(1)
```

We obtain the sequence

4.4 4.489 4.4932 4.49340 4.493409 4.49340944 4.493409457 4.493409458

The iterative process is depicted graphically in Fig. 2-4. We start by assuming a value x_0. The first iteration gives the result $y_0 = \phi(x_0)$ at point P. This becomes the starting value x_1 for the next cycle. The new abscissa x_1 is located by drawing the horizontal and vertical lines shown. A second iteration leads to the result $y_1 = \phi(x_1)$ at point Q. This becomes x_2. If the procedure is successful, successive iterations converge toward the exact result at point R.

FIG. 2-4



It is obvious that, given an equation $f(x) = 0$, the choice of the equation $x = \phi(x)$ is essentially arbitrary. For example, given Eq. 2-7, we might have chosen to write $x = \tan x$ instead of Eq. 2-8. This choice has a very strong effect on the convergence of the iterative process. In fact, an iterative solution of the equation $x = \tan x$ diverges. We need a criterion for convergence, and a very simple sufficient condition is available. The iterative process converges provided that the slope

$$\left| \frac{d\phi}{dx} \right| < 1$$

satisfies the inquality $|d\phi/dx| < 1$ throughout the interval of iteration. To show this, let k be the maximum absolute value of the slope on the interval. Then it is clear that

$$\left| \frac{y - y_0}{x - x_0} \right| = \left| \frac{x - x_1}{x - x_0} \right| \leq k$$

$$\left| \frac{y - y_1}{x - x_1} \right| = \left| \frac{x - x_2}{x - x_1} \right| \leq k$$

$$\left| \frac{y - y_{n-1}}{x - x_{n-1}} \right| = \left| \frac{x - x_n}{x - x_{n-1}} \right| \leq k$$

where x and y are the true values at point R. By multiplying the middle and right members of these inequalities, we find that

$$|x - x_n| \leq k^n |x - x_0|$$

It follows that, if $k < 1$, then the error $x_n - x$ approaches 0 as the number of iterations n becomes large, and the process converges. Convergence is most rapid if the curve $y = \phi(x)$ is approximately horizontal, that is, if $d\phi/dx$ is close to zero. It is not necessary in practice to check the value of $d\phi/dx$ throughout the interval; the value at the starting point P usually gives a good indication of convergence, provided that the initial estimate is not too far from the correct value.

To illustrate the application of these remarks, we return to Eq. 2-5. It can be seen that the iteration based on Eq. 2-6 succeeded because

$$\frac{d\phi}{dx} = \frac{1}{2x} \approx \frac{1}{2 \cdot 3.6} \approx .14$$

which satisfies the condition for convergence. An extension of the plot of Fig. 2-2 shows that there is a second root $x \approx .0025$. If Eq. 2-6 is used to evaluate this root by iteration, we have $d\phi/dx \approx 200$, so the process may be expected to diverge. This turns out to be true; we obtain the sequence

.0025 .0043 .272 2.35

We now try a different iterative equation, namely

$$x = e^{2x-6}$$

Then

$$\frac{d\phi}{dx} = 2e^{2x-6} = 2x \approx .005$$

so the process may be expected to converge. Lines 10 and 40 of the program become

```
10  X=.0025
40  X=EXP(2*X-6)
```

We obtain the sequence

.0025 .0024912 .002491133 .0024911328

The convergence of the iteration process is affected by the choice of the starting value, but it is difficult to predict this effect in advance. We now return to Eq. 2-8 and start the evaluation with a very poor guess, say $x = 4.0$. The next value will be 4.467, which is somewhat better than the starting value 4.4 actually used. Hence the worst that can happen is that the poor initial guess necessitates one extra cycle in the iterative process. In some problems, the effect of a poor initial guess is greater, but in general the simple iteration process is less sensitive to the initial guess than many other methods.

In the foregoing programs we have obtained results by displaying the iterative sequence on the screen and deciding by inspection at each step whether the process has converged satisfactorily. This method is generally satisfactory for a microcomputer (or for any computer that is used interactively), but there is an alternative. It is possible to include a segment in the program to check the error after each iteration and terminate execution when it falls within some predetermined limit. Unfortunately we do not know the true error unless we happen to know the exact solution, in which case an iterative solution is unnecessary. Our next problem is to find some way around this difficulty. We remark that the difference of two successive approximations is not an indication of the error; it may be either greater or smaller than the true error. However, we shall see that a very good estimate of the error can be derived from three successive approximations. At the same time we shall consider another problem: the slow convergence that sometimes occurs with iterative solutions, as, for example, in the solution of Eq. 2-4. The two problems are closely related; if we know the approximate error of an evaluation, we can adjust the result to obtain accelerated convergence.

We begin by referring to Fig. 2-4. Starting with the initial estimate

x_0, we obtained the iterative results $y_0 = x_1$ at point P and $y_1 = x_2$ at point Q. We then approached the exact solution at point R by repeated iterations. We now adopt a different procedure. Having located points P and Q, we pass a straight line through them and extrapolate it to an intersection with the line $y = x$. The result will be very close to the exact point R. The equation of the line through P and Q is

$$y = y_1 + r(x - x_1)$$

The parameter r, the slope, is given by the equation

$$r = \frac{y_1 - y_0}{x_1 - x_0} = \frac{x_2 - x_1}{x_1 - x_0} \tag{2-9a}$$

The intersection of the line PQ with the line $y = x$ is found by substituting x for y in the equation for PQ. At the same time we use the fact that $y_1 = x_2$. Thus

$$x = x_2 + r(x - x_1)$$

By solving for x, we find that

$$x = \frac{x_2 - rx_1}{1 - r} = x_2 - \frac{x_2 - x_1}{1 - \frac{1}{r}}$$

We now break the last result into the two equations

$$e = \frac{x_2 - x_1}{1 - \frac{1}{r}} \tag{2-9b}$$

$$x = x_2 - e \tag{2-9c}$$

The parameter e is the estimated error of the last iterative result x_2. Eqs. 2-9 are sometimes combined into the single formula

$$x = x_2 - \frac{(x_2 - x_1)^2}{x_2 - 2x_1 + x_0}$$

which is known as Aitken's extrapolation formula. However, the separate Eqs. 2-9a,b,c are more useful for our purposes.

Eqs. 2-9 may be used in either of two ways. We may obtain successive results by ordinary iteration, using Eqs. 2-9a,b to estimate the error at each step. The other possibility is to use the present results as an extrapolation formula to accelerate the convergence of the iterative process. The

procedure that we shall follow is a combination of the two. We start by iterating twice; then we estimate the error. If the error is within a specified limit, the evaluation is completed. Otherwise we extrapolate, iterate twice from the extrapolated result, and check the error again. This cycle is repeated as many times as necessary until the error falls within the specified limit.

The program should include a segment to test successive results for convergence and terminate execution if the iterative process is diverging. We have seen that the criterion for convergence is that

$$\left|\frac{d\phi}{dx}\right| \approx |r| < 1$$

A program follows for Eq. 2-4. Line 1 is the title. Line 10 contains the initial value of x, and line 20 assigns this to x_0. Lines 30 through 70 constitute a FOR-NEXT loop that performs two iterations. At the same time the appropriate values are assigned to x_1 and x_2. Line 80 terminates the iteration if two successive results are identical, because this means that the iteration has already converged to the full accuracy of the computer. Also, the termination forestalls a division by zero in a subsequent step. Line 90 calculates r. Lines 100 through 120 test whether the iteration is converging and, if it is not, terminate the execution, printing an appropriate message. Line 130 estimates the error, and line 140 calculates the extrapolated value of x. Line 150 compares the estimated error with the allowable limit. If the error is excessive, the cycle is repeated. Line 160 prints the final result. The error of the final result is usually substantially less than the limit allowed in line 150, because the error of x_2 is checked against the allowable limit, whereas the extrapolated value of x is printed.

```
  1  REM: AUTOMATIC ITERATION WITH EXTRAPOLATION
 10  X=3.2                          Assigns initial estimate of x.
 20  X0=X                           Assigns value of x to x0.
 30  FOR J=1 TO 2                   ┐
 40  X=SQR(4+2*X)                   │
 50  X1=X2                          ├ Iterates twice.
 60  X2=X                           │
 70  NEXT J                         ┘
 80  IF X1=X2 THEN 160              Terminates iteration if x1=x2.
 90  R=(X2-X1)/(X1-X0)              Calculates r.
100  IF ABS(R)<1 THEN 130           ┐
110  PRINT "THE ITERATION           ├ Terminates execution
     DIVERGES."                     ┘ if process diverges.
120  END
130  E=(X2-X1)/(1-1/R)              Estimates error.
140  X=X2-E                         Calculates extrapolated x.
150  IF ABS(E)>10^-6 THEN 20        Checks error and repeats
                                    cycle if excessive.
160  PRINT X                        Prints result.
```

When the RUN command is entered, the computer prints the result 3.236067978. This is identical to the exact value $\sqrt{5}+1$ to ten significant figures, and it is far more accurate than line 150 requires it to be. The results for intermediate cycles can be obtained by inserting the line

```
25 PRINT X
```

The resulting sequence appears in the first line of the following figures:

```
3.2  3.2360766  3.236067978
3.2  3.2326     3.23574
```

In the second line we show results obtained previously at corresponding stages of the simple iteration process, remembering that each cycle of the accelerated process represents two iterations. It can be seen that the extrapolation yields a great improvement in convergence.

The value of the allowable error in line 150 may be chosen according to the desired accuracy of the solution and the accuracy of the computer. The value 10^{-6} is generally satisfactory for most computers. It may be necessary to allow a somewhat larger error for the TRS-80 because of this computer's lower accuracy.

The same program can be applied to other equations; new lines 10 and 40 must be filled in by the user each time. The amendments are exactly the same as those used in the simple iteration process. For Eq. 2-6 we obtain the result 3.646944902 and for Eq. 2-8 we get 4.493409458. Both results are correct to ten significant figures.

2-2. The Newton-Raphson Method

The Newton-Raphson method is also an iterative method of solving an equation of the type $y = f(x) = 0$. However, instead of choosing the equation $x = \phi(x)$ arbitrarily, we adopt a more systematic viewpoint. Consider the Taylor series

$$y = y_0 + (x - x_0)y_0' + \frac{1}{2}(x - x_0)^2 y_0'' + \cdots \tag{2-10}$$

By taking only two terms on the right and solving for x, we obtain the equation

$$x = x_0 + \frac{y - y_0}{y_0'}$$

For the desired result $y = 0$, this becomes

$$x = x_0 - \frac{y_0}{y_0'} \tag{2-11}$$

We now apply the method to Eq. 2-3. Let

$$y = x^2 - 2x - 4 = 0$$

Then

$$y' = 2x - 2$$

It follows that

$$x = x_0 - \frac{x_0^2 - 2x_0 - 4}{2x_0 - 2} = \frac{x_0^2 + 4}{2(x_0 - 1)} \tag{2-12}$$

Equation 2-12 is a little more complicated than the analogous Eq. 2-4, which we used in solving the problem by the simple iteration method. However, exactly the same programs can be used, except that the equation line must be changed. It is not necessary to distinguish between x_0 and x. We again start with the estimate $x = 3.2$. The program is

```
 1  REM: ROOTS OF EQUATIONS BY THE NEWTON-
    RAPHSON METHOD
10  X=3.2                    Assigns initial estimate of x.
20  PRINT X                  Prints x.
30  INPUT Q$                 Interrupts execution.
40  X=(X*X+4)/2/(X-1)        Calculates new value of x.
50  GOTO 20                  Starts next cycle.
```

After the RUN command is entered, the starting estimate 3.2 is displayed. Successive approximations are obtained by pressing ENTER each time a question mark appears on the screen. We obtain the sequence

3.2 3.2364 3.236068 3.23606798

In this example the Newton-Raphson method is much more efficient than the simple iteration method.

We again consider Eq. 2-5, which is

$$y = x - \frac{1}{2}\ln x - 3 = 0$$

It is clear that

$$y' = 1 - \frac{1}{2x}$$

and it follows that

$$x = x_0 - \frac{x_0 - \frac{1}{2}\ln x_0 - 3}{1 - \frac{1}{2x_0}} = \frac{\ln x_0 + 5}{2 - \frac{1}{x_0}} \tag{2-13}$$

As in Sec. 2-1, we start with the estimate $x = 3.6$. Lines 10 and 40 of the program become

```
10  X=3.6
40  X=(LOG(X)+5)/(2-1/X)
```

This leads to the sequence

3.6 3.6470 3.64694490

In this example the Newton-Raphson method is again much more efficient than the simple iteration method.

We again consider Eq. 2-7, which is

$$y = \tan x - x = 0$$

Then

$$y' = \tan^2 x$$

and it follows that

$$x = x_0 - \frac{\tan x_0 - x_0}{\tan^2 x_0} = \frac{x_0}{\sin^2 x_0} - \frac{1}{\tan x_0} \tag{2-14}$$

As in Sec. 2-1, we start with the estimate $x = 4.4$. Lines 10 and 40 of the program become

```
10  X=4.4
40  X=X/SIN(X)/SIN(X)-1/TAN(X)
```

We obtain the sequence

4.4 4.536 4.5019 4.49375 4.4934100 4.493409458

In this example the Newton-Raphson method is not much more efficient than the simple iteration method.

The Newton-Raphson method is shown graphically in Fig. 2-5. The desired root x is estimated, and the tangent is drawn to the curve $y = f(x)$

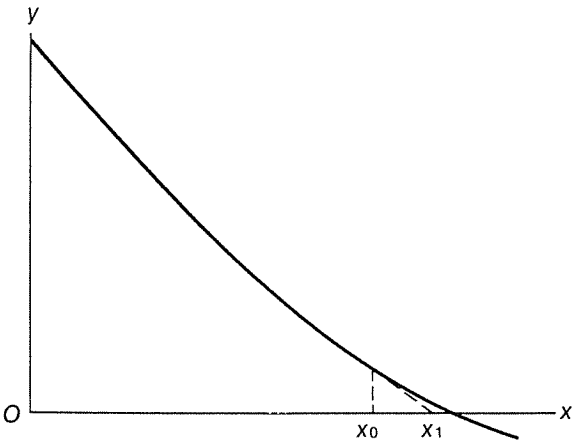


FIGURE 2-5

at that point. The intersection of the tangent with the x axis represents the solution to Eq. 2-11. The process is repeated as often as necessary until the desired accuracy is obtained.

Many difficulties can occur in using any numerical method. Sometimes a process is inherently unstable for a particular equation; in other cases the difficulty is caused by a poor initial estimate of the root. We rework the last example, this time starting with the poor initial guess $x_0 = 4.2$. Then we obtain the sequence

4.2 4.96 5.56 13.6

It is clear that the process diverges. The stability of the Newton-Raphson method is more sensitive to the initial error than that of the simple iteration method; the latter method converged for this problem even with the very poor initial guess $x = 4.0$.

The Newton-Raphson method can also be used in conjunction with the extrapolation program of Sec. 2-1.

2-3. The Secant Method

Geometrically the Newton-Raphson method consists of making a first estimate of x and then obtaining an improved value by drawing the tangent and extrapolating it to the x axis. A commonly used alternative method is to choose two points that bracket the exact root, then draw the chord connecting them and take the intersection with the x axis as the result, as shown in Fig. 2-6. This is known as the secant method. It is essentially a form of inverse linear interpolation. The equation is

$$x = \frac{x_1 y_2 - x_2 y_1}{y_2 - y_1} \tag{2-15}$$

The process is repeated as many times as desired. The last and next-to-last previous values of x and y are used as the starting points for each

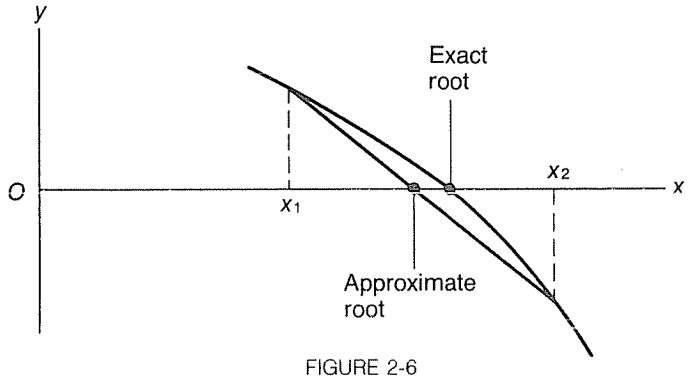


FIGURE 2-6

new cycle. The program for this method is more complicated than those used previously, and the method is somewhat less efficient than the Newton-Raphson method. The major advantage of the method is that the calculations are entirely automatic; no preliminary calculus and algebra are required.

The program follows. It is longer than the programs of Secs. 2-1 and 2-2 because each iteration uses four prior values: two of x and two of y. The earlier programs used only the immediately preceding value of x. The easiest way to explain the program is to deviate slightly from the order in which the lines appear; the reasons for some of the earlier lines will become apparent later. Line 1 is the title. Lines 10 through 50 read the values of the initial estimates x_1 and x_2 from the data line, initialize y_1, print x_1, assign the value of x_1 to x, and calculate y. We use Eq. 2-3, which is

$$y = x^2 - 2x - 4 = 0$$

Line 70 calculates the next iteration for x. (Lines 20 and 60 cause this calculation to be bypassed on the first cycle, since x_2 cannot be calculated from x_1; it is read from the data line.) It is not necessary to distinguish between y and y_2. Lines 80 through 110 reassign the values of the xs and y in preparation for the next cycle. (Lines 80 and 90 are skipped on the first cycle; x_1 and x_2 retain their original values.) Line 120 prints the latest value of x. Line 130 is a dummy INPUT statement. When this is reached, the computer stops the execution of the program and waits for the operator to press ENTER to start the next iteration. Line 140 then sends the execution back to line 50 to start the next cycle. After satisfactory convergence has been obtained, execution is terminated by pressing the BREAK key. Line 150 is the data line, which contains the values of x_1 and x_2. From Fig. 2-1, we obtain the estimates $x_1 = 3.2$, $x_2 = 3.3$. Lines 50 and 150 are filled in by the operator each time the program is applied to a new equation.

```
  1  REM: ROOTS OF EQUATIONS BY THE SECANT METHOD
 10  READ X1,X2                      Reads initial estimates.
 20  Y1=0                            Initializes y1.
 30  PRINT X1                        Prints x1.
 40  X=X1                            Assigns value of x1 to x.
 50  Y=X*(X-2)-4                     Calculates y.
 60  IF Y1=0 THEN 100                Bypasses iteration on first cycle.
 70  X=(X1*Y-X2*Y1)/(Y-Y1)           Calculates new value of x.
 80  X1=X2
 90  X2=X                            } Reassignments.
100  Y1=Y
110  X=X2
120  PRINT X                         Prints latest value of x.
130  INPUT Q$                        Interrupts execution.
140  GOTO 50                         Returns for next cycle.
150  DATA 3.2, 3.3                   Data line for x1 and x2.
```

We obtain the sequence

3.2 3.3 3.2356 3.236061 3.23606798

The starting values 3.2 and 3.3 appear on the screen immediately after the RUN command is entered. Subsequent results are obtained by pressing ENTER each time a question mark appears on the screen.

As a second example, we again use Eq. 2-5, which is

$$y = x - \frac{1}{2}\ln x - 3 = 0$$

From Fig. 2-2 we obtain the starting values $x_1 = 3.6$, $x_2 = 3.7$. Lines 50 and 150 of the program are edited to

```
 50  Y=X-LOG(X)/2-3
150  DATA 3.6, 3.7
```

We obtain the sequence

3.6 3.7 3.64689 3.6469448 3.64694490

As a third example we again use Eq. 2-7, which is

$$y = \tan x - x = 0$$

We start with the values $x_1 = 4.4$, $x_2 = 4.6$. Then lines 50 and 150 of the program become

```
 50  Y=TAN(X)-X
150  DATA 4.4, 4.6
```

The results are

4.4 4.6 4.447 4.470 4.4985 4.4928 4.49340 4.4934095 4.493409458

The initial iteration in the secant method as depicted in Fig. 2-6 is an interpolation. Subsequent results may be either interpolations or extrapolations. If successive results oscillate about the true value, each iteration is an interpolation between a high and a low estimate. However, an inspection of the results of the foregoing examples shows that it is also possible for results to approach the exact value from one side. In this case the method is an extrapolation process rather than an interpolation process.

The foregoing examples illustrate the fact that the secant method converges more slowly than the Newton-Raphson method. However, it is easier to use because it requires no preliminary calculus and algebra. The secant method shares one disadvantage with the Newton-Raphson method; a reasonably good starting estimate is necessary to ensure convergence.

2-4. Roots of Equations by Lagrange Interpolation

In the secant method, each estimate of the root is made by inverse linear interpolation between the two immediately preceding results. More rapid convergence is obtained by using all of the prior data in each iteration. We use inverse Lagrange interpolation. The desired equations are obtained by interchanging x and y in Eqs. 1-16, then setting $y = 0$. This leads to

$$x_3 = \frac{x_1}{1 - \frac{y_1}{y_2}} + \frac{x_2}{1 - \frac{y_2}{y_1}} \tag{2-16a}$$

$$x_4 = \frac{x_1}{\left(1 - \frac{y_1}{y_2}\right)\left(1 - \frac{y_1}{y_3}\right)} + \frac{x_2}{\left(1 - \frac{y_2}{y_1}\right)\left(1 - \frac{y_2}{y_3}\right)} + \frac{x_3}{\left(1 - \frac{y_3}{y_1}\right)\left(1 - \frac{y_3}{y_2}\right)} \tag{2-16b}$$

$$x_5 = \frac{x_1}{\left(1 - \frac{y_1}{y_2}\right)\left(1 - \frac{y_1}{y_3}\right)\left(1 - \frac{y_1}{y_4}\right)} + \frac{x_2}{\left(1 - \frac{y_2}{y_1}\right)\left(1 - \frac{y_2}{y_3}\right)\left(1 - \frac{y_2}{y_4}\right)} + \frac{x_3}{\left(1 - \frac{y_3}{y_1}\right)\left(1 - \frac{y_3}{y_2}\right)\left(1 - \frac{y_3}{y_4}\right)} + \frac{x_4}{\left(1 - \frac{y_4}{y_1}\right)\left(1 - \frac{y_4}{y_2}\right)\left(1 - \frac{y_4}{y_3}\right)} \tag{2-16c}$$

Clearly these results can be extended indefinitely. The first approximation is identical to Eq. 2-15 for the secant method, but subsequent approximations yield increasingly higher orders of accuracy. It can be seen that,

for the approximation x_{k+1}, there are k terms on the right side of the equation. We denote the general term of each approximation by u_j, where j runs from 1 through k. Only the last term u_k of each approximation is entirely new. Each of the terms u_1 through u_{k-1} is formed by dividing the corresponding term of the next lower approximation by the factor $(1 - y_j/y_k)$.

The program follows. Since all the earlier values of u and y are needed in each cycle, we represent these by subscripted variables. The variable x is not subscripted. A DIMension statement is not needed because the process usually converges in a few iterations. The program is set up to solve Eq. 2-3, which is

$$y = x^2 - 2x - 4 = 0$$

The program is organized in the same way as the program for the secant method in Sec. 2-3. However, it is longer because Eqs. 2-16 are more complicated than Eq. 2-15. The evaluation of x now occupies the segment from line 90 through line 160; in the program for the secant method, this was accomplished in line 70 alone. On the other hand, it is no longer necessary to reassign the values of all of the xs and ys in preparation for the next cycle; we simply add 1 to the index k in line 170. The equation line 60 is filled in by the operator each time the program is run. Ordinary variables are used, exactly as in the program for the secant method. The variable x is not subscripted, and y is automatically converted to subscripted form in line 70. To fill in the data line 210, we refer to Fig. 2-1 for the estimates $x_1 = 3.2$, $x_2 = 3.3$.

```
  1 REM: ROOTS OF EQUATIONS BY LAGRANGE INTERPOLATION
 10 READ X1,X2                    Reads initial estimates.
 20 K=1                           Initializes k.
 30 PRINT X1                      Prints x1.
 40 X=X1                          Assigns value of x1 to x.
 50 U(K)=X                        Initializes uk.
 60 Y=X*(X-2)-4                   Calculates y.
 70 Y(K)=Y                        Converts y to subscripted form.
 80 IF K=1 THEN 160               Bypasses iteration on first cycle.
 90 S=0                           }
100 FOR J=1 TO K-1                }
110 U(J)=U(J)/(1-Y(J)/Y(K))       }
120 S=S+U(J)                      } Calculates x2.
130 U(K)=U(K)/(1-Y(K)/Y(J))       }
140 NEXT J                        }
150 X2=S+U(K)                     }
160 X=X2                         Assigns value of x2 to x.
170 K=K+1                        Adjusts value of k.
180 PRINT X                      Prints latest result.
```

```
190 INPUT Q$          Interrupts execution.
200 GOTO 50           Returns for next cycle.
210 DATA 3.2, 3.3     Data line for x1 and x2.
```

To operate the program, lines 60 and 210 are filled in by the user. The starting values 3.2 and 3.3 appear on the screen immediately after the RUN command is entered. Subsequent results are obtained by pressing ENTER each time a question mark appears on the screen. After satisfactory convergence has been obtained, the execution of the program is terminated by pressing the BREAK key. We obtain the sequence

3.2 3.3 3.2356 3.2360681 3.23606798

This program runs as it stands on almost any microcomputer in common use, with one reservation. A few microcomputers, such as the TI-99/4, do not allow one character to represent both an ordinary variable and a subscripted variable. On a machine with this limitation, line 70 generates an error message. This trouble can be corrected either by substituting a new symbol for the unsubscripted Y in lines 60 and 70 or by deleting line 70 and writing Y(K) for Y in line 60.

As a second example we again use Eq. 2-5, which is

$$y = x - \tfrac{1}{2} \ln x - 3 = 0$$

As in Sec. 2-3, we use the starting values $x_1 = 3.6$, $x_2 = 3.7$. Lines 60 and 210 of the program are edited to

```
 60 Y=X-LOG(X)/2-3
210 DATA 3.6, 3.7
```

We obtain the sequence

3.6 3.7 3.64689 3.64694490

As a third example we again use Eq. 2-7, which is

$$y = \tan x - x = 0$$

As in Sec. 2-3, we start with the values $x_1 = 4.4$, $x_2 = 4.6$. Then lines 60 and 210 of the program become

```
 60 Y=TAN(X)-X
210 DATA 4.4, 4.6
```

The results are

4.4 4.6 4.447 4.5041 4.49398 4.493411 4.49340946

The method of finding roots by Lagrange interpolation gives better convergence than the secant method; in the examples considered here, the convergence is roughly equivalent to that of the Newton-Raphson method. Like the secant method, the present method is easy to use; it requires no preliminary calculus and algebra.

2-5. Quadratic Equations

Polynomial equations occur in many applications in science and engineering. If the roots are real, one of the methods of the preceding sections may be used. However, it is more convenient to have analytical solutions that give all the roots directly with no preliminary estimates required, especially if there are complex roots. Analytical solutions are available for quadratic, cubic, and quartic equations. These will be considered in this section and in the two sections that follow.

The general quadratic equation is

$$ax^2 + bx + c = 0 \tag{2-17}$$

We assume that the coefficients a, b, and c are real. Also, we exclude the trivial case $a = 0$, which has the single root $x = -c/b$. We solve the equation by dividing through by a and completing the square. This leads to

$$x^2 + \frac{b}{a}x + \frac{b^2}{4a^2} = \frac{b^2}{4a^2} - \frac{c}{a}$$

The solution is

$$x = -\frac{b}{2a} \pm \left[\left(\frac{b}{2a}\right)^2 - \frac{c}{a}\right]^{1/2} \tag{2-18}$$

For the numerical evaluation, it is convenient to make the substitutions

$$e = -\frac{b}{2a} \tag{2-19a}$$

$$G = \left(\frac{b}{2a}\right)^2 - \frac{c}{a} = e^2 - \frac{c}{a} \tag{2-19b}$$

Then the two roots are

$$x_1 = e - \sqrt{G} \tag{2-20a}$$

$$x_2 = e + \sqrt{G} \tag{2-20b}$$

The character of the roots depends on the value of G, which is known as the discriminant. If $G \geq 0$, the roots are real and their values are given directly by Eqs. 2-20. If $G < 0$, the roots are complex. Since the computer cannot handle imaginary numbers directly, the solution for this case must be rewritten as

$$x_1 = e + i\sqrt{-G} \tag{2-21a}$$

$$x_2 = e - i\sqrt{-G} \tag{2-21b}$$

The case $G = 0$ clearly represents a double real root $x_1 = x_2 = e$. However, a difficulty can occur in the numerical evaluation of this case. Sometimes the calculated value of G is some very small positive or negative number, because of machine error. If it is negative, the execution of the program will take the wrong branch and arrive at a complex result. Even if the calculated result is a small positive number, there may be trouble. The process of extracting a square root magnifies a small error. Suppose that the value of G should be 0, but the computer calculates the value 10^{-6}. This error may seem trivial, but when the square root is taken, it becomes .001, which may be excessive for some applications. Difficulties of this type are more serious in the analyses of cubic and quartic equations that follow, because the calculations are more intricate. It is desirable to consider the problem now.

We assume that any very small calculated result should probably be an exact zero. A correction is necessary if the calculated parameter governs the choice of branch in an IF-THEN statement or if a root will be extracted later. Then we specify an interval for which any calculated result will be set equal to zero. The width of the interval depends on the computer. A generally safe rule is to consider any result that appears only in the last two digits of the calculation to be zero. For a typical microcomputer with nine-digit accuracy before rounding, G should be set equal to zero whenever its calculated absolute value is less than 10^{-7}.

The program follows. Line 1 is the title. Line 2 is the general quadratic equation. Line 10 generates a blank line between the output for successive cases. Line 20 calls for the values of the coefficients, and lines 30 through 60 print their values with a heading. Lines 70 and 80 calculate e and G. Lines 90 and 100 set G equal to zero if its calculated value does not exceed the stipulated limit. (It does not matter whether we use $>$ or $\geq$ in line 90, because the limit is arbitrary anyhow.) Line 110 generates a blank line between the coefficients and the roots. Line 120 prints a heading for the roots. Line 130 tests to see whether the roots are real or complex. If real, they are printed by lines 140 and 150; if complex, they are printed by lines 170 and 180. The execution is returned to the beginning by line 160 or 190.

```
1  REM: ROOTS OF A QUADRATIC EQUATION
2  REM: A*X^2+B*X+C=0
10 PRINT                                   Generates blank line.
```

```
 20  INPUT "ENTER A,B,C,";A,B,C
 30  PRINT "THE COEFFICIENTS OF
     THE QUADRATIC EQUATION ARE:"
 40  PRINT "A=";A
 50  PRINT "B=";B
 60  PRINT "C=";C
 70  E=-B/2/A
 80  G=E*E-C/A
 90  IF ABS(G)>10^-7 THEN 110
100  G=0
110  PRINT
120  PRINT "THE TWO ROOTS ARE:"
130  IF G<0 THEN 170
140  PRINT E-SQR(G)
150  PRINT E+SQR(G)
160  GOTO 10
170  PRINT E;"+";SQR(-G);"I"
180  PRINT E;"-";SQR(-G);"I"
190  GOTO 10
```

Calls for values of coefficients.

Prints coefficients with heading.

Calculates E and G.

Assigns exact 0 if G is small.

Generates blank line.

Prints heading for roots.

Transfers execution if roots are complex.

Calculates and prints roots if real.

Returns for further input.

Calculates and prints roots if complex.

Returns for further input.

This program may be used as it stands on almost any microcomputer in common use, with the exception of one line. The appropriate value of the constant in line 90 depends on the computer. The number 10^{-7} is suitable for a typical microcomputer with nine-digit accuracy, such as the Apple IIe or II Plus or the Commodore 64. For the TRS-80 with seven-digit accuracy, the constant should be 10^{-5}; for the TI-99/4 with thirteen-digit accuracy, a value of 10^{-11} may be used.

We consider a few examples. For the equation

$$x^2 - 5x + 6 = 0$$

we obtain the roots 2 and 3. For the equation

$$4x^2 - 4x + 5 = 0$$

we obtain the roots $.5 + i$ and $.5 - i$. For the equation

$$x^2 - 6x + 9 = 0$$

we obtain the double root 3.

2-6. Cubic Equations

Solutions of cubic and quartic equations can be found in many books on the theory of equations, but they are not in a form suitable for automatic computation. For this reason, we must give the derivations in this section

and the next in detail. Any reader who is not interested in the theory may skip the derivations and proceed directly to the programs.

In this section we consider Cardan's solution of the cubic equation

$$ax^3 + bx^2 + cx + d = 0 \tag{2-22}$$

We assume that the coefficients a, b, c, and d are real. Also, we exclude the trivial case $a = 0$, which is really a quadratic equation. We divide through the equation by a and make the substitutions

$$e = \frac{b}{3a} \qquad x = \mathrm{u} - \mathrm{e} \tag{2-23a,b}$$

The resulting equation is

$$u^3 + 3pu + 2q = 0 \tag{2-24}$$

where

$$p = \frac{c}{3a} - e^2 \tag{2-25a}$$

$$q = \frac{d - ce}{2a} + e^3 = \frac{d}{2a} - e\left(p + \frac{c}{6a}\right) \tag{2-25b}$$

To solve Eq. 2-24, we make the further substitution

$$u = v - \frac{p}{v} \tag{2-26}$$

This leads to

$$v^6 + 2qv^3 - p^3 = 0$$

which is a quadratic equation for v^3. The solution is

$$v^3 = -q \pm (q^2 + p^3)^{1/2}$$

We make the substitutions

$$F = q^2 + p^3 \tag{2-27}$$

$$G = (-q + \sqrt{F})^{1/3} \qquad H = (-q - \sqrt{F})^{1/3} \tag{2-28a,b}$$

The parameter F is known as the cubic discriminant. The roots are

$$v = G,\ G\omega,\ G\omega^2,\ H,\ H\omega,\ H\omega^2$$

where ω and ω^2 are the complex cube roots of 1 given by the equations

$$\omega = \frac{-1+\sqrt{-3}}{2} \qquad \omega^2 = \frac{-1-\sqrt{-3}}{2} \tag{2-29a,b}$$

By substituting the six values of v into Eq. 2-26 and observing that $GH = -p$, we obtain the following three distinct values of u:

$$u = G + H,\ G\omega + H\omega^2,\ G\omega^2 + H\omega$$

With the help of Eqs. 2-29 and 2-23b, it follows that

$$x_1 = G + H - e \tag{2-30a}$$

$$x_2 = -\frac{1}{2}(G+H) - e + \frac{i\sqrt{3}}{2}(G-H) \tag{2-30b}$$

$$x_3 = -\frac{1}{2}(G+H) - e - \frac{i\sqrt{3}}{2}(G-H) \tag{2-30c}$$

The character of the solution depends on the sign of the cubic discriminant F. If $F > 0$, it is clear that the values of G and H given by Eqs. 2-28 are real. Then the root x_1 is real, and the roots x_2 and x_3 are complex. If $F < 0$, it can be seen from Eqs. 2-28 that G^3 and H^3 are complex conjugates. The same thing must be true of G and H. Therefore $G + H$ is real, and $G - H$ is a pure imaginary number. It follows from Eqs. 2-30 that all three roots are real. To find the roots, a trigonometric solution is necessary. We write the parameter G^3 in polar form. Thus

$$G^3 = -q + i\sqrt{-F} = \rho(\cos\phi + i\sin\phi)$$

It follows that

$$\rho^2 = q^2 - F = -p^3$$

and

$$\phi = \arccos\frac{-q}{(-p)^{3/2}}$$

It is clear that p is negative when F is negative, so the expression $(-p)^{3/2}$ will not cause any trouble. Since standard BASIC does not provide an inverse cosine function, we use the elementary identity

$$\arccos t = \frac{\pi}{2} - \arctan\left(\frac{t}{\sqrt{1-t^2}}\right)$$

Then

$$\phi = \frac{\pi}{2} + \arctan \frac{q}{\sqrt{-F}} \tag{2-31}$$

By solving for G, remembering that H is the complex conjugate of G, and substituting the results into Eqs. 2-30, we arrive at the result

$$x_j = 2\sqrt{-p} \cos \frac{\phi + 2j\pi}{3} - e \qquad j = 1,2,3 \tag{2-32}$$

The numerical evaluation is straightforward. The parameters p and q are found from Eqs. 2-25, F is found from Eq. 2-27, ϕ is found from Eq. 2-31, and the three roots are given by Eq. 2-32.

If $F = 0$, it follows from Eqs. 2-28 that G and H are real and equal. Then the roots given by Eqs. 2-30 are all real, with a double root $x_2 = x_3$. This case will be combined with the real case $F < 0$. However, Eq. 2-31 breaks down when $F = 0$. It is clear that the appropriate equation for this limit is

$$\phi = \frac{\pi}{2}(1 + \operatorname{sgn} q) \tag{2-33}$$

The analysis of the case $F = 0$ is identical to that of the case $F < 0$, except that Eq. 2-33 is used instead of 2-31.

We point out an important feature of Eq. 2-32: The three roots are in ascending numerical order. To see this we observe that the principal value of ϕ given by Eq. 2-31 or 2-33 lies in the interval $0 \leq \phi \leq \pi$. Then it follows that

$$-1 \leq \cos \frac{\phi + 2\pi}{3} \leq -\frac{1}{2} \leq \cos \frac{\phi + 4\pi}{3} \leq \frac{1}{2} \leq \cos \frac{\phi + 6\pi}{3} \leq 1$$

Therefore $x_1 \leq x_2 \leq x_3$.

We now return to the case in which $F > 0$. Then G and H are real. The root given in Eq. 2-30a is real, and the roots given in Eqs. 2-30b,c are complex. Equations 2-30 could be used as they stand, but a revised set of equations leads to a neater program. We introduce the new parameters

$$K = (|q| + \sqrt{F})^{1/3} \qquad L = (|q| - \sqrt{F})^{1/3} \tag{2-34a.b}$$

Comparison of Eqs. 2-28 and 2-34 shows that G and H are related to K and L as follows:

$q \leq 0$: $G = K$, $H = L$

$q \geq 0$: $G = -L$, $H = -K$

Equations 2-30 now become

$$x_1 = -(K + L)\text{sgn } q - e \tag{2-35a}$$

$$x_2 = \frac{1}{2}(K + L)\text{sgn } q - e + \frac{i\sqrt{3}}{2}(K - L) \tag{2-35b}$$

$$x_3 = \frac{1}{2}(K + L)\text{sgn } q - e - \frac{i\sqrt{3}}{2}(K - L) \tag{2-35c}$$

Equation 2-34b is never used directly; we deduce from Eqs. 2-34 and 2-27 that $KL = -p$ and therefore that

$$L = -\frac{p}{K}$$

This division is always possible. It breaks down only if $K = p = q = F = 0$. Since we are now considering the complex case $F > 0$, there is no difficulty. For this reason, we use the parameters K and L instead of G and H; the last two are sometimes equal to zero. We now rewrite Eqs. 2-35 in the final form in which they will be used in the program:

$$x_1 = \left(\frac{p}{K} - K\right)\text{sgn } q - e \tag{2-36a}$$

$$x_r = -\frac{1}{2}(x_1 + 3e) \tag{2-36b}$$

$$x_i = \frac{\sqrt{3}}{2}\left(\frac{p}{K} + K\right) \tag{2-36c}$$

$$x_2 = x_r + ix_i \tag{2-36d}$$

$$x_3 = x_r - ix_i \tag{2-36e}$$

The subscripts r and i stand for real and imaginary, respectively. The desired results are given by Eqs. 2-36a,d,e.

The program follows. Line 1 is the title. Line 2 is the general cubic equation. Line 10 generates a blank line between successive cases. Line 20 calls for the values of the coefficients. Lines 30 through 70 print the values of the coefficients, with a heading. Lines 80 through 110 represent Eqs. 2-23a, 2-25a, 2-25b, and 2-27. Lines 120 and 130 set F equal to 0 if its calculated absolute value does not exceed some specified limit (Sec. 2-5). Line 140 generates a blank line between the values of the coefficients and the roots. Line 150 prints a heading for the roots. Line 160 tests the value of F to see whether the roots are real or complex. If they are real, they are evaluated and printed by lines 170 through 250. Lines 190, 210, and 230 represent Eqs. 2-31, 2-33, and 2-32, respectively. Line 260 returns the execution of the program to the beginning in preparation for

further input. If the roots are complex, they are evaluated and printed by lines 270 through 330. Lines 280, 290, and 300 represent Eqs. 2-36a,b,c. Line 310 prints the value of x_1, and lines 320 and 330 print the values of x_2 and x_3 according to Eqs. 36d,e. Line 340 returns the execution of the program to the beginning in preparation for further input.

```
  1 REM: ROOTS OF A CUBIC EQUATION
  2 REM: A*X^3+B*X^2+C*X+D=0
 10 PRINT                                    Generates blank line.
 20 INPUT "ENTER A,B,C,D ";A,B,C,D           Calls for values of coefficients.
 30 PRINT "THE COEFFICIENTS OF
    THE CUBIC EQUATION ARE:"
 40 PRINT "A=";A                             Prints coefficients
 50 PRINT "B=";B                             with heading.
 60 PRINT "C=";C
 70 PRINT "D=";D
 80 E=B/3/A                                  Calculates constants
 90 P=C/3/A-E*E                              E, P, Q, F.
100 Q=D/2/A-E*(P+C/6/A)
110 F=Q*Q+P*P*P
120 IF ABS(F)>10^-7 THEN 140                Assigns exact 0
130 F=0                                     if F is small.
140 PRINT                                   Generates blank line.
150 PRINT "THE THREE ROOTS ARE:"            Prints heading for roots.
160 IF F>0 THEN 270                         Transfers execution if roots
                                            are complex.
170 PI=4*ATN(1)
180 IF F=0 THEN 210
190 PHI=PI/2+ATN(Q/SQR(-F))                 Calculates and prints
200 GOTO 220                                roots if real.
210 PHI=PI/2*(1+SGN(Q))
220 FOR J=1 TO 3
230 X=2*SQR(-P)*COS((PHI+2*J*PI)/3)-E
240 PRINT X
250 NEXT J
260 GOTO 10                                 Returns for further input.
270 K=(ABS(Q)+SQR(F))^(1/3)
280 X1=(P/K-K)*SGN(Q)-E
290 XR= -(X1+3*E)/2                         Calculates and prints
300 XI=(P/K+K)*SQR(3)/2                     roots if complex.
310 PRINT X1
320 PRINT XR;"+";XI;"I"
330 PRINT XR;"-";XI;"I"
340 GOTO 10                                 Returns for further input.
```

This program runs as it stands on almost any microcomputer in common use. However, the constant in line 120 should be adjusted to fit the accuracy

of the computer, as discussed in Sec. 2-5. Also, a few computers, such as the Commodore 64, have a built-in constant for π. When this feature is available, it may be used instead of line 170. However, the program can also be used as it stands.

We consider a few examples. For the equation

$$x^3 - 6x^2 + 11x - 6 = 0$$

we obtain the roots 1, 2, 3. For the equation

$$x^3 - x^2 - 4x - 6 = 0$$

we obtain the roots 3, $-1 + i$, $-1 - i$. For the equation

$$x^3 - 4x^2 + 5x - 2 = 0$$

we obtain the roots 1, 1, 2.

This program gives three real roots in ascending order, or one real root followed by two complex roots. It can easily be seen from the derivation that this is true in general.

2-7. Quartic Equations

The general fourth-degree algebraic equation is

$$ax^4 + bx^3 + cx^2 + dx + c = 0 \tag{2-37}$$

We assume that the coefficients are real. Also, we exclude the trivial case $a = 0$, which is really a cubic equation. We shall use Ferrari's method of solution. The basic concept is similar to that used for the quadratic equation. We divide through by a, and then complete the square. However, we shall make one change in the format. Instead of retaining the original coefficients throughout the analysis, we start by rewriting Eq. 2-37 as

$$x^4 + Bx^3 + Cx^2 + Dx + E = 0 \tag{2-38}$$

It is not possible in the program to distinguish between upper-case and lower-case characters. However, this will not cause any difficulty, because the original coefficients are never used after the input. We now complete the square by adding the expression $(rx + s)^2$ to both sides of Eq. 2-38. The resulting equation will have the form

$$\left(x^2 + \frac{B}{2}x + \frac{t}{2}\right)^2 = (rx + s)^2 \tag{2-39}$$

where r, s, and t are real constants whose values will be determined. By collecting like powers of x and then equating coefficients of correspond-

ing terms in Eqs. 2-38 and 2-39, we arrive at the following set of equations for r, s, and t:

$$r^2 = t + \frac{B^2}{4} - C \tag{2-40a}$$

$$rs = \frac{Bt}{4} - \frac{D}{2} \tag{2-40b}$$

$$s^2 = \frac{t^2}{4} - E \tag{2-40c}$$

We multiply the first equation by the third and equate the result to the square of the second. This leads to the following equation for t:

$$t^3 - Ct^2 + (BD - 4E)t + 4CE - D^2 - B^2E = 0 \tag{2-41}$$

The substitution

$$t = u + \frac{C}{3} \tag{2-42}$$

leads to

$$u^3 + 3pu + 2q = 0 \tag{2-43}$$

where

$$p = \frac{1}{3}\left(BD - \frac{C^2}{3} - 4E\right) \tag{2-44a}$$

$$q = \frac{C}{6}\left(BD - \frac{2}{9}C^2 + 8E\right) - \frac{1}{2}(B^2E - D^2) \tag{2-44b}$$

Equation 2-43 is identical to Eq. 2-24, which has already been solved. We also need the cubic discriminant F. This has been given in Eq. 2-27 as

$$F = q^2 + p^3 \tag{2-45}$$

With u known, the values of t, r, and s follow from Eqs. 2-42 and 2-40. Equation 2-39 breaks down into the two quadratic equations

$$x^2 + \left(\frac{B}{2} + r\right)x + \frac{t}{2} + s = 0 \tag{2-46}$$

$$x^2 + \left(\frac{B}{2} - r\right)x + \frac{t}{2} - s = 0 \tag{2-47}$$

which can easily be solved for x.

In principle we now have all the elements of the solution of Eq. 2-37. However, some further work is necessary before a numerical evaluation can be carried out. The numerical analysis starts with the evaluation of t. Since Eq. 2-43 for u is identical to 2-24, this evaluation is almost exactly the same as that given previously for the cubic equation, except that Eqs. 2-44 for p and q take the place of 2-25. However, we now need only one root. To choose the appropriate root, we observe that, if we expect to obtain useful results from the quadratic Eqs. 2-46 and 2-47, the parameters r, s, and t must be real. If $F > 0$, there is only one real value of t. This is

$$t = \left(\frac{p}{K} - K\right) \operatorname{sgn} q + \frac{C}{3} \tag{2-48}$$

which is adapted from Eq. 2-36a by using 2-42 instead of 2-23b. If $F \leq 0$, there are three real values of u and t, but not all of them necessarily correspond to real values of r and s.* Inspection of Eqs. 2-40 shows that the largest value of t is the one that we need. Then we have

$$t = 2\sqrt{-p} \cos\frac{\phi}{3} + \frac{C}{3} \tag{2-49}$$

which is adapted from Eq. 2-32 with $j = 1$

With t known, r is found from Eq. 2-40a. The sign of r is immaterial; a reversal of sign would merely interchange Eqs. 2-46 and 2-47. We choose the positive sign. Some care must be taken in evaluating s. With t and r known, either Eq. 2-40b or 2-40c can be used to find s. However, neither equation alone is entirely adequate. The sign of s must be consistent with the sign of r, but Eq. 2-40c gives only the magnitude. Equation 2-40b gives the magnitude and the sign, but it breaks down when $r = 0$. We use Eq. 2-40b if $r \neq 0$ and 2-40c if $r = 0$. (If $r = 0$, the sign of s is immaterial.)

With r, s, and t known, the four required values of x are obtained from the quadratic Eqs. 2-46 and 2-47. Actually we use the equation

$$x^2 + 2mx + n = 0 \tag{2-50}$$

where the coefficients m and n are chosen to fit Eqs. 2-46 and 2-47. For Eq. 2-46, it is clear that

* The quartic Eq. 2-38 can be decomposed into the quadratic Eqs. 2-46 and 2-47 in three ways. Let the four roots be x_1, x_2, x_3, and x_4. Then we may have x_1 paired with x_2 and x_3 with x_4, or x_1 with x_3 and x_2 with x_4, or x_1 with x_4 and x_2 with x_3. Each combination corresponds to one of the three roots of the cubic Eq. 2-41. It is possible for Eq. 2-41 to have three real roots at the same time that the quartic 2-37 or 2-38 has complex roots. In this case only one of the three values of t—the one that groups the complex roots of the quartic into conjugate pairs—will lead to real coefficients in the quadratic Eqs. 2-46 and 2-47, and thus to a workable solution.

$$m = \frac{B}{4} + \frac{r}{2} \qquad n = \frac{t}{2} + s \tag{2-51a,b}$$

and for Eq. 2-47

$$m = \frac{B}{4} - \frac{r}{2} \qquad n = \frac{t}{2} - s \tag{2-52a,b}$$

The solution of Sec. 2-5 is used. Also, we need the quadratic discriminant

$$G = m^2 - n \tag{2-53}$$

For quadratic and cubic equations, we obtained exact zeros of the discriminant by using the IF-THEN statement to set it equal to 0 if its calculated value fell within some specified interval. For a quartic equation, we have the two discriminants F and G. The latter appears twice in the program. Zero values of r^2 and s^2 are also critical, because the square roots of these are used. Instead of using the IF-THEN statement five times to set these parameters equal to 0 if their calculated values fall within some specified interval, we shall adopt a slightly neater method. This is based on the fact that a computer cannot handle numbers with very large or very small absolute values. The occurrence of a very large number (an overflow) causes a computer to display an error message and stop running. The occurrence of a very small number (an underflow) does not generate an error message; the number is set equal to zero and the execution proceeds. We may take advantage of this by intentionally creating an underflow to get rid of a small error term. The procedure is to first divide the calculated result by some very large number—say Z—and then multiply it by the same number. If the calculated result is very small, the final result will be exactly zero. Otherwise the original value will be recovered. The appropriate value of Z depends on the computer.

Consider a typical Microsoft computer such as the Apple IIe or the Commodore 64. The acceptable interval for the absolute value of a number runs from approximately 10^{-38} to 10^{38}. The accuracy is approximately nine digits. To eliminate a result that differs from 0 only in the last two digits, we set $Z = 10^{31}$. For the TRS-80, with the same range but only seven-digit accuracy, we set $Z = 10^{33}$. For the TI-99/4, with a range of 10^{-128} to 10^{128} and thirteen-digit accuracy, we set $Z = 10^{117}$.

The program follows. Line 1 is the title. Line 2 is the general quartic equation. Line 10 generates a blank line between successive cases. Line 20 calls for the values of the coefficients of Eq. 2-37. Lines 30 through 80 print the coefficients, with a heading. Lines 90 through 120 calculate the coefficients of the modified Eq. 2-38. Line 130 assigns a value to the parameter Z. This is used to create an underflow when an exact zero result is required, as discussed. Lines 140, 150, and 160 represent Eqs 2-44a,b and 2-45. Line 170 generates a blank line between the values of the coefficients and the roots. Line 180 prints a heading for the roots.

Lines 190 through 260 calculate t; this segment is similar to the cubic program of Sec. 2-6. Lines 270 through 310 calculate r and s, using Eqs. 2-40. Lines 320 and 330 calculate the coefficients m and n, using Eqs. 2-51. Lines 350 and 360 calculate and test the quadratic discriminant G. If the roots are real, they are calculated and printed by lines 410 and 420. If they are complex, lines 370 through 390 switch the analysis from Eq. 2-46 to Eq. 2-47. In this way the real roots (if any) are obtained first. If the new roots are also complex, line 400 transfers the execution to lines 440 and 450, which print them.

In the programs for quadratic equations and cubic equations, all of the roots were found at once. In this program only two roots are found at one time. Hence it is necessary to count the roots as the execution proceeds and to end the calculation when the number reaches four. This is done in lines 340, 460, and 470. Line 480 returns the execution of the program to the beginning in preparation for further input. With any computer that has an ELSE statement, lines 470 and 480 may be combined.

```
 1  REM: ROOTS OF A QUARTIC EQUATION
 2  REM: A*X^4+B*X^3+C*X^2+D*X+E=0
10  PRINT                                   Generates blank line.
20  INPUT "ENTER A,B,C,D,E ";               } Calls for values
    A,B,C,D,E                               } of coefficients.
30  PRINT "THE COEFFICIENTS OF              }
    THE QUARTIC EQUATION ARE:"              }
40  PRINT "A=";A                            }
50  PRINT "B=";B                            } Prints coefficients
60  PRINT "C=";C                            } with heading.
70  PRINT "D=";D                            }
80  PRINT "E=";E                            }
90  B=B/A                                   }
100 C=C/A                                   } Calculates coefficients
110 D=D/A                                   } of Eq. 2-38.
120 E=E/A                                   }
130 Z=(See text.)                           Assigns value of underflow parameter.
140 P=(B*D-C*C/3-4*E)/3                     }
150 Q=C*(B*D-2*C*C/9+8*E)/6                 } Calculates P, Q, F.
      -(B*B*E+D*D)/2                        }
160 F=(Q*Q+P*P*P)/Z*Z                       }
170 PRINT                                   Generates blank line.
180 PRINT "THE FOUR ROOTS ARE:"             }
190 ON SGN(F)+2 GOTO 200,220,250            }
200 PHI=2*ATN(1)+ATN(Q/SQR(-F))             }
210 GOTO 230                                }
220 PHI=2*ATN(1) *(1+SGN(Q))                } Solves cubic equation for t.
230 T=2*SQR(-P)*COS(PHI/3)+C/3              }
240 GOTO 270                                }
250 K=(ABS(Q)+SQR(F))^(1/3)                 }
260 T=(P/K-K)*SGN(Q)+C/3                    }
```

```
270 R=SQR((T+B*B/4-C)/Z*Z)
280 IF R=0 THEN 310
290 S=(B*T/2-D)/2/R
300 GOTO 320
310 S=SQR((T*T/4-E)/Z*Z)
320 M=B/4+R/2
330 N=T/2+S
340 J=0
350 G=(M*M-N)/Z*Z
360 IF G>=0 THEN 410
370 M=B/2-M
380 N=T-N
390 G=(M*M-N)/Z*Z
400 IF G<0 THEN 440
410 PRINT -M-SQR(G)
420 PRINT -M+SQR(G)
430 GOTO 460
440 PRINT -M;"+";SQR(-G);"I"
450 PRINT -M;"-";SQR(-G);"I"
460 J=J+2
470 IF J<4 THEN 370
480 GOTO 10
```

Calculates r and s.

Calculates m and n.

Initializes number of roots.
Calculates quadratic discriminant.
Transfers execution if roots are real.
Switches coefficients of quadratic equation.
Calculates quadratic discriminant.
Transfers execution if roots are complex.
Calculates and prints real roots.
Transfers execution.
Calculates and prints complex roots.
Counts roots.
Returns for remaining roots.
Returns for further input.

A few results are given in the following table:

a	b	c	d	e	*Roots*
1	−4	−1	16	−12	−2, 1, 2, 3
1	−2	−3	8	−4	−2, 1, 1, 2
2	5	−8	−17	−6	−3,−1,−.5, 2
1	4	6	4	−15	$-3,\ 1,\ -1\pm 2i$
1	2	2	10	25	$1\pm 2i,\ -2\pm i$

It can be shown that this program always gives real roots in ascending order, followed by complex roots.

Problems

Solve equations 2-1 through 2-6 numerically.

2-1. $x + \ln x = 5$

2-2. $3x - 4 \sin x = 2$

2-3. $e^{-x} + \sin x - 5x + 2 = 0$

2-4. $e^x = (5 - x)^3$

2-5. $x = 2 \sin x$

2-6. $\cosh x = 2 \cos x$

2-7. Find the smallest positive nonzero root of the equation $\tan x = 2x$.

2-8. Use the Newton-Raphson method to evaluate $\sqrt{5}$.

2-9. Use the Newton-Raphson method to evaluate $\sqrt[3]{5}$.

2-10. Verify the solutions to the three quadratic equations given at the end of Sec. 2-5.

2-11. Verify the solutions to the three cubic equations given at the end of Sec. 2-6.

2-12. Solve the following equations:

a. $x^3 - 4x^2 + 3x + 1 = 0$
b. $x^3 - 18.1x - 34.8 = 0$
c. $x^3 + 2x^2 + 10x - 20 = 0$

2-13. It is shown in the theory of elasticity that the principal stresses in a three-dimensional body are given by the three roots of the equation

$$\sigma^3 - (\sigma_x + \sigma_y + \sigma_z)\sigma^2 + (\sigma_x\sigma_y + \sigma_y\sigma_z + \sigma_z\sigma_x - \tau_{xy}^2 - \tau_{yz}^2 - \tau_{zx}^2)\sigma - (\sigma_x\sigma_y\sigma_z + 2\tau_{xy}\tau_{yz}\tau_{zx} - \sigma_x\tau_{yz}^2 - \sigma_y\tau_{zx}^2 - \sigma_z\tau_{xy}^2) = 0$$

where σ_x, σ_y, and σ_z are the tensile stresses in the x, y, and z directions, respectively, and the τs are the shear stresses. Modify the program of Sec. 2-6 to solve this problem directly, using σ_x, σ_y, σ_z, τ_{xy}, τ_{yz}, and τ_{zx} as input. Check the program for the following values:

$\sigma_x = 100$ $\quad \sigma_y = -80$ $\quad \sigma_z = 50$
$\tau_{xy} = 20$ $\quad \tau_{yz} = 30$ $\quad \tau_{zx} = -10$

2-14. Verify the solutions to the five quartic equations given at the end of Sec. 2-7.

2-15. Solve the following equations:

a. $x^4 - x^3 - 3x^2 + 2x + 2 = 0$
b. $x^4 + x^3 - 7x^2 - 4x + 6 = 0$
c. $x^4 - 3x^3 + 6x^2 - 3x - 5 = 0$
d. $x^4 - 5x^3 + 11x^2 - 14x + 4 = 0$

2-16. Solve the equation:

$$x^6 - x^5 + x^4 - 7x^3 - 8x^2 - 34x - 24 = 0$$

3

Some Higher Transcendental Functions

The elementary trigonometric, exponential, and logarithmic functions are provided with the computer. When more advanced functions are required, it is necessary to write programs for them. In this chapter we shall develop programs for a number of commonly occurring functions.

3-1. The Sine Integral and the Cosine Integral

The sine integral $\mathrm{Si}(x)$ is defined by the equation

$$\mathrm{Si}(x) = \int_0^x \frac{\sin t}{t}\, dt \tag{3-1}$$

By expanding the integrand into an infinite series and integrating term by term, we find that

$$\mathrm{Si}(x) = x - \frac{x^3}{3 \cdot 3!} + \frac{x^5}{5 \cdot 5!} - \frac{x^7}{7 \cdot 7!} + \cdots \tag{3-2}$$

There are several possible nested forms of this equation. The most convenient one for our purpose is

$$\mathrm{Si}(x) = x\left(1 - \frac{x^2}{2 \cdot 3}\left(\frac{1}{3} - \frac{x^2}{4 \cdot 5}\left(\frac{1}{5} - \frac{x^2}{6 \cdot 7}\left(\frac{1}{7} - \cdots\right)\right)\right)\right) \tag{3-3}$$

The number of terms n needed to obtain satisfactory convergence depends on the value of x. It is desirable to include an automatic provision in the program to determine the appropriate value of n, rather than try several values manually for each value of x. We use the equation

$$n = \mathrm{Int}(1.5x + 6) \tag{3-4}$$

This nomenclature means that the result on the right side of the equation is truncated to the greatest integer that does not exceed the value of the expression in parentheses. Equation 3-4 guarantees that the error due to the neglect of higher-order terms in the series will not affect the first ten significant figures of the result. (This equation is obtained by trial and error, trying various values of n until further increases no longer affect the result.) The same criterion of accuracy is followed throughout this chapter. This, of course, does not guarantee that the results will be accurate to ten significant figures. The accuracy of the results depends on the accuracy of the computer as well as on the number of terms retained in the series.

The program follows. Line 1 is the title. Line 10 generates a blank line between successive sets of data. Line 20 calls for the input x, using a prompting message, and prints the response on the screen. Line 30 calculates n, the required number of terms of the series, using Eq. 3-4. Line 40 calculates the number at the extreme right of Eq. 3-3, and the next three lines constitute a FOR-NEXT loop that calculates S, the sum of the nested series, proceeding from right to left and ending with the 1 inside the left parenthesis. Line 80 calculates and prints the final result, $\mathrm{Si}(x)$. Line 90 is optional; if results are required for a number of values of x, this line makes it possible to obtain them without typing RUN each time. The execution of the program is terminated by pressing the BREAK key.

```
 1  REM: SINE INTEGRAL
10  PRINT                              Generates blank line.
20  INPUT "X=";X                       Calls for value of x.
30  N=INT(1.5*X+6)                     Calculates n.
40  S=1/(2*N-1)                        Initializes S.
50  FOR J=2*N-3 TO 1 STEP -2           }
60  S=1/J-X*X/(J+1)/(J+2)*S            } Calculates S.
70  NEXT J                             }
80  PRINT "SI(X)=";X*S                 Calculates and prints result.
90  GOTO 10                            Returns for new input.
```

Accurate numerical results from reference 1 appear in the following table:

x	Si(x)	x	Si(x)	x	Si(x)
0	0.00000 0000	4	1.75820 3139	8	1.57418 6822
1	0.94608 3070	5	1.54993 1245	9	1.66504 0076
2	1.60541 2977	6	1.42468 7551	10	1.65834 7594
3	1.84865 2528	7	1.45459 6614		

Results obtained by running the program on the TI-99/4 microcomputer are identical to those shown. Other computers give essentially the same results, although the last few digits may be lost, depending on the accuracy of the model used. These remarks apply to all the programs of this chapter, unless otherwise noted.

The cosine integral (Ci(x) is defined by the equation

$$\text{Ci}(x) = -\int_x^\infty \frac{\cos t}{t}\, dt \tag{3-5}$$

In this case the series expansion is not elementary. It can be shown that*

$$\text{Ci}(x) = \gamma + \ln x - \frac{x^2}{2 \cdot 2!} + \frac{x^4}{4 \cdot 4!} - \frac{x^6}{6 \cdot 6!} + \ldots \tag{3-6}$$

where γ is Euler's constant. The nested form of this equation is

$$\text{Ci}(x) = \gamma + \ln x - \frac{x^2}{1 \cdot 2}\left(\frac{1}{2} - \frac{x^2}{3 \cdot 4}\left(\frac{1}{4} - \frac{x^2}{5 \cdot 6}\left(\frac{1}{6} - \ldots\right)\right)\right) \tag{3-7}$$

The required number of terms is again given by Eq. 3-4.

The program follows. It corresponds very closely to the foregoing program for the sine integral.

```
  1 REM: COSINE INTEGRAL
 10 PRINT                                  Generates blank line.
 20 INPUT "X=";X                           Calls for value of x.
 30 N=INT(1.5*X+6)                         Calculates n.
 40 S=1/2/N                                Initializes S.
 50 FOR J=2*N-2 TO 2 STEP -2               }
 60 S=1/J-X*X/(J+1)/(J+2)*S                } Calculates S.
 70 NEXT J                                 }
 80 CI=.5772156649+LOG(X)-X*X*S/2          Calculates result.
 90 PRINT "CI(X)=";CI                      Prints result.
100 GOTO 10                                Returns for new input.
```

* All the algorithms used in this chapter can be found in reference 1, unless other references are cited. Derivations can be found in reference 3.

Accurate numerical results from reference 1 appear in the following table. Results from the program are identical to those shown.

x	Ci(x)	x	Ci(x)	x	Ci(x)
1	.33740 39229	5	−.19002 97497	9	.05534 75313
2	.42298 08288	6	−.06805 72439	10	−.04545 64330
3	.11962 97860	7	.07669 52785		
4	−.14098 16979	8	.12243 38825		

A remark about the utility of the two foregoing programs may be of interest. If we need a few values of Si(x) or Ci(x), it is easier to look them up in a handbook than to write a program. However, integrals of this type sometimes appear as intermediate steps in more complicated analyses. In this case it is not convenient to interrupt the execution, look up the value of the required function, and enter it into the computer for each run. Programs of the type given here are used as program segments or subroutines in bigger programs. The same remark applies to all the programs of this chapter.

3-2. The Exponential Integrals

The exponential integral Ei(x) is defined by the equation

$$\text{Ei}(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t}\,dt \tag{3-8}$$

The series expansion is

$$\text{Ei}(x) = \gamma + \ln x + \frac{x}{1\cdot 1!} + \frac{x^2}{2\cdot 2!} + \frac{x^3}{3\cdot 3!} + \cdots \tag{3-9}$$

In nested form, this becomes

$$\text{Ei}(x) = \gamma + \ln x + x\left(1 + \frac{x}{2}\left(\frac{1}{2} + \frac{x}{3}\left(\frac{1}{3} + \frac{x}{4}\left(\frac{1}{4} + \cdots\right)\right)\right)\right) \tag{3-10}$$

The equation for the number of terms n is

$$n = \text{Int}(2.3x + 12)$$

The program follows. Everything except line 100 corresponds closely to the program for the cosine integral in Sec. 3-1. Line 100 prints the value of $xe^{-x}\,\text{Ei}(x)$, which is needed to check the program results against accurate results from reference 1. After it has been verified that the program is running properly, this line may be deleted.

```
  1 REM: EXPONENTIAL INTEGRAL
 10 PRINT                                    Generates blank line.
 20 INPUT "X=";X                             Calls for value of x.
 30 N=INT(2.3*X+12)                          Calculates n.
 40 S=1/N                                    Initializes S.
 50 FOR J=N-1 TO 1 STEP -1                   }
 60 S=1/J+X/(J+1)*S                          }Calculates S.
 70 NEXT J                                   }
 80 EI=.5772156649 + LOG(X)+X*S              Calculates EI(x).
 90 PRINT "EI(X)=";EI                        }
100 PRINT "XEXP(-X)EI(X)=";                  }Prints results.
    X/EXP(X)*EI                              }
110 GOTO 10                                  Returns for new input.
```

Accurate numerical results from reference 1 appear in the following table. Results from the program agree with those shown.

x	$xe^{-x}\mathrm{Ei}(x)$	x	$xe^{-x}\mathrm{Ei}(x)$	x	$xe^{-x}\mathrm{Ei}(x)$
1	0.69717 4883	5	1.35383 1278	9	1.15275 9209
2	1.34096 5420	6	1.27888 3860	10	1.13147 0205
3	1.48372 9204	7	1.22240 8053		
4	1.43820 8032	8	1.18184 7987		

The exponential integral $E_1(x)$ is defined by the equation

$$\mathrm{E}_1(x) = \int_x^\infty \frac{e^{-t}}{t}\,dt \tag{3-12}$$

This integral is more difficult to evaluate numerically than the ones that we have considered previously. The most commonly used series expansion is

$$\mathrm{E}_1(x) = -\gamma - \ln x + \frac{x}{1\cdot 1!} - \frac{x^2}{2\cdot 2!} + \frac{x^3}{3\cdot 3!} - \cdots \tag{3-13}$$

which is obtained from the integral representation

$$\mathrm{E}_1(x) = -\gamma - \ln x + \int_0^x \frac{1-e^{-t}}{t}\,dt$$

It is difficult to obtain accurate results from Eq. 3-13 unless x is small, because $E_1(x)$ approaches zero rapidly as x becomes large. At the same time the individual terms of the series become numerically large and alternately positive and negative. Hence the desired result is the small difference of large terms, and is subject to a large roundoff error. The best remedy is to rewrite the integral representation as

$$\mathrm{E}_1(x) = -\gamma - \ln x + e^{-x}\int_0^x \frac{e^x - e^{x-t}}{t}\,dt$$

This generates the series

$$E_1(x) = -\gamma - \ln x + e^{-x}(a_1 x + a_2 x^2 + a_3 x^3 + \ldots) \tag{3-14}$$

The coefficients are given by the equation

$$a_j = \frac{1}{j!}\left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{j}\right) \tag{3-15}$$

This can be evaluated as it stands, but a result with a slightly smaller roundoff error is obtained by a different procedure. We write

$$a_j = \frac{c_j}{(j!)^2} \tag{3-16a}$$

The first few cs are

$$c_1 = 1 \qquad c_2 = 3 \qquad c_3 = 11 \qquad c_4 = 50 \qquad c_5 = 274$$

It is easy to show that the cs are given by the recurrence relations

$$c_j = jc_{j-1} + b_{j-1} \qquad c_0 = 0 \tag{3-16b}$$

$$b_j = j! = jb_{j-1} \tag{3-16c}$$

When we try to write the series in Eq. 3-14 in nested form, a difficulty arises. A nested series is evaluated from right to left. Therefore we need a_n, a_{n-1}, a_{n-2} . . . in that order. However, Eqs. 3-16 give the cs and as in ascending order, starting with c_o. It is desirable to adhere to the nested format, since this is more efficient than direct summation. We rewrite Eq. 3-14 as

$$E_1(x) = -\gamma - \ln x + x^n e^{-x}\left(a_n + \frac{a_{n-1}}{x} + \frac{a_{n-2}}{x^2} + \ldots\right)$$

This can be written in nested form as

$$E_1(x) = -\gamma - \ln x + x^n e^{-x}\left(a_n + \frac{1}{x}\left(a_{n-1} + \frac{1}{x}\left(a_{n-2} + \ldots\right)\right)\right) \tag{3-17}$$

The number of terms required for convergence is

$$n = \text{Int}(2.5x + 16) \tag{3-18}$$

Equation 3-14, as well as Eq. 3-17, which is derived from it, is less subject to roundoff error than Eq. 3-13 because all of the terms of the series are positive. However, there is still some difficulty because $\gamma + \ln x$ is nearly

equal to $x^n e^{-x}(\ldots)$ in Eq. 3-17. The accuracy of the results depends on the computer used. A program based on Eq. 3-17 works well with the TI-99/4; with other models the errors are greater.

The program follows. Line 1 is the title. Line 10 generates a blank line between successive sets of data. Line 20 calls for the input x, using a prompting message, and prints the response on the screen. Line 30 calculates n, the required number of terms of the series, using Eq. 3-18. Lines 40 through 120 evaluate the nested series in Eq. 3-17, and line 130 calculates $E_1 x$. Line 140 prints the result. Line 150 prints the value of $xe^x E_1(x)$. This is needed to check the program results against accurate values from reference 1. Line 160 returns the execution of the program to the beginning in preparation for further input.

```
 1  REM: EXPONENTIAL INTEGRAL E1(X)
10  PRINT                                  Generates blank line.
20  INPUT "X=";X                           Calls for value of x.
30  N=INT(2.5*X+16)                        Calculates n.
40  S=0                                    }
50  B=1                                    } Initializes variables.
60  C=0                                    }
70  FOR J=1 TO N                           }
80  C=J*C+B                                }
90  B=J*B                                  } Calculates S.
100 A=C/B/B                                }
110 S=A+S/X                                }
120 NEXT J                                 }
130 E1=-.5772156649015                     } Calculates E1(X).
    -LOG(X)+X^N/EXP(X)*S                   }
140 PRINT "E1(X)=";E1                      }
150 PRINT "XEXP(X)E1(X)=";                 } Prints results.
    X*EXP(X)*E1                            }
160 GOTO 10                                Returns for new input.
```

Results obtained by running the program on the TI-99/4 and the Apple IIe are shown in the following table:

	$xe^x E_1(x)$			
x	Ref. 1	TI-99/4	Apple IIe	Apple IIe (modified prg.)
1	.59634 7362	.59634 7362	.59634 7364	.59634 7368
2	.72265 7234	.72265 7234	.72265 7221	.72265 7221
3	.78625 1221	.78625 1221	.78625 1435	.78625 1234
4	.82538 2600	.82538 2600	.82538 2415	.82538 2415
5	.85211 0880	.85211 0881	.85211 4991	.85211 1274
6	.87160 5775	.87160 5768	.87163 4899	.87161 8652
7	.88648 7675	.88648 7683	.88666 3229	.88660 4108
8	.89823 7113	.89823 7189	overflow	.89826 9859
9	.90775 7602	.90775 7441	"	.90944 4793
10	.91563 3339	.91563 2694	"	.91525 9189

Results found with the Commodore 64 are identical to those found with the Apple IIe. Accurate results from reference 1 are shown for comparison. Two difficulties are apparent. Even though we have taken precautions to minimize roundoff error, this trouble has not been entirely eliminated. With the TI-99/4, which has thirteen-digit accuracy, results are very good. The error increases as x increases, but it is never large in the interval $0 < x \leq 10$. For the Apple IIe, with nine-digit accuracy, the error is greater. Also, the calculations with the Apple break down when $x > 7$ because of an overflow. Every computer has some upper and lower limits to the size of number that it can handle. On the TI-99/4, the range is from 10^{-128} to 10^{128}. With computers such as the Apple IIe, Commodore 64 and TRS-80 that use Microsoft BASIC, the limits are 10^{-38} and 10^{38}. The parameters a, b, and c are outside these limits when $x > 7$; a is very small, while b and c are very large. Also x^n is large. The overflow can easily be corrected by amending two lines of the program as follows:

```
 50  B=(X/2)^-N
130  E1=-.5772156649015-LOG(X)+2^N/EXP(X)*S
```

Results found by running the amended program on the Apple IIe and the Commodore 64 appear in the last column of the preceding table. The accuracy, although not high, is good enough for many applications.

The foregoing program does not give satisfactory results on the TRS-80 unless x is small (say $x \leq 4$) because of the lower accuracy of this computer. However, a partial remedy can be found by a careful study of the performance characteristics of this machine. For ordinary arithmetic, the results are accurate to seven significant figures. For scientific functions the accuracy is lower and depends on the specific operation performed. (The TRS-80 has a double precision mode that gives results accurate to 17 significant figures. However, this applies only to ordinary arithmetic and not to scientific functions.) For the LOG and EXP functions, the accuracy is only slightly lower than that of ordinary arithmetic, but for exponentiation it is much poorer. Thus the result of using X^N is much poorer than the result obtained by carrying out the multiplication X*X*X* . . . to n factors. We therefore modify the original program to avoid the use of exponentiation. The appropriate changes are:

```
 65  P=1
115  P=X*P
130  E1=-.5772157-LOG(X)+P/EXP(X)*S
```

With this amendment, the TRS-80 gives results accurate to three significant figures provided that $x \leq 7$. For larger values of the argument, an overflow occurs. Although this can be corrected, the results are still unsatisfactory if $x > 7$.

With all the preceding programs for $E_1(x)$, the accuracy deteriorates as x becomes large. We now consider an approximate evaluation that is

very good for large values of x and does not require a highly accurate computer. We start with the integral of Eq. (3-12) and integrate repeatedly by parts. In this way we obtain the results

$$E_1(x) = \int_x^\infty \frac{e^{-t}}{t}\,dt$$

$$= \frac{e^{-x}}{x} - \int_x^\infty \frac{e^{-t}}{t^2}\,dt$$

$$= \frac{e^{-x}}{x} - \frac{e^{-x}}{x^2} + 1\cdot 2\int_x^\infty \frac{e^{-t}}{t^3}\,dt$$

$$= \frac{e^{-x}}{x} - \frac{e^{-x}}{x^2} + 1\cdot 2\frac{e^{-x}}{x^3} - 1\cdot 2\cdot 3\int_x^\infty \frac{e^{-t}}{t^4}\,dt$$

. . . . . .

We may now express the result as

$$E_1(x) = \frac{e^{-x}}{x}\left(1 - \frac{1!}{x} + \frac{2!}{x^2} - \frac{3!}{x^3} + \ldots\right)$$

The series in parentheses is known as an asymptotic series. Proceeding from left to right, the first few terms decrease in absolute value. Later terms increase. The series clearly diverges, since the ratios of successive terms approach infinity as we proceed toward the right. Nevertheless it is possible to obtain useful results for large values of x. It is clear from the integrations by parts that the signs alternate and that the sign of the remainder is always opposite to that of the last term considered. It follows that the true value of the series is bounded by any two successive partial sums. If x is large, a good approximation can be obtained by terminating the series with the smallest term, using only half of this term. The result is midway between the narrowest possible upper and lower bounds. In nested form, the series becomes

$$E_1(x) = \frac{e^{-x}}{x}\left(1 - \frac{1}{x}\left(1 - \frac{2}{x}\left(1 - \frac{3}{x}\left(1 - \ldots\left(\frac{1}{2}\right)\right)\right)\right)\right)$$

The program follows:

```
10  PRINT                                  Generates blank line.
20  INPUT "X=";X                           Calls for value of x.
30  S=.5                                   Initializes S.
40  FOR J=INT(X) TO 1 STEP -1              }
50  S=1-J/X*S                              } Calculates S.
60  NEXT J                                 }
70  PRINT "E1(X)=";S/X/EXP(X)              }
80  PRINT "XEXP(X)E1(X)=";S                } Prints results.
90  GOTO 10                                Returns for new input.
```

Results follow for $x = 7$, 8, 9, and 10.

x	7	8	9	10
$xe^xE_1(x)$	.88638	.89827	.907745	.915638

These results are inferior to those given by the first program with the TI-99/4, but better than those found with the Apple IIe and the Commodore 64, and far better than those that can be found with the TRS-80. Other methods of evaluation are also possible. Convenient formulas have been obtained for many functions by fitting simple algebraic expressions to the exact results. An evaluation of this type for $E_1(x)$ is given in Prob. 3-2. An evaluation using numerical integration is given in Chapter 4, Prob. 4-21. This is more accurate than any of the evaluations given here, although it is less convenient to use.

3-3. The Error Function

The error function erf x is defined by the equation

$$\text{erf } x = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt \tag{3-20}$$

By expanding the integrand into an infinite series and integrating term by term, we find that

$$\text{erf } x = \frac{2}{\sqrt{\pi}} x \left(1 - \frac{x^2}{3 \cdot 1!} + \frac{x^4}{5 \cdot 2!} - \frac{x^6}{7 \cdot 3!} + \frac{x^8}{9 \cdot 4!} - \cdots\right) \tag{3-21}$$

In nested form, this becomes

$$\text{erf } x = \frac{2}{\sqrt{\pi}} x \left(1 - x^2 \left(\frac{1}{3} - \frac{x^2}{2}\left(\frac{1}{5} - \frac{x^2}{3}\left(\frac{1}{7} - \frac{x^2}{4}\left(\frac{1}{9} - \cdots\right)\right)\right)\right)\right) \tag{3-22}$$

The number of terms necessary for convergence is

$$n = \text{Int}(14x + 3) \tag{3-23}$$

The program follows. Line 1 is the title. Line 10 generates a blank line between successive sets of data. Line 20 calls for the input x, using an input prompting message, and prints the response on the screen. Line 30 calculates n, the required number of terms of the series, using Eq. 3-23. Lines 40 through 70 evaluate S, the sum of the nested series in Eq. 3-22. Line 80 completes the evaluation of erf x, and line 90 prints the result. Line 100 returns the execution of the program to the beginning in preparation for further input.

```
  1 REM: ERROR FUNCTION
 10 PRINT                              Generates blank line.
 20 INPUT "X=";X                       Calls for value of x.
 30 N=INT(14*X+3)                      Calculates n.
 40 S=1/(2*N-1)                        Initializes S.
 50 FOR J=N-1 TO 1 STEP -1             }
 60 S=1/(2*J-1)-X*X/J*S                }Calculates S.
 70 NEXT J                             }
 80 ERF=X/SQR(ATN(1))*S                Calculates result.
 90 PRINT "ERF(X)=";ERF                Prints result.
100 GOTO 10                            Returns for new input.
```

Accurate numerical results from reference 9 appear in the following table:

x	erf x	x	erf x
0.0	.00000 00000	2.0	.99532 22650
0.5	.52049 98778	2.5	.99959 30480
1.0	.84270 07929	3.0	.99997 79095
1.5	.96610 51465	3.5	.99999 92569

Results from the program are identical to those shown with the exception of some slight discrepancies in the last digit. For larger values of x, erf x is usually taken as 1. An alternate program is given in Prob. 3-4.

3-4. Complete Ellipitic Integrals

The complete elliptic integral of the first kind $K(k)$ is defined by the equation

$$K(k) = \int_0^{\frac{\pi}{2}} \frac{d\theta}{(1 - k^2 \sin^2 \theta)^{1/2}} \tag{3-24}$$

The best way to evaluate this integral numerically is to use the infinite product (reference 4):

$$K(k) = \frac{\pi}{2}(1 + k_1)(1 + k_2)(1 + k_3) \ldots \tag{3-25}$$

The ks are given by the recurrence relation

$$k_j = \frac{1 - \sqrt{1 - k_{j-1}^2}}{1 + \sqrt{1 - k_{j-1}^2}} \qquad k_0 = k \tag{3-26}$$

This can be simplified to either

$$k_j = \left(\frac{1 - \sqrt{1 - k_{j-1}^2}}{k_{j-1}}\right)^2 \qquad k_0 = k \tag{3-27}$$

or

$$k_j = \left(\frac{k_{j-1}}{1+\sqrt{1-k_{j-1}^2}}\right)^2 \quad k_0 = k \tag{3-28}$$

The last form is preferable because it does not break down when $k = 0$. Results converge to ten significant figures provided that $k^2 \leq .99$, and five terms are taken in the infinite product of Eq. 3-25.

The program follows. Line 1 is the title. Line 10 generates a blank line between successive sets of data. Line 20 calls for the input k, using an input prompting message, and prints the response on the screen. Lines 30 through 80 calculate the product P on the right side of Eq. 3-25. The parameter Q in the program is k_j of Eq. 3-28. Line 90 prints the result, and line 100 returns the execution of the program to the beginning in preparation for further input.

```
  1 REM: COMPLETE ELLIPTIC INTEGRAL K(K)
 10 PRINT                                  Generates blank line.
 20 INPUT "K=";K                           Calls for value of k.
 30 Q=K                                    Initializes Q.
 40 P=2*ATN(1)                             Initializes P.
 50 FOR J=1 TO 5                          \
 60 Q=(Q/(1+SQR(1-Q*Q)))^2                 } Calculates P.
 70 P=(1+Q)*P                             /
 80 NEXT J                                /
 90 PRINT "K(K)=";P                        Prints result.
100 GOTO 10                                Returns for new input.
```

Accurate numerical results from reference 1 are given in the table at the end of this section. Results from the program are identical to those shown. The integral of Eq. (3-24) diverges when $k = 1$.

The complete elliptic integral of the second kind $E(k)$ is defined by the equation

$$E(k) = \int_0^{\frac{\pi}{2}} (1 - k^2 \sin^2\theta)^{1/2} d\theta \tag{3-29}$$

This is evaluated numerically by using the expansion

$$E(k) = K(k)\left[1 - \frac{k^2}{2}\left(1 + \frac{k_1}{2} + \frac{k_1}{2}\frac{k_2}{2} + \frac{k_1}{2}\frac{k_2}{2}\frac{k_3}{2} + \ldots\right)\right] \tag{3-30}$$

The program follows. Since the value of $K(k)$ is needed as an intermediate step in the evaluation of $E(k)$, the program calculates both. When this

program is used, the earlier program is not needed. Line 1 is the title. The remaining lines are identical to those of the first program with the exception of lines 50, 60, 100, 110, 130, and 150. These lines calculate and print $E(k)$. The parameter R is the general term of the series of Eq. 3-30, and S is the partial sum.

```
 1  REM: COMPLETE ELLIPTIC INTEGRALS K(K) AND E(K)
10  PRINT                                  Generates blank line.
20  INPUT "K=";K                           Calls for value of k.
30  Q=K
40  P=2*ATN(1)                             } Initializes variables.
50  R=1
60  S=1
70  FOR J=1 TO 5
80  Q=(Q/(1+SQR(1-Q*Q)))^2
90  P=(1+Q)*P                              } Calculates P and S.
100 R=Q*R/2
110 W=R+S
120 NEXT J
130 E=P*(1-K*K*S/2)                        Calculates E(K).
140 PRINT "K(K)=";P                        } Prints results.
150 PRINT "E(K)=";E
160 GOTO 10                                Returns for new input.
```

Accurate numerical results from reference 1 appear in the following table:

k^2	$K(k)$	$E(k)$	k^2	$K(k)$	$E(k)$
0.0	1.57079 6327	1.57079 6327	0.6	1.94956 7750	1.29842 8034
.1	1.61244 1349	1.53075 7637	.7	2.07536 3135	1.24167 0567
.2	1.65962 3599	1.48903 5058	.8	2.25720 5327	1.17848 9924
.3	1.71388 9448	1.44536 3064	.9	2.57809 2113	1.10477 4733
.4	1.77751 9371	1.39939 2139	.99	3.69563 7363	1.01599 3546
.5	1.85407 4677	1.35064 3881			

Results from the program are identical to those shown. The integral of Eq. (3-29) converges to the value 1 when $k = 1$, but the expansion of Eq. (3-30) breaks down.

One further comment about the table may be helpful. The basic parameter has been taken as k^2 instead of k so the results can be checked against reference 1. This means that the input numbers are $\sqrt{.1}$, $\sqrt{.2}$, ... A few versions of BASIC allow a simple algebraic expression such as SQR(.1) to be used as input, but most do not. To generate the table, it may be advantageous to modify the beginnings of the programs to

```
20  INPUT "K^2=";L
25  K=SQR(L)
```

3-5. The Factorial Function

The factorial function $x!$ is defined by the equation*

$$x! = \int_0^\infty t^x e^{-t} dt \tag{3-31}$$

By integrating by parts k times, we find that

$$\begin{aligned} x! &= x(x-1)(x-2) \ldots (x-k+1) \int_0^\infty t^{x-k} e^{-t} dt \\ &= x(x-1)(x-2) \ldots (x-k+1)[(x-k)!] \end{aligned} \tag{3-32}$$

If x is an integer, we set $k = x$. Then this reduces to the elementary factorial. A program to evaluate the elementary factorial has been given in Sec. 1-3. The easiest way to evaluate $x!$ in the general case is to use the asymptotic formula

$$\begin{aligned} \ln x! = \left(x + \frac{1}{2}\right) \ln x - x + \frac{1}{2} \ln(2\pi) - \frac{1}{12x} \left(- \frac{1}{30x^2} \right. \\ \left. + \frac{1}{105x^4} - \frac{1}{140x^6} + \frac{1}{99x^8} - \ldots \right) \end{aligned} \tag{3-33}$$

This can be written in nested form as

$$\begin{aligned} \ln x! = \left(x + \frac{1}{2}\right) \ln x - x + \frac{1}{2} \ln (2\pi) + \frac{1}{12x} \left(1 - \frac{1}{x^2} \left(\frac{1}{30} \right. \right. \\ \left. \left. - \frac{1}{x^2} \left(\frac{1}{105} - \frac{1}{x^2} \left(\frac{1}{140} - \frac{1}{x^2} \left(\frac{1}{99} \right) \right) \right) \right) \right) \end{aligned} \tag{3-34}$$

Equation 3-34 is very good for large values of x. For $x \geq 5$, it gives results that are accurate to ten significant figures (if this is within the capability of the computer). For smaller values of x, results are obtained by using Eq. 3-32 in conjunction with 3-34. Thus, for example

$$1.3! = \frac{5.3!}{5.3 \cdot 4.3 \cdot 2.3}$$

The program follows. Line 1 is the title. Line 10 generates a blank line between successive sets of data. Line 20 calls for the input x, using a prompting message, and prints the response on the screen. In lines 30

* The Gamma function is also used. This is defined by the equation

$$\Gamma(x) = (x-1)! = \int_0^\infty x^{x-1} e^{-t} dt$$

through 80, the value of x is assigned to a parameter z, which is then tested to see whether it is equal to or greater than 5. If it is less than 5, it is incremented by 1. This operation is repeated until $z \geq 5$. At the same time the product $P = x(x + 1) \ldots$ is calculated. Lines 90 through 130 calculate S, the sum of the nested series in Eq. 3-34. The calculation proceeds from right to left, starting with 1/99 and ending with the 1 inside the first parenthesis. Line 140 is a RESTORE statement. If there is a subsequent evaluation, this causes the READ statement to start over with the first data entry. Line 150 calculates $T = \ln z!$. Line 160 calculates and prints the final result, $x!$. Line 170 is an optional line that returns the execution of the program to the beginning so further results can be obtained without entering RUN each time. (If this is not used, line 140 may be deleted.) Line 180 is the data line, which contains the constants for Eq. 3-34.

```
  1 REM: FACTORIAL FUNCTION
 10 PRINT                                  Generates blank line.
 20 INPUT "X=";X                           Calls for value of x.
 30 Z=X                                    Initializes z.
 40 P=1                                    Initializes P.
 50 IF Z>=5 THEN 90                        }
 60 Z=Z+1                                  } Adjusts value of z.
 70 P=Z*P                                  }
 80 GOTO 50                                }
 90 S = 1/99                               Initializes S.
100 FOR J=1 TO 4                           }
110 READ C                                 } Calculates S.
120 S=1/C-S/Z/Z                            }
130 NEXT J                                 }
140 RESTORE                                Restores data.
150 T=(Z+.5)*LOG(Z)-Z+                     } Calculates T.
    .5*LOG(8*ATN(1))+S/Z/12                }
160 PRINT "X!=";EXP(T)/P                   Calculates and prints result.
170 GOTO 10                                Returns for new input.
180 DATA 140,105,30,1                      Data line for constants in Eq. 3-34.
```

This program may be used for any real value of x, positive or negative, provided that x is not a negative integer, in which case $x!$ is infinite.

3-6. Bessel Functions

Bessel functions occur in many applications in science and engineering, and are discussed in books on advanced engineering mathematics. The series expansion for the Bessel function of the first kind $J_\nu(x)$ is

$$J_\nu(x) = \frac{1}{\nu!}\left(\frac{x}{2}\right)^\nu \left[1 - \frac{\left(\frac{x}{2}\right)^2}{(\nu+1)1!} + \frac{\left(\frac{x}{2}\right)^4}{(\nu+1)(\nu+2)2!} - \cdots\right] \tag{3-35}$$

where x is any real positive number and ν is any real number except a negative integer. In nested form this becomes

$$J_\nu(x) = \frac{1}{\nu!}\left(\frac{x}{2}\right)^\nu \left(1 - \frac{\left(\frac{x}{2}\right)^2}{1(\nu+1)}\left(1 - \frac{\left(\frac{x}{2}\right)^2}{2(\nu+2)}\left(1 - \cdots\right)\right)\right) \tag{3-36}$$

The number of terms necessary for convergence is

$$n = \text{Int}\,(2x + 5) \tag{3-37}$$

Most of the length of a program for $J_\nu(x)$ is taken up by the evaluation of the factorial function. The series alone can be evaluated by a simple program similar to the one written for the sine integral in Sec. 3-1. Let $G_\nu(x) = \nu! J_\nu(x)$. A program for $G_\nu(x)$ follows.

```
  1  REM: G=NU!*JNU(X)
 10  PRINT                          Generates blank line.
 20  INPUT "NU=";NU                 Calls for value of ν.
 30  INPUT "X=";X                   Calls for value of x.
 40  N=INT(2*X+5)                   Calculates n.
 50  S=1                            Initializes S.
 60  FOR J=N-1 TO 1 STEP -1
 70  S=1-X*X/4/J/(NU+J)*S           } Calculates S.
 80  NEXT J
 90  G=S*(X/2)^NU                   Calculates result.
100  PRINT "G=";G                   Prints result.
110  GOTO 10                        Returns for new input.
```

The foregoing program is adequate for the solution of most practical problems involving Bessel functions. Usually the factorial functions cancel or occur in combinations that cancel. Two typical equations involving Bessel functions are

$$J_{1/4}(x) = 0 \qquad y = \frac{J_{1/4}(x)}{J_{-3/4}(x)}$$

These can be reduced to the equivalent forms

$$G_{1/4}(x) = 0 \qquad y = 4\frac{G_{1/4}(x)}{G_{-3/4}(x)}$$

When ν is a positive integer, say $\nu = p$, a segment to evaluate $1/p!$ can easily be incorporated into the program. The program for $J_p(x)$ is

```
 1  REM: BESSEL FUNCTION JP(X)
10  PRINT                                  Generates blank line.
20  INPUT "ENTER P,X ";P,X                 Calls for values of p and x.
30  N=INT(2*X+5)                           Calculates n.
40  S=1                                    Initializes S.
50  FOR J=N-1 TO 1 STEP -1                 }
60  S=1-X*X/4/J/(P+J)*S                    }Calculates S.
70  NEXT J                                 }
80  FOR J=1 TO P                           }
90  S=S/J                                  }Divides S by p!.
100 NEXT J                                 }
110 JP=S*(X/2)^P                           Calculates Jp(x).
120 PRINT "P=";P                           }
130 PRINT "X=";X                           }Prints results.
140 PRINT "JP(X)=";JP                      }
150 GOTO 10                                Returns for new input.
```

Numerical results are given in the following table for several values of p and x. These are taken from reference 1. Results from the program are identical to those shown.

x	$J_0(x)$	$J_1(x)$	$J_2(x)$
0	1.00000 00000	0.00000 00000	0.00000 00000
2	.22389 07791	.57672 48078	.35283 40286
4	−.39714 98099	−.06604 33280	.36412 81459
6	.15064 52573	−.27668 38581	−.24287 32100
8	.17165 08071	.23463 63469	−.11299 17204
10	−.24593 57645	.04347 27462	.25463 03137

If a complete evaluation of $J_\nu(x)$ is needed for a nonintegral value of ν, this can be obtained by using the first program of this section in conjunction with the factorial program of Sec. 3-5. The two programs can easily be combined if desired, but a combined program is seldom needed.

When ν is a positive integer, say $\nu = p$, it is sometimes necessary to use the Bessel function of the second kind as well as the Bessel function of the first kind. The Bessel function of the second kind $Y_p(x)$ is given by the formula

$$Y_p(x) = \frac{1}{\pi}\left[2\left(\ln\frac{x}{2} + \gamma\right) J_p(x) - U_p(x) - V_p(x)\right] \tag{3-38}$$

where γ is Euler's constant and

$$J_p(x) = \frac{1}{p!}\left(\frac{x}{2}\right)^p \left[1 - \frac{\left(\frac{x}{2}\right)^2}{1\cdot(p+1)} + \frac{\left(\frac{x}{2}\right)^4}{1\cdot 2\cdot(p+1)(p+2)} - \cdots\right] \tag{3-39}$$

$$U_p(x) = \frac{1}{p!}\left(\frac{x}{2}\right)^p \left\{\phi(p) - [\phi(1) + \phi(p+1)]\frac{\left(\frac{x}{2}\right)^2}{1\cdot(p+1)} + \cdots\right\} \tag{3-40}$$

$$V_p(x) = (p-1)!\left(\frac{x}{2}\right)^{-p}\left[1 - \frac{1}{1\cdot(p-1)}\left(\frac{x}{2}\right)^2 + \frac{1}{1\cdot 2(p-1)(p-2)}\left(\frac{x}{2}\right)^4 + \cdots + \frac{1}{1\cdot 2\cdot 3\cdots(p-1)(p-1)!}\left(\frac{x}{2}\right)^{2p-2}\right] \tag{3-41}$$

The function ϕ is defined by the equation

$$\phi(k) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} \tag{3-42}$$

The evaluation of $Y_p(x)$ is much more complicated than the evaluation of $J_p(x)$ given earlier. The series of Eq. 3-40 for $U_p(x)$ resembles the series of Eq. 3-14 for $E_1(x)$. The coefficients are calculated in ascending order $i = 1,2,3, \ldots, n$, whereas we need them in inverse order for a nested evaluation. We again use the backward nested format of Sec. 3-2. It is convenient to calculate $J_p(x)$ at the same time, since Eq. 3-39 is the same as 3-40 except that it does not contain the ϕ functions. We rewrite Eq. 3-39 as

$$J_p(x) = \frac{1}{p!}\left(\frac{x}{2}\right)^{p+2n-2}\left(a_n + \frac{4}{x^2}\left(a_{n-1} + \frac{4}{x^2}\left(a_{n-2} + \cdots\right)\right)\right) \tag{3-43}$$

where

$$a_i = \frac{(-1)^{i-1}}{i!(p+1)(p+2)\cdots(p+i)} \tag{3-44}$$

The as are given by the recurrence formula

$$a_i = -\frac{a_{i-1}}{i(p+i)} \qquad a_1 = 1 \tag{3-45}$$

Equation 3-40 may be rewritten in the same form as 3-43. The general coefficient is $a_i g_i$, where

$$g_i = \phi(i-1) + \phi(p+i-1) \tag{3-46}$$

The function $V_p(x)$ is suitable for a nested evaluation as it stands. We use the finite series

$$V_p(x) = (p-1)!\left(\frac{x}{2}\right)^{-p}\left[1+\frac{1}{1(p-1)}\frac{x^2}{4}\left(1-\frac{1}{2(p-2)}\frac{x^2}{4}\right.\right.$$
$$\left.\left.\cdots\left(1-\frac{1}{(p-1)!}\frac{x^2}{4}\right)\right)\right] \tag{3-47}$$

The program follows. Line 1 is the title. Line 10 creates a blank line between successive sets of data. Line 20 calls for the input p,x. Lines 30 through 90 evaluate $\phi(p)$ and $p!$, and lines 100 through 220 evaluate $J_p(x)$ and $U_p(x)$. The parameter S is the partial sum of the nested series for $J_p(x)$ in Eq. 3-43, and R is the partial sum of the corresponding series for $U_p(x)$. Lines 230 through 300 evaluate $V_p(x)$. The parameter T is the partial sum of the series for $V_p(x)$ in Eq. 3-41. Line 310 evaluates $Y_p(x)$, using Eq. 3-38. Lines 320 through 350 print the results, and line 360 returns the execution to the beginning in preparation for further input.

```
  1 REM: BESSEL FUNCTIONS OF THE FIRST AND SECOND
    KINDS
 10 PRINT                                  Generates blank line.
 20 INPUT "ENTER P,X ";P,X                 Calls for values of p and x.
 30 M=1
 40 PHI=0
 50 IF P=0 THEN 100
 60 FOR I=1 TO P                           } Evaluates φ(p) and p!.
 70 M=I*M
 80 PHI=PHI+1/I
 90 NEXT I
100 N=INT(2*X+5)                           Calculates n.
110 A=1
120 S=1                                    } Initializes variables.
130 G=PHI
140 R=G
150 FOR I=1 TO N-1
160 A=-A/I/(P+I)
170 S=A+4*S/X/X                            } Calculates J_p(x) and U_p(x).
180 G=G+1/I+1/(P+I)
190 R=A*G+4*R/X/X
200 NEXT I
210 J=S*(X/2)^(P+2*N-2)/M
220 U=J*R/S
```

```
230  V=0
240  IF P=0 THEN 310
250  T=1
260  IF P=1 THEN 300
270  FOR I=1 TO P-1
280  T=1+X*X/4/I/(P-I)*T
290  NEXT I
300  V=M/P*T/(X/2)^P
310  Y=(2*J*(LOG(X/2)+.5772156649)
     -U-V)/ATN(1)/4
320  PRINT "P=";P
330  PRINT "X=";X
340  PRINT "JP(X)=";J
350  PRINT "YP(X)=";Y
360  GOTO 10
```

Calculates $v_p(x)$.

Calculates $Y_p(x)$.

Prints results.

Returns for new input.

Some numerical results from reference 1 appear in the following table for $Y_p(x)$.

x	$Y_0(x)$	$Y_1(x)$	$Y_2(x)$
2	0.51037 56726	−0.10703 24315	−0.61740 810
4	−.01694 07393	.39792 57106	.21590 359
6	−.28819 46840	−.17501 03443	.22985 790
8	.22352 14894	−.15806 04617	−.26303 660
10	.05567 11673	.24901 54242	−.00586 808

Results from the program agree with those shown. The values of $J_p(x)$ given by the program are also consistent with those shown earlier.

Problems

3-1. Given the identity

$$\int_0^\infty \frac{e^{-xt}}{t^2+1}\,dt = \sin x\ \mathrm{Ci}(x) + \cos x\left[\frac{\pi}{2} - \mathrm{Si}(x)\right]$$

write a program to evaluate the integral. The following results may be used to check the program:

x	1	2	5	10
I	.62144 96242	.39902 09886	.18814 27746	.09819 10348

3-2. The following equation from reference 1 can be used to evaluate the function $E_1(x)$ for large values of x:

$$xe^xE_1(x) = \frac{x^2 + 4.0364x + 1.15198}{x^2 + 5.03637x + 4.1916}$$

Using this equation, write a program to evaluate $E_1(x)$ and $xe^xE_1(x)$. Verify the following results, and compare them with those found in the text.

x	7	8	9	10
$xe^xE_1(x)$	.88649 1323	.89823 8205	.90775 7824	.91563 3304

3-3. The function $E_n(x)$ is defined by the equation

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n}\,dt \qquad n = 0,1,2,3,\ldots$$

Show that $E_0(x) = e^{-x}/x$ and that $E_1(x)$ is consistent with Eq. 3-12. Also show that $E_n(x)$ satisfies the recurrence formula

$$E_n(x) = \frac{1}{n-1}\left[e^{-x} - xE_{n-1}(x)\right] \qquad n \geq 2$$

Write a program to evaluate $E_n(x)$ for $n \geq 1$. The program can be checked against numerical results given on pages 245–248 of reference 1.

3-4. By rewriting Eq. 3-20 as

$$\text{erf } x = \frac{2}{\sqrt{\pi}}\, e^{-x^2} \int_0^x e^{x^2 - t^2} dt$$

obtain the series expansion

$$\text{erf } x = \frac{2x}{\sqrt{\pi}}\, e^{-x^2}\left[1 + \frac{(2x^2)}{1\cdot 3} + \frac{(2x^2)^2}{1\cdot 3\cdot 5} + \ldots\right]$$

Write a program based on this series. Check the numerical results given in Sec. 3-3 by using both the program of that section and the new program. Use the same computer for both programs.

With most computers, a program based on the present algorithm gives much more accurate results than the program of Sec. 3-3. Roundoff errors are smaller because all terms are positive. (The difference may not be apparent if a highly accurate computer such as the TI-99/4 is used.)

3-5. Dawson's integral $F(x)$ is defined by the equation

$$F(x) = e^{-x^2} \int_0^\infty e^{t^2} dt$$

Obtain the series expansion

$$F(x) = xe^{-x^2}\left(1 + \frac{x^2}{3\cdot 1!} + \frac{x^4}{5\cdot 2!} + \frac{x^6}{7\cdot 3!} + \ldots\right)$$

and write a program to evaluate $F(x)$. The following results may be used to check the program:

x	1	2	5	10
$F(x)$	.53807 95069	.30134 03889	.10213 40744	.05025 38471

3-6. The complementary error function erfc x occurs in some applications. This is defined by the equation

$$\text{erfc } x = 1 - \text{erf } x$$

For large values of x, it is not possible to evaluate erfc x by using the program of Sec. 3-3 for erf x, because the value of erf x is practically indistinguishable from 1. (The program of Prob. 3-4 is better, but it is still not satisfactory for large values of x.) Write a program

to evaluate erfc x for large values of x by using the asymptotic expansion

$$\operatorname{erfc} x = \frac{e^{-x^2}}{\sqrt{\pi}\, x}\left[1 - \frac{1}{(2x^2)} + \frac{1\cdot 3}{(2x^2)^2} - \frac{1\cdot 3\cdot 5}{(2x^2)^3} + \cdots\right]$$

The following results may be used to check the program:

x	3.5	5	7	10
$xe^{x^2}\operatorname{erfc} x$	.5435276	.5535232	.5586004	.5614099

3-7. Show that

a. $$\int_0^b \frac{dx}{[(a^2-x^2)(b^2-x^2)]^{1/2}} = \frac{1}{a}K\left(\frac{b}{a}\right) \qquad a > b \geq 0$$

b. $$\int_0^b \frac{dx}{[(a^2+x^2)(b^2-x^2)]^{1/2}} = \frac{1}{\sqrt{a^2+b^2}}K\left(\frac{b}{\sqrt{a^2+b^2}}\right) \qquad a \geq b \geq 0$$

c. $$\int_0^\infty \frac{dx}{[(a^2+x^2)(b^2+x^2)]^{1/2}} = \frac{1}{a}K\left(\frac{\sqrt{a^2-b^2}}{a}\right) \qquad a \geq b > 0$$

d. $$\int_b^a \frac{dx}{[(a^2-x^2)(x^2-b^2)]^{1/2}} = \frac{1}{a}K\left(\frac{\sqrt{a^2-b^2}}{a}\right) \qquad a \geq b > 0$$

Evaluate the integrals numerically for $a = .5$ and $b = .4$.

Ans. a. 3.99060 5555 b. 2.76554 5985 c,d. 3.50150 7606

3-8. Revise the program of Sec. 3-4 for the elliptic integrals $K(k)$ and $E(k)$, using a nested format for Eq. 3-30.

3-9. Write a program to evaluate the Beta function $\mathrm{B}(p,q)$. The equation is

$$\mathrm{B}(p,q) = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} = \frac{(p+q-1)!}{(p-1)!(q-1)!}$$

3-10. The Struve function $H_\nu(x)$ is related to the Bessel function $J_\nu(x)$. For $\nu = p =$ a positive integer or zero, the equation is

$$H_p(x) = \frac{\dfrac{2}{\pi}\left(\dfrac{x}{2}\right)^{p+1}}{\dfrac{1}{2}\dfrac{3}{2}\dfrac{5}{2}\cdots\left(p+\dfrac{1}{2}\right)}\left[1 - \frac{\left(\dfrac{x}{2}\right)^2}{\dfrac{3}{2}\left(p+\dfrac{3}{2}\right)} + \frac{\left(\dfrac{x}{2}\right)^4}{\dfrac{3}{2}\dfrac{5}{2}\left(p+\dfrac{3}{2}\right)\left(p+\dfrac{5}{2}\right)} - \cdots\right]$$

Write a program to evaluate the Struve function $H_p(x)$. The following numerical values of $H_p(x)$ may be used to check the program:

p \ x	1	2	3	5
0	.56865 66	.79085 88	.57430 61	−.18521 68
1	.19845 73	.64676 37	1.02010 96	.80781 19

4

Numerical Integration

The problem of evaluating a definite integral occurs frequently in applications. The best procedure is to find an exact analytical solution. However, this is often impossible. A second method is to expand the integrand into an infinite series and integrate term by term. We have used this procedure in Chapter 3. A third method is to calculate the value of the integrand at a number of discrete points and replace the integral by a weighted sum, that is, approximate the value of the integral by an equation of the type

$$\int_a^b f(x)\,dx = (b-a)\sum w_j f(x_j) \tag{4-1}$$

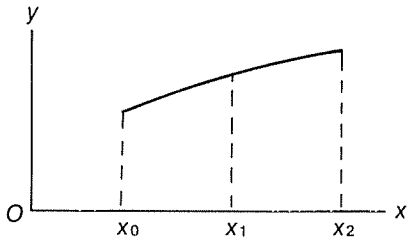
where w_j is an appropriate weighting factor. In this chapter we shall consider several methods of this type. It will be assumed throughout the first three sections that the integrand is continuous and that the interval is finite.

4-1. Simpson's Rule

One very simple and widely used formula for numerical integration is

$$I = \int_{x_0}^{x_2} y dx = \frac{h}{3}(y_0 + 4y_1 + y_2) \tag{4-2}$$

FIG. 4-1



Points 0 and 2 are the end points and point 1 is the midpoint of the interval, as shown in Fig. 4-1. h is the length of one subinterval. Equation 4-2 can be derived by passing a parabola through the three points. This formula is exact if $y = f(x)$ is a polynomial of degree ≤ 3. In general it is an approximation. Better accuracy is obtained by breaking the interval into an even number n of subintervals, each of length h, as shown in Fig. 4-2, and applying Eq. 4-2 to successive pairs of subintervals. This leads to

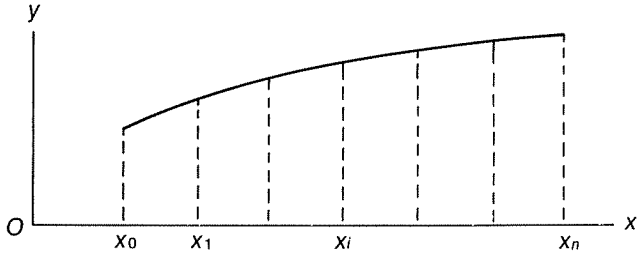
$$I = \int_{x_0}^{x_n} y dx = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots + y_n) \tag{4-3}$$

or

$$I = \frac{h}{3}\left[(y_0 + y_n) + \sum_{j=1}^{n-1} w_j y_j\right] \qquad w_j = 2 \text{ or } 4 \tag{4-4}$$

Equations 4-2, 4-3, and 4-4 are known as Simpson's rule.

FIG. 4-2



The program follows. Line 1 is the title. Line 10 reads the values of x_0 and x_n from the data line, and line 20 generates a blank line between successive sets of output. Line 30 calls for the value of n. (The reason for using an INPUT statement for n is that it is customary in using Simpson's rule to make several approximations with different values of n. The present format makes it possible to do so without editing the program each time.) Line 40 calculates h, the length of the subinterval. Lines 50, 60, and 70 calculate the value of y_0 and assign it to the parameter S, which is the partial sum of the series in Eq. 4-3 or 4-4. Lines 80, 90, and 100 calculate y_n and add it to S. Lines 110 through 170 calculate the weighted values of y_1 through y_{n-1} and add them to S. Line 180 calculates and prints the value of I. Line 190 returns the execution of the program to the beginning in preparation for the next approximation with a new value of n. Line 200 is the subroutine for the integral

$$I = \int_0^{\pi/2} x^2 \cos x \, dx \tag{4-5}$$

Line 210 is the RETURN statement and line 220 is the data line, which contains the values of x_0 and x_n. Execution is started by entering RUN. Varying levels of approximation are obtained by entering a new value of n each time the prompting message of line 30 appears on the screen. After satisfactory convergence has been obtained, the execution is terminated by pressing the BREAK key.

```
  1  REM: SIMPSON'S RULE
 10  READ X0,XN                 Reads values of x0 and xn.
 20  PRINT                      Generates blank line.
 30  INPUT "N=";N               Calls for value of n.
 40  H=(XN-X0)/N                Calculates length of subinterval.
 50  X=X0                       } Evaluates contribution of
 60  GOSUB 200                  } left end point to integral.
 70  S=Y                        }
 80  X=XN                       } Evaluates contribution
 90  GOSUB 200                  } of right end point.
100  S=S+Y                      }
110  W=4                        }
120  FOR J=1 TO N-1             }
130  X=XO+J*H                   } Evaluates contributions
140  GOSUB 200                  } of intermediate points.
150  S=S+W*Y                    }
160  W=6-W                      }
170  NEXT J                     }
180  PRINT "I=";H*S/3           Calculates and prints result.
```

```
190  GOTO 20                    Returns for next value of n.
200  Y=X*X*COS(X)               Subroutine.
210  RETURN                     RETURN statement.
220  DATA 0,1.570796327         Data line for end points.
```

The exact value of the integral of Eq. 4-5 is

$$I = \frac{\pi^2}{4} - 2 = .4674011003$$

Numerical results from the program are as follows:

n	2	4	8	16	32
I	.4568	.46689	.467371	.4673993	.46740099

With the program still in the computer, we can easily proceed to evaluate other integrals. To terminate the foregoing execution, we press the BREAK key. New lines 200 and 220 are then edited into the program. Next, we enter the RUN command to start execution, and proceed as in the first example. Consider the integral

$$I = \int_0^1 \frac{\ln(1+x)}{1+x^2}\,dx \tag{4-6}$$

which has the exact value

$$I = \frac{\pi}{8}\ln 2 = .2721982613$$

Lines 200 and 220 now become

```
200  Y=LOG(1+X)/(1 + X*X)
220  DATA 0,1
```

Numerical results from the program are given in the following table.

n	2	4	8	16	32
I	.2470	.27233	.272206	.2721987	.27219829

In the foregoing two examples, the accuracy of the results could be checked by comparing them with exact solutions. This is not possible in a practical problem; if an exact solution is known, there is no point in a numerical evaluation. The accuracy of an evaluation is inferred by comparing two results with different values of n. The results may be assumed to be correct through the point through which the digits coincide.

Thus it follows from the last two results that the value of the integral of Eq. 4-6 is .272198.

Several difficulties sometimes occur in using Simpson's rule, and we shall now consider the most common ones. Even when the integrand is continuous, it may have indeterminacies at one or more points, most often at an end. Consider, for example

$$I = \int_0^2 \frac{\sin x}{x}\, dx \tag{4-7}$$

At the point $x = 0$, the integrand has the form 0/0. It is easily found by l'Hospital's rule that its value at this point is 1, but this evaluation cannot be made by the computer; it must be inserted. We therefore revise line 60 as well as 200 and 220. These lines are now edited to

```
 60  Y=1
200  Y=SIN(X)/X
220  DATA 0,2
```

The accurate value of the integral is 1.60541 2977 (see Sec. 3-1). The program gives the following approximations:

n	2	4	8	16	32
I	1.6069	1.60550	1.605418	1.6054133	1.60541300

Some care must be taken in using Simpson's rule when the integrand is very large in a small part of the interval and negligible elsewhere, as in the case of a rapidly varying exponential. In this case, a uniform spacing of points over the entire interval cannot be expected to give good results; the points must be spaced more closely in the region in which the integrand is large.

The basic Simpson's rule is unsuitable for a function that contains an oscillatory component. In this case the points must be spaced closely enough so that the subinterval h is a small fraction not merely of the interval, but of the wave length. If the interval contains a number of waves, the required number of points is so great that the method becomes impractical. A modification of Simpson's rule has been developed by Filon for the integrals

$$\int_{x_0}^{x_n} f(x)\sin x\, dx \qquad \int_{x_0}^{x_n} f(x)\cos x\, dx \tag{4-8}$$

where $f(x)$ is a function of the type that could be integrated directly by the ordinary Simpson's rule. The method is described in reference 12. The formulas can also be found in reference 1, pages 890-891.

4-2. Gauss Integration

The basic principle of numerical integration is to replace an integral by a weighted sum. Simpson's rule consists of repeated applications of the parabolic formula, Eq. 4-2. There are more efficient methods of utilizing data from a large number of points. One procedure is to again use equally spaced points, but fit a higher-order polynomial to all of the points. Formulas obtained in this way are known as Newton-Cotes formulas. However, it is more efficient to drop the requirement of equally spaced points, which is entirely arbitrary. This doubles the number of adjustable parameters in Eq. 4-1, since the x_js may now be chosen for optimum computational efficiency as well as the $w_j s$. In this way it is possible to fit a polynomial of order $2n - 1$ to n points. This procedure is known as Gauss integration. The derivation, which can be found in Sec. 1 of the appendix, is rather lengthy. However, the results are simple and easy to use. The basic formula for Gauss integration is

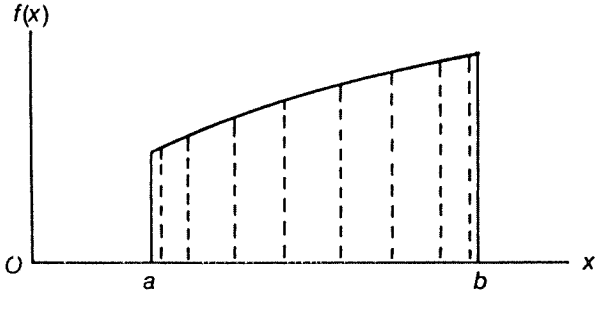
$$I = \int_a^b f(x)dx = \frac{b-a}{2} \sum_{j=1}^{n} w_j f(x_j) \tag{4-9a}$$

where w_j is a weighting factor and

$$x_j = \frac{b+a}{2} + \frac{b-a}{2} \xi_j \tag{4-9b}$$

The parameter n is the number of points (not the number of subintervals). The points are symmetrically located, as shown in Fig. 4-3 for $n = 8$. The end points are not included in the set of points at which the function is to be evaluated. (For this reason, the limits in Eq. 4-9a are denoted as a and b instead of x_0 and x_n.) It is convenient to choose an even number of points and arrange them in symmetrically located pairs. Then the equations can be rewritten in the form

FIG. 4-3



$$h = \frac{b-a}{2} \qquad x_m = \frac{b+a}{2} \tag{4-10a,b}$$

$$I = h \sum_{j=1}^{n/2} w_j \left[f\left(x_m + h\,\xi_j\right) + f\left(x_m - h\,\xi_j\right) \right] \tag{4-10c}$$

The parameter h is the half-length of the interval, and x_m is the abscissa of the midpoint.

The parameters ξ_j and w_j are found from the following table, which is abstracted from reference 1. (The same results can be obtained from a program in Sec. 1 of the appendix.) Extensive formulas and tables for various forms of Gauss integration can be found in references 1 and 11.

n	ξ_j	w_j
2	0.57735 02692	1.00000 00000
4	.33998 10436	.65214 51549
	.86113 63116	.34785 48451
6	.23861 91861	.46791 39346
	.66120 93865	.36076 15730
	.93246 95142	.17132 44924
8	0.18343 46425	0.36268 37834
	.52553 24099	.31370 66459
	.79666 64774	.22238 10345
	.96028 98565	.10122 85363

The program follows. Line 1 is the title. Line 10 reads the values of a and b from the data line. Line 20 assigns the starting value 0 to the parameter S, which is the partial sum of the series in Eq. 4-10c. Line 30 calculates h, the half length of the interval, and line 40 calculates x_m, the abscissa of the midpoint. Lines 50 through 110 constitute a FOR-NEXT loop that evaluates S. Line 120 calculates I and prints the result. Line 130 is an END statement that prevents the execution of the program from running into the subroutine, which appears in lines 140 and 150. Line 140 is the equation for the integrand $y = f(x)$. We have chosen Eq. 4-5, which is

$$I = \int_0^{\pi/2} x^2 \cos x \, dx$$

Line 150 adds the weighted value of y to the partial sum S. Line 160 is the RETURN statement. Line 170 is the data line for the limits a and b. Line 180 is the data line for the ξ_js and w_js. The data in lines 170 and 180 are given to ten significant figures. These numbers may be rounded off if desired.

```
 1  REM:GAUSS INTEGRATION-8 POINTS
10  READ A,B                    Reads values of a and b.
20  S=0                         Initializes S.
30  H=(B-A)/2                   Calculates half-length of interval.
40  XM=(B+A)/2                  Calculates abscissa of midpoint.
```

```
 50  FOR J=1 TO 4                    ┐
 60  READ XI,W                       │
 70  X=XM+H*XI                       │
 80  GOSUB 140                       ├ Evaluates S.
 90  X=XM-H*XI                       │
100  GOSUB 140                       │
110  NEXT J                          ┘
120  PRINT "I=";H*S                  Calculates and prints result.
130  END                             END statement.
140  Y=X*X*COS(X)                    ┐
150  S=S+W*Y                         ┘ Subroutine.
160  RETURN                          RETURN statement.
170  DATA 0,1.570796327              Data line for end points.
180  DATA .1834346425, .3626837834,  ┐
          .5255324099, .3137066459,  │ Data lines for Gauss
          .7966664774, .2223810345,  │ coefficients.
          .9602898565, .1012285363   ┘
```

The data line 180 requires two comments. The data entries are shown in block form to make them easy to read; actually they run continuously. Also, for the Commodore 64, which has a maximum line length of 80 characters, line 180 must be broken into two lines.

For the integral of Eq. 4-5, the program gives the result $I =$ .4674011003, which is correct to ten significant figures.

We again consider the integral of Eq. 4-6

$$I = \int_0^1 \frac{\ln (1 + x)}{1 + x^2}\, dx$$

which has been evaluated previously by Simpson's rule. Lines 140 and 170 of the program are edited to read

```
140  Y=LOG(I+X)/(1+X*X)
170  DATA 0,1
```

The numerical result is $I =$ .2721982613, which is also correct to ten significant figures.

The two foregoing examples may tend to give a false sense of confidence in the accuracy of Gauss integration. This method sometimes gives rather poor results, even for some very simple integrals. Consider

$$I = \int_0^1 \sqrt{x}\, dx \tag{4-11}$$

Lines 140 and 170 now read

```
140 Y=SQR(X)
170 DATA 0, 1
```

The correct result is 0.66666...., but the program gives 0.66683... . The present result is much poorer than the two preceding ones. This situation will be clarified to some extent later in this section. However, there is no practical way to determine the accuracy of Gauss integration in advance. It is always necessary to carry out at least two evaluations of an integral using different values of n. The results may be assumed to be correct up to the point through which the digits coincide.

The restrictions noted for Simpson's rule at the end of Sec. 4-1 also apply to Gauss integration. However, indeterminacies are less likely to occur with Gauss integration than with Simpson's rule, because the former method does not use the end points.

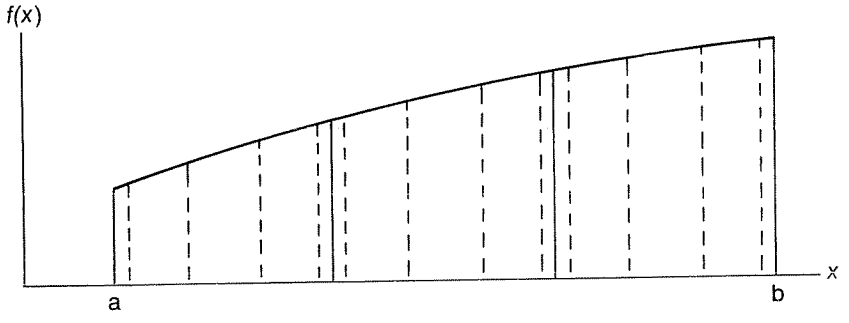
For a given level of accuracy, Gauss integration is much faster and more efficient than Simpson's rule. Also, indeterminacies are rare. However, these advantages are largely offset by the difficulty of verifying the results; it is inconvenient to use several programs for each problem. It is often advantageous to use a modified form of Gauss integration that is shown schematically in Fig. 4-4. In this method the interval is broken into m equal panels. The Gauss method with n points is applied to each panel, and the results for the m panels are added. Fig. 4-4 shows the scheme for $m = 3$, $n = 4$. The method is less accurate than a direct application of the Gauss method with mn points, but it reduces the tabular data to n entries instead of mn. Different levels of accuracy are obtained with a single program by varying m while keeping n fixed. The equations are

$$h = \frac{b-a}{2m} \tag{4-12a}$$

$$x_k = a + kh \qquad k = 1,3,5, \ldots, 2m-1 \tag{4-12b}$$

$$I = h \sum_{k=1,3,5}^{2m-1} \sum_{j=1}^{n/2} w_j \left[f(x_k + h\xi_j) + f(x_k - h\xi_j) \right] \tag{4-12c}$$

FIG. 4-4



A program follows for modified Gauss integration with $n = 8$ and any value of m. The content of the program is similar to that of the basic Gauss program, but the organization is different. Line 1 is the title. Line 10 reads the limits a and b from the data. Instead of being read each time they are used, the ξs and ws are read only once, in lines 20 through 40, and their values are assigned to subscripted variables. Line 50 prints a blank space between successive sets of output and line 60 calls for the value of m. Line 70 calculates h, and line 80 assigns the initial value 0 to the parameter S, which is the partial sum of the series in Eq. 4-12c. Lines 90 through 170 constitute a nested FOR-NEXT loop. The inner loop, which runs from line 110 through line 160, evaluates the contribution of one panel to the integral. The outer loop moves the execution from panel to panel and adds the results. Line 180 prints the final result, and line 190 returns the execution to the beginning in preparation for the next approximation. Lines 200 and 210 are the subroutine. Line 200 calculates the value of the integrand $y = f(x)$, and line 210 adds the weighted result to the partial sum S. Line 220 is a RETURN statement. Line 230 is the data line for the limits a and b, and line 240 is the data line for the ξs and ws. Varying levels of approximation are obtained by entering different values of m in response to the INPUT statement. With $m = 1$ this program gives exactly the same results as the earlier program. It follows that the integral of Eq. 4-5 will not provide an adequate test case for this program, because a highly accurate result would be obtained with $m = 1$ and the operation of the outer loop would not be checked. We use the integral

$$I = \int_0^1 \frac{\arcsin x}{x}\, dx \tag{4-13}$$

```
 1  REM: MODIFIED GAUSS INTEGRATION
10  READ A,B                          Reads values of a and b.
20  FOR J=1 TO 4                      }
30  READ XI(J),W(J)                   } Reads Gauss coefficients.
40  NEXT J                            }
50  PRINT                             Generates blank line.
60  INPUT "M=";M                      Calls for value of m.
70  H=(B-A)/M/2                       Calculates half length of one
                                      panel.
80  S=0                               Initializes S.
90  FOR K=1 TO 2*M STEP 2             }
100 XK=A+K*H                          }
110 FOR J=1 TO 4                      }
120 X=XK+H*XI(J)                      } Evaluates contribution of one panel.
130 GOSUB 200                         } Covers entire interval.
140 X=XK-H*XI(J)                      }
150 GOSUB 200                         }
160 NEXT J                            }
170 NEXT K                            }
```

```
180  PRINT "I=";H*S                          Calculates and prints result.
190  GOTO 50                                 Returns for next value of m.
200  Y=ATN(X/SQR(1-X*X))/X                   } Subroutine.
210  S=S+Y*W(J)                              }
220  RETURN                                  RETURN statement.
230  DATA 0,1                                Data line for end points.
240  DATA .1834346425, .3626837834,          }
          .5255324099, .3137066459,          } Data lines for
          .7966664774, .2223810345,          } Gauss coefficients.
          .9602898565, .1012285363           }
```

The integral of Eq. 4-13 has the exact value

$$I = \frac{\pi}{2} \ln 2 = 1.088793045$$

The program gives the following results:

m	1	2	3	4	5
I	1.08856	1.08871	1.088747	1.088763	1.088772

With the program still in the computer, we can easily proceed to evaluate other integrals. To terminate the present evaluation, we press the BREAK key. New lines 200 and 230 are then edited into the program. With any value of n, we verify the results found with the first Gauss program for the integrals in Eqs. 4-5 and 4-6.

We now return to a topic mentioned earlier: the uneven accuracy of results obtained with Gauss integration. Some integrals like those of Eqs. 4-5 and 4-6 show excellent convergence, some like that of Eq. 4-13 show fair to good convergence, and some like that of Eq. 4-11 show poor convergence. With the modified Gauss method it is possible to ignore the problem and rely upon "brute force"; almost any proper integral can be evaluated to a reasonably high degree of accuracy by using very large values of m. Nevertheless, the question of convergence is of theoretical interest. Also, the running time is shorter if an efficient process is used.

The Gauss method is based on the approximation of the exact function by a polynomial. The method works well if the function can be represented throughout the interval by a Taylor series which is substantially convergent for terms of order no higher than those contained in the approximating polynomial. The best results are obtained if the integrand is represented by a rapidly convergent Taylor series. The integrands of Eqs. 4-11 and 4-13 do not satisfy this condition. This type of difficulty can often be corrected by a simple substitution. By writing x^2 for x in Eq. 4-11, we obtain the alternate form

$$I = \int_0^1 2\,x^2\,dx$$

For any values of n and m, Gauss integration now gives the exact result $I = \frac{2}{3}$.

The convergence of the Gauss process for the integral of Eq. 4-13 can be enhanced by getting rid of the arc sine term, which has a slowly convergent Taylor expansion. At the same time we must be careful not to introduce a singularity. The best procedure is to write sin x for x. Then the integral becomes

$$I = \int_0^{\pi/2} \frac{x}{\tan x}\, dx$$

We use the modified Gauss program. The edited lines 200 and 230 are

```
200  Y=X/TAN(X)
230  DATA 0,1.570796327
```

With $m = 1$ we find that $I = 1.088793045$, which is correct to ten significant figures.

A similar example is provided by the integral

$$I = \int_0^{\pi/2} \sin x \ln \sin x\, dx \tag{4-14}$$

which has the exact value

$$I = \ln 2 - 1 = -.3068528194$$

We evaluate the integral as it stands by using the modified Gauss program. Lines 200 and 230 become

```
200  Y=SIN(X)*LOG(SIN(X))
230  DATA 0,1.570796327
```

The results are

m	1	2	3	4	5
$-I$	.306973	.306883	.306866	.306860	.3068576

The results are adequate, but there is room for improvement. In this case the convergence is retarded by the presence of the ln sin x factor in the integrand, since the interval contains the origin. The best way to remove this factor is to integrate by parts. Then we find that

$$I = \left[(1 - \cos x) \ln \sin x\right]_0^{\pi/2} - \int_0^{\pi/2} (1 - \cos x)\cot x\, dx$$

We find by elementary calculus that the expression in brackets is equal to zero at both limits. It follows that

$$I = -\int_0^{\pi/2} \frac{1-\cos x}{\tan x}\,dx$$

Lines 200 and 230 of the modified Gauss program now become

```
200 Y=(COS(X)-1)/TAN(X)
230 DATA 0,1.570796327
```

With $m = 1$ we obtain the result $I = -.3068528194$, which is correct to ten significant figures.

4-3. Romberg Integration

We have considered Simpson's rule and Gauss integration. Another widely used method of evaluating definite integrals numerically is Romberg integration. Romberg integration consists of an extrapolation process that starts with the trapezoidal rule. This is

$$\int_a^b f(x)dx = \frac{l}{2n}\left[f(a) + 2f\left(a+\frac{l}{n}\right) + 2f\left(a+\frac{2l}{n}\right) + \ldots + f(b)\right] \quad (4\text{-}15)$$

where $l = b - a$, the length of the interval, and n is the number of equal subintervals. We consider the sequence of approximations obtained by setting $n = 1, 2, 4, 8, \ldots, 2^k, \ldots$. The first few values of I_k are

$$I_0 = \frac{l}{2}[f(a) + f(b)] \quad (4\text{-}16)$$

$$I_1 = \frac{l}{4}\left[f(a) + 2f\left(a+\frac{l}{2}\right) + f(b)\right]$$

$$I_2 = \frac{l}{8}\left[f(a) + 2f\left(a+\frac{l}{4}\right) + 2f\left(a+\frac{l}{2}\right) + 2f\left(a+\frac{3l}{4}\right) + f(b)\right]$$

It is not necessary to calculate values of f for all the points in each approximation. Only the odd points in each approximation are new. Thus we write

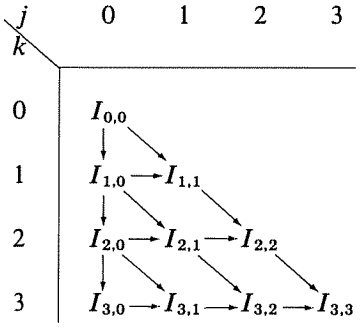
$$I_1 = \frac{I_0}{2} + \frac{l}{2}f\left(a+\frac{l}{2}\right)$$

$$I_2 = \frac{I_1}{2} + \frac{l}{4}\left[f\left(a + \frac{l}{4}\right) + f\left(a + \frac{3l}{4}\right)\right]$$

In general

$$I_k = \frac{I_{k-1}}{2} + \frac{l}{2^k} \sum_{r=1,3,5}^{2^k-1} f\left(a + \frac{rl}{2^k}\right) \tag{4-17}$$

The Romberg method consists of developing an array of approximations $I_{k,j}$ as shown in the following table:



The elements in the first column are the results I_k found from the trapezoidal rule. We now denote these as $I_{k,0}$. The remaining elements are found by repeated applications of the recurrence formula

$$I_{k,j} = \frac{4^j I_{k,j-1} - I_{k-1,j-1}}{4^j - 1} \tag{4-18}$$

The results on the diagonal converge toward the exact value. The theory of Romberg integration can be found in references 2 and 10.

The program follows. Line 1 is the title. Line 10 reads the values of x_0 and x_n, line 20 calculates the length of the interval, and line 30 assigns the starting value $k = 0$. Lines 40 through 100 evaluate and print $I(0) = I(0,0)$, using Eq. 4-16. Line 110 is a dummy input statement. This was introduced in Sec. 2-1. It temporarily stops the execution until the operator decides whether to proceed to the next higher approximation. If the operator decides to continue, he or she presses the ENTER key. Lines 120 through 200 calculate and print the next element in the column $j = 0$, using Eq. 4-17. Lines 210 through 240 calculate and print the remaining elements of the same row, using Eq. 4-18. Line 250 terminates the row in the display or printout by breaking the sequence of semicolons generated by line 230. Line 260 sends the execution back in preparation for the next cycle. Line 270 is the subroutine for the integral of Eq. 4-5, which is

$$I = \int_0^{\pi/2} x^2 \cos x \, dx$$

Line 280 is the RETURN statement, and line 290 is the data line. Lines 270 and 290 are ordinarily filled in by the user before running the program; line 270 contains the equation for y, and line 290 contains the limits a and b. After satisfactory convergence is obtained, the execution is terminated by pressing the BREAK key. Further integrals can be evaluated by editing new data into lines 270 and 290. Each new evaluation is started by entering RUN.

```
  1 REM: ROMBERG INTEGRATION--DOUBLE SUBSCRIPTS
 10 READ X0,XN                          Reads x0 and xn.
 20 L=XN-X0                             Calculates length of interval.
 30 K=0                                 Initializes k.
 40 X=X0                              ⎫
 50 GOSUB 270                         ⎪
 60 I(0,0)=L*Y/2                      ⎬ Generates first
 70 X=XN                              ⎪ element of table.
 80 GOSUB 270                         ⎪
 90 I(0,0)=I(0,0)+L*Y/2               ⎪
100 PRINT 0;I(0,0)                    ⎭
110 INPUT Q$                           Interrupts execution.
120 K=K+1                             ⎫
130 N=2^K                             ⎪
140 I(K,0)=I(K-1,0)/2                 ⎪
150 FOR R=1 TO N STEP 2               ⎬ Generates
160 X=X0+L*R/N                        ⎪ element
170 GOSUB 270                         ⎪ of column 0.
180 I(K,0)=I(K,0)+L*Y/N               ⎪
190 NEXT R                            ⎪
200 PRINT K;I(K,0);                   ⎭
210 FOR J=1 TO K                      ⎫
220 I(K,J)=(4^J*I(K,J-1)              ⎬ Generates remaining
    -I(K-1,J-1))/(4^J-1)              ⎪ elements of same row.
230 PRINT I(K,J);                     ⎪
240 NEXT J                            ⎭
250 PRINT                              Terminates row.
260 GOTO 110                           Returns for next cycle.
270 Y=X*X*COS(X)                       Subroutine.
280 RETURN                             RETURN statement.
290 DATA 0,1.570796327                 Data line for end points.
```

A DIMension statement has not been included because this method is almost never used with values of k greater than 10. However, a DIMension statement can easily be inserted at the beginning of the program if desired.

The Romberg table for the integral of Eq. 4-5 follows:

k \ j	0	1	2	3	4
0	.00000				
1	.34257	.45676 56			
2	.43581	.46689 03	.46756 523		
3	.45948	.4673713	.46740 333	.46740 0757	
4	.46542	.46739 93	.46740 113	.46740 1099	.46740 11004

The numbers in the first column are the results of the trapezoidal rule. The numbers in the second column correspond to Simpson's rule. This can be deduced from Eq. 4-18 and verified by referring back to Sec. 4-1. The results of interest are the numbers on the diagonal. These converge toward the exact result more rapidly than the Simpson results and much more rapidly than the trapezoidal results. The last result is correct to ten significant figures, with the exception of a slight discrepancy in the last digit.

It is proved in reference 2 that the Romberg process converges toward the exact result. However, it does not always converge rapidly. The integrals of Sec. 4-2 which showed poor convergence with Gauss integration converge even more poorly with Romberg integration. Some other examples of slow convergence are given in reference 2. The convergence of the Romberg process is sometimes slower than that of Simpson's rule or even the trapezoidal rule. The safest procedure in using Romberg integration is to print the entire table. (It is best to transcribe it onto paper in the format shown, because the screen and the print format of the average microcomputer are not wide enough to display complete rows horizontally.) The convergence of the various processes can then be examined, and the most satisfactory result can be chosen.

The program using variables with double subscripts is straightforward and easy to write, but it uses far more data memory than necessary. For a problem in which each member of an array is related to all the other elements, as in the solution of a set of simultaneous equations, the use of variables with double subscripts is essential. However, for a solution based on a recurrence formula, it is seldom necessary. We observed in Sec. 1-5 that an analysis based on a recurrence formula with single subscripts could be carried out without using subscripted variables in the program. Similarly, an analysis based on a recurrence formula with double subscripts can be carried out by using only single subscripts in the program. A program of this type is more difficult to write than the first program of this section, and the "brute force" approach of the first program is often preferred. With modern computers, this approach is usually feasible; there usually are computers available with ample capacity to handle most problems. However, it sometimes happens that, with routine programming,

a problem exceeds the capacity of the available computer, and a more efficient programming technique must be found. It seems desirable to consider an example of this type.*

To develop a more efficient program for Romberg integration, we begin by referring back to the diagram near the beginning of this section. We need elements from only two rows at any one time; each element of row k is obtained by using prior elements of rows $k-1$ and k. This point is shown a little more fully in the following diagram:

$$\begin{array}{cccc} I_{k-1,j-2} & I_{k-1,j-1} & I_{k-1,j} & I_{k-1,j+1} \\ I_{k,j-2} & I_{k,j-1} & I_{k,j} & I_{k,j+1} \end{array}$$

Suppose that we are in the process of calculating $I_{k,j}$. The only terms that we need to carry out the evaluation and continue indefinitely are shown in boldface type. We can proceed a step further and compress the two rows into a single row, overwriting the elements of the upper row with those of the lower row as we proceed. Two elements must be considered separately. Since $I_{k-1,j-1}$ is used at the same time as $I_{k,j-1}$ in the evaluation of $I_{k,j}$, we give it the temporary lable T. Also, to prevent the value of $I_{k-1,j}$ from being lost when $I_{k,j}$ is calculated, we give it the temporary label U. Using the single subscript j, we now have the equations

$$U = I_j$$

$$I_j = \frac{4^j I_{j-1} - T}{4^j - 1}$$

$$T = U$$

which are lines 230, 240, and 250 of the following program. These equations do not provide a starting value of T. We have to assign the old value of I_0 to T at the beginning of each cycle. This is done in line 120 of the following program.

```
  1  REM: ROMBERG INTEGRATION—SINGLE SUBSCRIPTS
 10  READ X0,XN                    Reads values of x0 and xn.
 20  L=XN-X0                       Calculates length of interval.
 30  K=0                           Initializes k.
 40  X=X0                          ⎫
 50  GOSUB 300                     ⎪
 60  I(0)=L*Y/2                    ⎪ Generates first
 70  X=XN                          ⎪ element of table.
 80  GOSUB 300                     ⎪
 90  I(0)=I(0)+L*Y/2               ⎪
100  PRINT 0;I(0)                  ⎭
```

* The more efficient method that follows is somewhat similar to the one used for Romberg integration with a programmable calculator in reference 8.

```
110  INPUT Q$                                  Interrupts execution.
120  T=I(0)                                    Assigns temporary label.
130  K=K+1                                     }
140  N=2^K                                     }
150  I(0)=I(0)/2                               }
160  FOR R=1 TO N STEP 2                       } Generates
170  X=X0+L*R/N                                } element
180  GOSUB 300                                 } of column 0.
190  I(0)=I(0)+L*Y/N                           }
200  NEXT R                                    }
210  PRINT K;I(0);                             }
220  FOR J=1 TO K                              }
230  U=I(J)                                    } Generates
240  I(J)=(4^J*I(J-1)-T)/(4^J-1)               } remaining
250  T=U                                       } elements of
260  PRINT I(J);                               } same row.
270  NEXT J                                    }
280  PRINT                                     Terminates row.
290  GOTO 110                                  Returns for next cycle.
300  Y=X*X*COS(X)                              Subroutine.
310  RETURN                                    RETURN statement.
320  DATA 0,1.570796327                        Data line for end points.
```

The new program operates in exactly the same way as the first program, except that the lines to be filled in by the user are now 300 and 320. Results are identical to those found with the first program. The lengths and running times of the two programs are almost identical. However, the second program uses much less space in the data memory.

Since the Romberg process uses the end points, indeterminacies sometimes occur. Consider the integral of Eq. 4-13 of Sec. 4-2, which is

$$I = \int_0^1 \frac{\arcsin x}{x}\, dx$$

The integrand is indeterminate at the point $x = 0$, but the limit is clearly 1. The procedure is similar to that used in Sec. 4-1 with Simpson's rule. We have to revise line 50 as well as 300 and 320. The edited lines are

```
 50  Y=1
300  Y=ATN(X/SQR(1-X*X))/X
320  DATA 0,1
```

The integrand is well behaved at the upper limit $x = 1$. However, the arc tangent term in the amended line 300 is not. Therefore we need the further revision

```
80  Y=2*ATN(1)
```

The correct value of the integral is $\pi \ln 2/2 = 1.088793$. Results given by the program appear in the following table:

k \ j	0	1	2	3	4	5
0	1.2854					
1	1.1663	1.1266				
2	1.1185	1.1026	1.1010			
3	1.0999	1.0938	1.0932	1.0930		
4	1.0929	1.0906	1.0904	1.0903	1.0903	
5	1.0903	1.0894	1.0893	1.0893	1.0893	1.0893

For this problem the Romberg method has little merit; the Romberg results on the diagnonal are not much better than the Simpson results in the second column.

Like the Gauss method, the Romberg method is sensitive to the form of the integrand. As in Sec. 4-2, we now consider the equivalent integral

$$I = \int_0^{\pi/2} \frac{x}{\tan x}\, dx$$

which is obtained by writing $\sin x$ for x in the integral of Eq. 4-13. The integrand is indeterminate at the lower limit, with a limiting value of 1. Also, the factor $\tan x$ is infinite at the upper limit. The following changes are now edited into the program:

```
 50 Y=1
 80 Y=0
300 Y=X/TAN(X)
320 DATA 0,1.570796327
```

Numerical results appear in the following table:

k \ j	0	1	2	3
0	.7854			
1	1.0095	1.084266		
2	1.0687	1.088427	1.088704	
3	1.0838	1.088768	1.088791	1.088792

It can be seen that these results are much better than those obtained with the original integral of Eq. 4-13. The Simpson results in the second

column are also greatly improved, although to a lesser extent than the Romberg results.

4-4. Integrals with Discontinuous Integrands

The methods considered in the first three sections apply only to proper integrals. The integrals are continuous and the intervals are finite. In this section the intervals are still finite, but the integrands are infinite at one or both end points. Simpson's rule and Romberg integration break down for an integral of this type, since y_0 or y_n or both are infinite. It is possible to obtain a numerical result by applying Gauss integration directly, since the end points do not appear explicitly in the formulas, but this procedure is unsound and does not usually lead to good results. The best procedure is to transform the improper integral into a proper integral, then use a method from one of the first three sections. We shall consider three commonly used methods of removing a singularity in the integrand.

One useful method is substitution. We illustrate this with the integral

$$I = \int_0^{2.25} \frac{e^{-x}}{\sqrt{x}}\, dx \tag{4-19}$$

which has an infinite integrand at the lower limit. We write x^2 for x. Then

$$I = \int_0^{1.5} 2e^{-x^2}\, dx$$

This is a proper integral. We use the modified Gauss program of Sec. 4-2. Lines 200 and 230 are edited to

```
200  Y=2*EXP(-X*X)
230  DATA 0,1.5
```

With $m = 1$ the numerical result is $I = 1.712376787$, which is accurate to ten significant figures. (This can easily be verified by using the program of Sec. 3-3.)

A second method is integration by parts. An example is provided by the integral

$$I = \int_0^1 \frac{\ln x}{1 + x^2}\, dx \tag{4-20}$$

which has a logarithmic singularity at the lower limit. We find that

$$I = \Big[\ln x \arctan x\Big]_0^1 - \int_0^1 \frac{\arctan x}{x}\, dx$$

We find by elementary calculus that the expression in brackets is equal to zero at both limits. It follows that

$$I = -\int_0^1 \frac{\arctan x}{x}\,dx$$

The new integrand is finite everywhere, with a limiting value of 1 at the lower limit. The integral can be evaluated numerically as it stands. However, it is somewhat similar to the integral of Eq. 4-13, and it leads to a rather inefficient numerical process because of the arc tangent factor. It is desirable to transform the integral further by writing tan x for x. Then we have

$$I = -\int_0^{\pi/4} \frac{x}{\sin x \cos x}\,dx = -\int_0^{\pi/2} \frac{x}{2 \sin x}\,dx$$

We use the modified Gauss program. Lines 200 and 230 become

```
200  Y=X/SIN(X)/2
230  DATA 0,1.570796327
```

With $m = 1$, we find that $I = -.9159655943$, which is correct to ten significant figures.

A third method of eliminating a singularity is to add and subtract a related integral. We again use the integral of Eq. 4-20 as an example. We rewrite it as

$$I = \int_0^1 \ln x\,dx - \int_0^1 \left(\ln x - \frac{\ln x}{1+x^2}\right) dx$$

$$= -I - \int_0^1 \frac{x^2 \ln x}{1+x^2}\,dx$$

The new integral is proper; the integrand is equal to zero at the lower limit. We apply the modified Gauss program. Lines 200 and 230 become

```
200  Y=X*X*LOG(X)/(1+X*X)
230  DATA 0,1
```

For several values of m, the results are

m	1	2	3	4	5
$-I$	.9159667	.91596571	.91596563	.91596561	.915965602

Some further examples may be of interest. Consider

$$I = \int_0^1 \frac{\ln x}{\sqrt{1-x}}\,dx \qquad (4\text{-}21)$$

The integrand is infinite at the lower limit. (It is zero at the upper limit.) We write $\sin^2 x$ for x. Then it follows that

$$I = 4\int_0^{\pi/2} \sin x \ln \sin x \, dx$$

which is proper. The new integral is identical to the integral of Eq. 4-14, which has already been evaluated. The result is $I = -1.227411278$.

This procedure can be made more general. An integral of the type

$$I = \int_a^b \frac{f(x)}{\sqrt{b-x}} dx \tag{4-22}$$

can often be evaluated by writing $b \sin^2 x$ for x. This usually leads to a successful result when $f(x)$ is a continuous function, and sometimes succeeds even when it is not, as in the preceding example.

It is not always easy to see by inspection what procedure will lead to a satisfactory result; sometimes it may be necessary to try two or more methods or a combination of methods. The integral

$$I = \int_0^1 \frac{\ln x}{\sqrt{1-x^2}} dx \tag{4-23}$$

resembles the integral of Eq. 4-21, and we might try to evaluate it by an analogous procedure, writing $\sin x$ for x. However, this does not work; the resulting integral still has a logarithmic singularity at the lower limit. Integration by parts leads to a more satisfactory result. Thus

$$I = \left[\ln x \arcsin x\right]_0^1 - \int_0^1 \frac{\arcsin x}{x} dx$$

$$= -\int_0^1 \frac{\arcsin x}{x} dx$$

The new integral is identical to the integral of Eq. 4-13, which has already been evaluated. The result is $I = -1.088793045$.

There are special methods of the Gauss type for a number of integrals, both proper and improper. Some of these are more complicated than the basic Gauss method; others are simpler. One useful special method known as Gauss-Chebyshev integration evaluates integrals of the form

$$I = \int_a^b \frac{f(x)dx}{[(x-a)(b-x)]^{1/2}} \tag{4-24}$$

where $f(x)$ is a continuous function. (This can be evaluated by integration by parts followed by the standard Gauss integration, but the special method is simpler and more efficient.) The formulas are

$$h = \frac{b-a}{2} \qquad x_m = a + h \tag{4-25a,b}$$

$$x_j = x_m + h \cos\frac{j\pi}{2n} \tag{4-25c}$$

$$I = \frac{\pi}{n} \sum_{j=1,3,5}^{2n-1} f(x_j) \tag{4-25d}$$

This method is much neater than the standard Gauss method. No tabular data are required because the x_is are given by an explicit formula and there are no weighting factors. The program, which follows, is essentially self-explanatory, but it differs in one detail from the earlier programs of this chapter. Since the integrand is evaluated at only one point in the program, this is done directly in line 90; there is no subroutine. Lines 90 and 150 are set up to evaluate the integral

$$I = \int_{-1}^{1} \frac{\cos x}{\sqrt{1-x^2}}\,dx = 2\int_{0}^{1} \frac{\cos x}{\sqrt{1-x^2}}\,dx \tag{4-26}$$

```
  1 REM: GAUSS-CHEBYSHEV INTEGRATION
 10 READ A,B                          Reads values of a and b.
 20 PRINT                             Generates blank line.
 30 INPUT "N=";N                      Calls for value of n.
 40 S=0                               Initializes S.
 50 H=(B-A)/2                         Calculates half-length of interval.
 60 XM=A+H                            Calculates abscissa of midpoint.
 70 FOR J=1 TO 2*N STEP 2
 80 X=XM+H*COS(2*J/N*ATN(1))
 90 Y=COS(X)                          } Calculates S.
100 S=S+Y
110 NEXT J
120 I=4*S*ATN(1)/N                    Calculates result.
130 PRINT "I=";I                      Prints result.
140 GOTO 20                           Returns for next value of n.
150 DATA -1,1                         Data line for end points.
```

The operation of this program is similar to that of the program for modified Gauss integration. Various levels of approximation are obtained by entering a value of n each time a question mark appears on the screen. After the results have converged satisfactorily, the execution of the program is terminated by pressing the BREAK key. The results for several values of n are

n	2	3	4	5	6
I	2.388	2.40407	2.4039388	2.403939432	2.403939431

The last result is correct to ten significant figures. Other integrals can be evaluated with the same program by editing lines 90 and 150.

Other Gauss-Chebyshev algorithms are available for integrals of the type

$$\int_a^b \left(\frac{x-a}{b-x}\right)^{1/2} f(x)dx \qquad \int_a^b [(x-a)(b-x)]^{1/2} f(x)dx \tag{4-27a,b}$$

where $f(x)$ is a continuous function. However, it is not necessary to write special programs for these. The program just given can be used by simply redefining $f(x)$ to include a factor of $(x-a)$ or $(x-a)(b-x)$. The integral of Eq. 4-27b can be evaluated directly by standard Gauss integration, since it is proper, but the special program is more efficient.

Formulas and tables for various types of Gauss integration can be found in references 1 and 11.

4-5. Integrals with Infinite Intervals

Integrals with infinite intervals are sometimes troublesome. Usually the best procedure for an integral of this type is a substitution. Consider

$$I = \int_0^\infty \frac{e^{-x}}{x^2+1}\,dx \tag{4-28}$$

the accurate value of which is .6214496242. (See Prob. 3-1.) The easiest way to evaluate this integral is to write tan x for x. Then we have

$$I = \int_0^{\pi/2} e^{-\tan x}\,dx$$

which is proper. It is always possible to convert an infinite interval into a finite interval by the tangent substitution. However, it sometimes happens that the transformed integral has some new anomaly that causes as much trouble as the original one. In the present case there is no difficulty. We use the modified Gauss program. Lines 200 and 230 become

```
200  Y=EXP(-TAN(X))
230  DATA 0,1.570796327
```

The results for several values of n are

m	1	2	3	4	5
I	.621479	.6214468	.6214499	.62144965	.621449605

Other substitutions can also be used. By writing $-\ln x$ for x, we obtain the integral

$$I = \int_0^1 \frac{dx}{(\ln x)^2 + 1}$$

However, this leads to a less efficient evaluation than the one just given.

An alternate procedure is to first break the interval and then make a substitution in one of the new integrals. Thus

$$I = \int_0^1 \frac{e^{-x}}{x^2+1}\,dx + \int_1^\infty \frac{e^{-x}}{x^2+1}\,dx \tag{4-29}$$

By writing $1/x$ for x in the second integral, then combining the two integrals, we arrive at the result

$$I = \int_0^1 \frac{e^{-x} + e^{-1/x}}{x^2+1}\,dx$$

Lines 200 and 230 of the modified Gauss program become

```
200  Y=(EXP(-X)+EXP(-1/X))/(X*X+1)
230  DATA 0,1
```

The results for several values of m are

m	1	2	3	4	5
I	.6214513	.62144989	.621449601	.621449625	.6214496242

Another procedure that is sometimes used as a last resort is to terminate the interval at some large but finite value of x, then apply numerical integration. Thus, for example, we can write

$$\int_0^\infty \frac{e^{-x}}{x^2+1}\,dx = \int_0^{10} \frac{e^{-x}}{x^2+1}\,dx + \int_{10}^\infty \frac{e^{-x}}{x^2+1}\,dx \tag{4-30}$$

We discard the second integral on the right and apply the modified Gauss program to the first. Lines 200 and 230 become

```
200  Y=EXP(-X)/(X*X+1)
230  DATA 0,10
```

For several values of m, we obtain the results

m	1	2	3	4
I	.62103	.6214420	.6214455	.6214489

We take the final result as .621449, which is very close to the correct value. It is easy to check the error incurred by dropping the last term in Eq. 4-30. We observe that

$$\int_{10}^{\infty} \frac{e^{-x}}{x^2+1}\,dx < \frac{1}{101}\int_{10}^{\infty} e^{-x}dx = \frac{1}{101 \cdot e^{10}} = .00000045$$

There are two special methods of the Gauss type for the evaluation of integrals with infinite intervals. Unfortunately their utility is limited. An integral of the type

$$I = \int_a^{\infty} e^{-x} f(x)dx \tag{4-31}$$

can be evaluated by a procedure known as Gauss-Laguerre integration. The difficulty with this method is that the points are not symmetrically distributed, so a solution with n points requires $2n$ data parameters (n ζ_js and n w_js). The method sometimes requires very extensive tabular data for a satisfactory level of accuracy; for example, a 20-point solution requires 40 numerical constants in the data lines. It is possible to break the interval into a finite part and an infinite part, as in Eq. 4-30, then use the modified Gauss program for the finite part and Gauss-Laguerre integration for the infinite part.

An integral of the type

$$I = \int_{-\infty}^{\infty} e^{-x^2} f(x)dx \tag{4-32}$$

can be evaluated by Gauss-Hermite integration. In this procedure the points are symmetrically distributed, but there is another drawback. In most integrals of the type in Eq. 4-32 that occur in practice, the lower limit is zero instead of $-\infty$. The two cases are not equivalent unless $f(x)$ happens to be an even function.

Problems

Evaluate the following integrals numerically. (Analytical results are given to make it easy to check the numerical evaluations.)

4-1. $\displaystyle\int_0^1 \frac{x^3}{1+x}\,dx = \frac{5}{6} - \ln 2$

4-2. $\displaystyle\int_0^1 \frac{dx}{1+x+x^2} = \frac{\pi}{3\sqrt{3}}$

4-3. $\displaystyle\int_0^1 \frac{dx}{x^2+5x+6} = \ln\frac{9}{8}$

4-4. $\displaystyle\int_0^1 \frac{dx}{\sqrt{1+x^2}} = \ln(\sqrt{2}+1)$

4-5. $\displaystyle\int_0^{\pi/2} \left(\frac{x}{\sin x}\right)^2 dx = \pi \ln 2$

4-6. $\displaystyle\int_0^{\pi} \frac{dx}{(2+\cos x)^2} = \frac{2\pi}{3\sqrt{3}}$

4-7. $\displaystyle\int_0^{\pi} \frac{x \sin x}{1+\cos^2 x}\, dx = \frac{\pi^2}{4}$

4-8. $\displaystyle\int_0^{\pi/4} \ln(1+\tan x)\, dx = \frac{\pi}{8} \ln 2$

4-9. $\displaystyle\int_0^{\pi/2} x^n \sin x\, dx \qquad n = 0,1,2,3,4,5$

4-10. $\displaystyle\int_0^{\pi/2} \ln \sin x\, dx = -\frac{\pi}{2} \ln 2$

4-11. $\displaystyle\int_0^{\pi} x \ln \sin x\, dx = -\frac{\pi^2}{2} \ln 2$

4-12. $\displaystyle\int_1^2 \frac{dx}{[(x-1)(2-x)]^{1/2}} = \pi$

4-13. $\displaystyle\int_1^2 \left(\frac{2-x}{x-1}\right)^{1/2} dx = \frac{\pi}{2}$

4-14. $\displaystyle\int_1^2 [(x-1)(2-x)]^{1/2}\, dx = \frac{\pi}{8}$

4-15. $\displaystyle\int_0^{\infty} \frac{dx}{x^4+1} = \frac{\pi}{2\sqrt{2}}$

4-16. $\displaystyle\int_0^{\infty} \frac{x\, dx}{e^x-1} = \frac{\pi^2}{6}$

4-17. $\displaystyle\int_0^{\infty} \frac{x\, dx}{e^{\alpha x}-1} \qquad \alpha = 0.5,1.0,1.5,2.0,2.5$

4-18. $\displaystyle\int_0^{\infty} \frac{x\, dx}{e^x+1} = \frac{\pi^2}{12}$

4-19. $\displaystyle\int_0^1 \frac{x \ln x}{1-x^2}\, dx = -\frac{\pi^2}{24}$

4-20. $\displaystyle\int_0^{\infty} \left(\frac{1}{e^x-1} - \frac{e^{-x}}{x}\right) dx = \gamma = .5772156649$

4-21. In Sec. 3-2, we studied the exponential integral

$$E_1(x) = \int_x^{\infty} \frac{e^{-t}}{t}\, dt$$

Evaluate the function $xe^x E_1(x)$ for $x = 2, 4, 6, 8, 10$, and check the results against those given in Chapter 3.

4-22. Verify the following limits used in this chapter:

a. $\lim_{x \to 0} \frac{\sin x}{x} = 1$

b. $\lim_{x \to 0} \frac{\arcsin x}{x} = 1$

c. $\lim_{x \to 0} \frac{x}{\tan x} = 1$

d. $\lim_{x \to 0} \frac{\arctan x}{x} = 1$

e. $\lim_{x \to 0} \frac{1 - \cos x}{\tan x} = 0$

f. $\lim_{x \to 0} \frac{x}{e^x - 1} = 1$

4-23. Verify the following limits used in this chapter:

a. $\lim_{x \to 0^+} [x \ln x] = 0$

b. $\lim_{x \to 0^+} [\ln x \arcsin x] = 0$

c. $\lim_{x \to 0^+} [\ln x \arctan x] = 0$

d. $\lim_{x \to 0^+} [(1 - \cos x) \ln \sin x] = 0$

e. $\lim_{x \to 0^+} \frac{x^2 \ln x}{1 + x^2} = 0$

f. $\lim_{x \to 1^-} \frac{\ln x}{\sqrt{1 - x}} = 0$

5
Differential Equations

5-1. First-Order Differential Equations

THE RUNGE-KUTTA METHOD

This is one of the most widely used methods of solving differential equations. We consider the general first-order differential equation

$$\frac{dy}{dx} = y' = f(x,y) \tag{5-1}$$

Let the value of y be known at one point, say $y = y_j$ at $x = x_j$. Then the value of y at a neighboring point $j + 1$ is given by the equations*

$$q_{(0)} = f(x_j, y_j) \tag{5-2a}$$

$$q_{(1)} = f\left(x_j + \frac{h}{2}, y_j + \frac{hq_{(0)}}{2}\right) \tag{5-2b}$$

* Derivations of all of the equations used in this chapter can be found in books on numerical analysis; see, for example, reference 5 or 10. Also see Sec. 2 of the appendix.

$$q_{(2)} = f\left(x_j + \frac{h}{2}, y_j + \frac{hq_{(1)}}{2}\right) \tag{5-2c}$$

$$q_{(3)} = f(x_j + h, y_j + hq_{(2)}) \tag{5-2d}$$

$$y_{j+1} = y_j + \frac{h}{6}(q_{(0)} + 2q_{(1)} + 2q_{(2)} + q_{(3)}) \tag{5-2e}$$

where h is the length of the interval. The subscripts enclosed in parentheses designate the Runge-Kutta parameters, all of which refer to the same interval. Subscripts without parentheses denote the interval (or subinterval).

If the desired point is some distance from the starting point, the interval is divided into a number of increments each of length h, where

$$h = \frac{x_n - x_0}{n} \tag{5-3}$$

and n is the number of increments.

Before writing the program, it is desirable to rewrite the foregoing equations as follows:

$$x_{(0)} = x_j \qquad y_{(0)} = y_j$$

$$x_{(1)} = x_j + \frac{h}{2} \qquad y_{(1)} = y_j + \frac{hq_{(0)}}{2}$$

$$x_{(2)} = x_j + \frac{h}{2} \qquad y_{(2)} = y_j + \frac{\mathrm{hq}_{(1)}}{2}$$

$$x_{(3)} = x_j + h \qquad y_{(3)} = y_j + hq_{(2)}$$

$$q_{(0)} = f(x_{(0)}, y_{(0)})$$

$$q_{(1)} = f(x_{(1)}, y_{(1)})$$

$$q_{(2)} = f(x_{(2)}, y_{(2)})$$

$$q_{(3)} = f(x_{(3)}, y_{(3)})$$

$$y_{j+1} = y_j + \frac{h}{6}(q_{(0)} + 2q_{(1)} + 2q_{(2)} + q_{(3)})$$

Instead of writing out all the foregoing equations individually in the program, we condense them into a form that makes it possible to use a loop. Thus, for $r = 0, 1, 2, 3$, we have

$$C_{(r)} = \frac{r}{2}(3 - r) + 1 \tag{5-4a}$$

$$E_{(r)} = \frac{h}{C} \operatorname{sgn} r \tag{5-4b}$$

$$x_{(r)} = x_j + E_{(r)} \tag{5-4c}$$

$$y_{(r)} = y_j + E_{(r)} q_{(r-1)} \tag{5-4d}$$

$$q_{(r)} = f(x_{(r)}, y_{(r)}) \tag{5-4e}$$

$$S = \sum_{r=0}^{3} C_{(r)} q_{(r)} \tag{5-4f}$$

$$y_{j+1} = y_j + \frac{hS}{6} \tag{5-4g}$$

The expression sgn r in Eq. 5-4b is a signum function; this has been discussed in Secs. 1-1 and 1-2. The present format has two advantages: It leads to a compact program, and it can easily be extended to higher-order differential equations.

A program follows for the equation

$$y' - y = x \tag{5-5}$$

with the initial conditions

$$x = 0 \qquad y = 1$$

Line 1 is the title. Line 10 reads the initial values of x and y from the data line, and lines 20 and 30 print them. The parameters x_j and y_j are not used in the program; their values are denoted as x_0 and y_0 (X0 and Y0 in the program) at the beginning of whatever increment is being considered. Line 40 creates a space between the initial values and the final values that follow. Line 50 calls for x_n, the value of x at the end of the interval, and n, the desired number of subintervals. Line 60 calculates h, using Eq. 5-3. Lines 70 through 190 constitute a nested FOR-NEXT loop that represents Eqs. 5-4. The inner loop of lines 90 through 160 carries out the Runge-Kutta analysis for one increment. Line 140 represents Eq. 5-5. The outer loop of lines 70 through 190 carries the analysis forward from increment to increment, starting at the initial point 0 and ending at the final point n. After the analysis is completed for each increment, the values of x and y at the end of the increment become the new x_0 and y_0 for the next cycle. Line 200 returns the execution of the program to the beginning in preparation for further calculations. The data line 210 contains the initial values x_0 and y_0. The equation line 140 and the data line 210 are filled in by the user each time the program is run.

```
 1  REM: FIRST ORDER D.E. (RUNGE-KUTTA)
10  READ X0,Y0                          Reads values of x0 and y0.
20  PRINT "INITIAL VALUES:"             Prints heading.
30  PRINT "X=";X0,"Y=";Y0               Prints x and y.
40  PRINT                               Generates blank line.
```

```
 50  INPUT "ENTER XN,N";XN,N
 60  H=(XN-X0)/N
 70  FOR J=0 TO N-1
 80  S=0
 90  FOR R=0 TO 3
100  C=R*(3-R)/2+1
110  E=H/C*SGN(R)
120  X=X0+E
130  Y=Y0+E*Q
140  Q=X+Y
150  S=S+C*Q
160  NEXT R
170  X0=X
180  Y0=Y0+H*S/6
190  NEXT J
200  GOTO 30
210  DATA 0,1
```

Calls for values of x_n and n.
Calculates length of subinterval.
Analyzes one increment.
Analyzes entire interval.
Returns for next interval.
Data line for x_0 and y_0.

Results are given in the second column of the table below for the following input data:

x_n	.5	1.0	1.5	2.0
n	5	10	15	20

Each value of n corresponds to the value of x_n directly above it. The increment is $h = .1$. The exact solution

$$y = 2e^x - x - 1 \tag{5-6}$$

is shown for comparison in the first column of the table. Higher accuracy can be obtained by using smaller increments. Results obtained with $h = .05$ are shown in the third column.

x_n \ y_n	Exact	h=.1	h=.05
0.5	1.797443	1.797441	1.797442
1.0	3.436564	3.436559	3.436563
1.5	6.463378	6.463368	6.463377
2.0	11.778112	11.778090	11.778111

When the value of y is required at more than one point, it is not necessary to start the calculation at the original point each time. The program is set up so that the final values of x and y become x_0 and y_0. Also, the GOTO statement in line 200 returns the execution of the program to line 30. The computer prints the result, continues to line 50, then stops and waits for further input. Consider the example just given with

$h = .1$. We have found the value of y at the point $x = .5$. To find the value of y at the point $x = 1$, we simply enter the value 1 and the desired number of increments in the next interval, 5. The result is identical to the one given in the table. Subsequent results are obtained in the same way. After the result at the point $x = 1$ has been found, we obtain the result at the point $x = 1.5$ by entering 1.5, 5.

The Runge-Kutta method has several advantages. It is easy to program and gives good accuracy. Also, the analysis for each increment is self-contained; results at the point $j + 1$ are found by using only data at the point j. Since the calculation does not require a knowledge of results at the left of the starting point, the procedure is self-starting. Also, it is possible to use different values of the increment h in the same calculation; this is sometimes advantageous if the function f varies slowly in one region and rapidly in another region. However, the method requires a great deal of computation, and it has a long running time in comparison with the more efficient method that we shall consider next.

THE ADAMS METHOD

It is sometimes more advantageous to use a different type of method in which the value of y at any point is expressed in terms of the values of f and possibly y at several preceding points. One commonly used method of this type is the Adams method. The equations are

$$y_{j+1} = y_j + \frac{h}{24}(55f_j - 59f_{j-1} + 37f_{j-2} - 9f_{j-3}) \tag{5-7a}$$

$$y_{j+1} = y_j + \frac{h}{24}(9f_{j+1} + 19f_j - 5f_{j-1} + f_{j-2}) \tag{5-7b}$$

Equation 5-7a is used first. This gives the value of y at the point $j + 1$ in terms of its value at point j and the values of f at points j, $j - 1$, $j - 2$, and $j - 3$. This method cannot be used to start an analysis; it is always necessary to have the values of f at three points to the left of the increment being considered. This can be done by using the Runge-Kutta method for the first three increments. The analysis is then switched to the Adams method. Equation 5-7a is not sufficiently accurate to give satisfactory results for most problems. Equation 5-7b is more accurate, but it cannot be used directly because the term f_{j+1} on the right side is not known until after y_{j+1} has been evaluated. The procedure is to use Eq. 5-7a first to obtain a preliminary estimate of y_{j+1}. The corresponding value of f_{j+1} is found by evaluating the function $f(x,y)$ defined by the differential equation. This is the function $q_{(0)}$ of the Runge-Kutta analysis; in the present program it is evaluated in a subroutine at the end. An improved value of y_{j+1} is then found from Eq. 5-7b. A method of this type is known as a predictor-corrector method. Equation 5-7a is the predictor, and Eq. 5-7b is the corrector. This method is much more efficient

than the Runge-Kutta method, because there is less computation in each cycle. In the Runge-Kutta solution, the function f is evaluated four times in each cycle; in the predictor-corrector solution, it is evaluated only once. If the differential equation is complicated, this makes a great difference in running time.

A program follows for the same differential Eq. 5-5 and initial conditions as before. The program is composed of several segments. Lines 1 through 50 contain some preliminary statements that are almost identical to those of the Runge-Kutta program. Lines 60 through 190 constitute the Runge-Kutta loop. This also corresponds very closely to the earlier program with the exception of line 140, which is new. We temporarily pass over lines 140, 200, and 210. Lines 220 through 270 constitute the Adams loop, which represents Eq. 5-7. The parameters f_j, f_{j-1}, f_{j-2}, and f_{j-3} are denoted in the program as F0, F1, F2, and F3, respectively. This is not a FOR-NEXT loop; the loop is generated by the incrementing of j in line 220 together with the IF-THEN statement of line 270. Lines 280 through 320 print the results and prepare for further calculations. Lines 330 through 350 are the subroutine, the RETURN statement, and the data line. Lines 330 and 350 are filled in by the user each time the program is run.

On each cycle it is necessary to reassign the values of the fs in preparation for the next cycle. This is done in line 200. The old values of f_j, f_{j-1}, and f_{j-2} become the new f_{j-1}, f_{j-2}, and f_{j-3}, respectively. The current value of $q_{(0)}$ becomes the new f_j. This segment operates with both the Runge-Kutta loop and the Adams loop. Execution is transferred from the Runge-Kutta loop by line 140 and returned by line 210. On the cycles $j = 0,1,2$, the Runge-Kutta loop calculates f_0, y_1, f_1, y_2, f_2, and y_3. It then starts the cycle $j = 3$ and calculates f_3. The parameters f_0, f_1, f_2, and f_3 are stored as f_{j-3}, f_{j-2}, f_{j-1}, and f_j (F3, F2, F1, and F0 in the program) in preparation for the Adams loop. On the cycle $j = 3$ the execution does not return to the Runge-Kutta loop but passes to the Adams loop, where it remains thereafter.* The execution is controlled by line 210, not by the FOR statement in line 60. The value of the upper limit in line 60 is immaterial, provided that it is not less than 3.

```
 1  REM: FIRST ORDER D.E. (RUNGE-KUTTA & ADAMS)
10  READ X0,Y0                          Reads values of x0 and y0.
20  PRINT "INITIAL VALUES:
    X=";X0, "Y=";Y0                     } Prints initial values.
30  PRINT                               Generates blank line.
40  INPUT "ENTER XN,N";XN,N             Calls for values of xn and n.
50  H=(XN-X0)/N                         Calculates length of subinterval.
```

* The FOR-NEXT loop is never completed because of the final transfer on the cycle $j = 3$. With some computers, an incomplete FOR-NEXT loop may cause a malfunction when a subsequent FOR-NEXT loop is executed. In this program there is no trouble because there is no subsequent FOR-NEXT loop.

```
 60 FOR J=0 TO 3                          ⎫
 70 S=0                                   ⎪
 80 FOR R=0 TO 3                          ⎪
 90 C=R*(3-R)/2+1                         ⎪
100 E=H/C*SGN(R)                          ⎪
110 X=X0+E                                ⎪
120 Y=Y0+E*Q                              ⎬ Runge-Kutta loop.
130 GOSUB 330                             ⎪
140 IF R=0 THEN 200                       ⎪
150 S=S+C*Q                               ⎪
160 NEXT R                                ⎪
170 X0=X                                  ⎪
180 Y0=Y0+H*S/6                           ⎪
190 NEXT J                                ⎭
200 F3=F2:F2=F1:F1=F0:F0=Q                Reassignments.
210 IF J<3 THEN 150                       Controls execution.
220 J=J+1                                 ⎫
230 Y=Y0+H*(55*F0-59*F1+37*F2-9*F3)/24    ⎪
240 X=X+H                                 ⎬ Adams
250 GOSUB 330                             ⎪ loop.
260 Y0=Y0+H*(9*Q+19*F0-5*F1+F2)/24        ⎪
270 IF J<N THEN 200                       ⎭
280 PRINT "X=";X,"Y=";Y0                  Prints results.
290 PRINT                                 Generates blank line.
300 INPUT "ENTER N";N                     Calls for new value of n.
310 N=J+N                                 Adjusts values of n.
320 GOTO 200                              Returns for next interval.
330 Q=X+Y                                 Subroutine.
340 RETURN                                RETURN statement.
350 DATA 0,1                              Data line for x0 and y0.
```

This program differs in one respect from all previous programs in this book. Until now we have consistently used only one statement on each line. This has two advantages. Programs written in this way can be used as they stand on almost any computer. Also, they are usually a little easier to read than programs with multiple statements. However, this format becomes rather clumsy when a program contains a number of very short and very closely related consecutive statements. We have combined four reassignments into line 200. This does not limit the generality of the program; for any computer that does not accept multiple statements on one line, line 200 can easily be expanded into lines 200, 201, 202, and 203.

We mention in passing that the reassignments in line 200 could be avoided by using subscripted variables for f. Then, instead of starting with F0 at the beginning of each increment, we would use F(J), where the value of J increases indefinitely as we proceed. However, this does not significantly shorten the program, and it takes much more space in

the data memory, since all the old values of f are retained. If the calculations are carried out to large values of x, this eventually limits the value of x that can be reached. With the present method the calculations can be continued indefinitely, since the same data memory space is continually reused.

Results are given in the second column of the table below for the following input data:

n_n	.5,	1.0,	1.5,	2.0
n	5,	10,	15,	20

Each value of n corresponds to the value of x_n directly above. The increment is $h = .1$. Exact results from Eq. 5-6 are shown for comparison in the first column of the table. Results obtained with $h = .05$ are shown in the third column.

x_n \ y_n	Exact	$h = .1$	$h = .05$
0.5	1.797443	1.797442	1.797443
1.0	3.436564	3.436561	3.436564
1.5	6.463378	6.463371	6.463379
2.0	11.778112	11.778095	11.778114

Like the Runge-Kutta program, this program is set up so that results after the first can be obtained without returning to the original starting point. However, there is an important difference in input. In the Runge-Kutta method, we had a free choice of the increment h for each new interval, so the new values of both x_n and n were entered. In a predictor-corrector solution, the calculations at each point use data from preceding points, so h is fixed. Only the new value of n is entered.

ACCURACY OF NUMERICAL SOLUTIONS

Iteration is sometimes used to improve the accuracy of predictor-corrector solutions. Consider the Adams program. After the corrected value of y_{j+1} has been found from Eq. 5-7b in line 260 of the program, it is possible to run the subroutine again with the corrected value of y_{j+1} and obtain an improved value of f_{j+1}. The calculation with Eq. 5-7b is then repeated, using the improved value of f_{j+1} to obtain a more accurate value of y_{j+1}. This cycle can be repeated as many times as desired. However, this refinement increases the length and running time of the program, and it does not guarantee accurate results. The iterations do not approach the exact result because Eq. 5-7b is not exact; it has an inherent error for any value of h greater than zero. If highly accurate results are desired, the most effective and reliable procedure is to reduce the value of h. In all the methods of this chapter, the error per step is of order h^5, and the

error for the entire interval is of order h^4, that is, of order n^{-4}. Hence the accuracy of the results improves rapidly as n is increased. However, this procedure cannot be continued indefinitely because the accuracy of the results is eventually limited by roundoff error.

Normally results approach the exact values as $h \to 0$ (with the exception of roundoff errors). However, it occasionally happens that a numerical evaluation breaks down, and no satisfactory result can be obtained. Extensive discussions of this problem can be found in books on numerical analysis. Here we simply remark that the frequency with which difficulties occur depends on the method used. The Runge-Kutta method is very reliable. Predictor-corrector solutions occasionally break down. However, the Adams method is the most reliable method of this type; anomalous results seldom occur in problems of practical interest.

Occasionally a numerical solution may be unstable because of a peculiarity in the differential equation and the initial conditions regardless of the computational method used. A simple heuristic discussion may clarify this problem. Consider the differential Eq. 5-5. The analytical solution is

$$y = Ae^x - x - 1$$

where A is a numerical constant. Suppose that the initial condition is $y = -1$ at $x = 0$. Then $A = 0$. However, if we carry out a numerical evaluation and fit the analytical solution to the resulting points, we will obtain a value of A that is very small but not identically zero. If the numerical evaluation is continued to very large values of x, the spurious exponential term eventually overshadows the legitimate terms, and the solution breaks down.

5-2. Systems of Differential Equations; Second-Order Differential Equations

THE RUNGE-KUTTA METHOD

This method can easily be extended to simultaneous differential equations. Consider the two simultaneous equations

$$y' = f_a(x,y,u) \qquad u' = f_b(x,y,u) \tag{5-8a,b}$$

The Runge-Kutta formulas are

$$q_{a(0)} = f_a(x_j, y_j, u_j)$$

$$q_{b(0)} = f_b(x_j, y_j, u_j)$$

$$q_{a(1)} = f_a\left(x_j + \frac{h}{2}, y_j + \frac{h}{2} q_{a(0)}, u_j + \frac{h}{2} q_{b(0)}\right)$$

$$q_{b(1)} = f_b\left(x_j + \frac{h}{2}, y_j + \frac{h}{2} q_{a(0)}, u_j + \frac{h}{2} q_{b(0)}\right)$$

$$q_{a(2)} = f_a\left(x_j + \frac{h}{2}, y_j + \frac{h}{2} q_{a(1)}, u_j + \frac{h}{2} q_{b(1)}\right)$$

$$q_{b(2)} = f_b\left(x_j + \frac{h}{2}, y_j + \frac{h}{2} q_{a(1)}, u_j + \frac{h}{2} q_{b(1)}\right)$$

$$q_{a(3)} = f_a(x_j + h, y_j + hq_{a(2)}, u_j + hq_{b(2)})$$

$$q_{b(3)} = f_b(x_j + h, y_j + hq_{a(2)}, u_j + hq_{b(2)})$$

$$y_{j+1} = y_j + \frac{h}{6}(q_{a(0)} + 2q_{a(1)} + 2q_{a(2)} + q_{a(3)})$$

$$u_{j+1} = u_j + \frac{h}{6}(q_{b(0)} + 2q_{b(1)} + 2q_{b(2)} + q_{b(3)})$$

We proceed to write these equations directly in the condensed format. For $r = 0,1,2,3$, the equations are

$$C_{(r)} = \frac{r}{2}(3 - r) + 1 \tag{5-9a}$$

$$E_{(r)} = \frac{h}{C} \operatorname{sgn} r \tag{5-9b}$$

$$x_{(r)} = x_j + E_{(r)} \tag{5-9c}$$

$$y_{(r)} = y_j + E_{(r)} q_{a(r-1)} \tag{5-9d}$$

$$u_{(r)} = u_j + E_{(n} q_{b(r-1)} \tag{5-9e}$$

$$q_{a(r)} = f_a(x_{(r)}, y_{(r)}, u_{(r)}) \tag{5-9f}$$

$$q_{b(r)} = f_b(x_{(r)}, y_{(r)}, u_{(r)}) \tag{5-9g}$$

$$S_a = \sum_{r=0}^{3} C_{(r)} q_{a(r)} \tag{5-9h}$$

$$S_b = \sum_{r=0}^{3} C_{(r)} q_{b(r)} \tag{5-9i}$$

$$y_{j+1} = y_j + \frac{hS_a}{6} \tag{5-9j}$$

$$u_{j+1} = u_j + \frac{hS_b}{6} \tag{5-9k}$$

A program follows for the two simultaneous equations

$$y' = u \qquad u' = 3u - 2y + x \tag{5-10a,b}$$

with the initial conditions

$$x = 0 \qquad y = 1 \qquad u = 1$$

The program is organized in exactly the same way as the Runge-Kutta program of Sec. 5-1. The only thing new is that there are now two equation lines to be filled in by the user: lines 160 and 170. The data line 260, which contains the initial values of x, y, and u, is also filled in by the user.

```
 1  REM: TWO SIMULTANEOUS FIRST ORDER D.E.'S (RUNGE-KUTTA)
10  READ X0,Y0,U0                          Reads initial values.
20  PRINT "INITIAL VALUES:"                Prints heading.
30  PRINT "X=";X0,"Y=";Y0,                 }
    "U=";U0                                } Prints x,y,y'.
40  PRINT                                  Generates blank line.
50  INPUT "ENTER XN,N";XN,N                Calls for values of x_n and n.
60  H=(XN-X0)/N                            Calculates length of subinterval.
70  FOR J=0 TO N-1                         }
80  SA=0                                   }
90  SB=0                                   }
100 FOR R=0 TO 3                           }
110 C=R*(3-R)/2+1                          }
120 E=H/C*SGN(R)                           }
130 X=X0+E                                 }
140 Y=Y0+E*QA                              } Analyzes entire interval.
150 U=U0+E*QB                              } Analyzes one increment.
160 QA=U                                   }
170 QB=3*U-2*Y+X                           }
180 SA=SA+C*QA                             }
190 SB=SB+C*QB                             }
200 NEXT R                                 }
210 X0=X                                   }
220 Y0=Y0+H*SA/6                           }
230 U0=U0+H*SB/6                           }
240 NEXT J                                 }
250 GOTO 30                                Returns for next interval.
260 DATA 0,1,1                             Data line for initial values.
```

This program operates in exactly the same way as the Runge-Kutta program of Sec. 5-1. Numerical results for y appear in the following table:

x_n \ y_n	Exact	h=.1	h=.05
0.5	1.679570	1.679563	1.679570
1.0	3.097264	3.097222	3.097261
1.5	6.521384	6.521214	6.521373
2.0	15.399538	15.398921	15.399496

The exact solution

$$y = \frac{1}{4}(e^{2x} + 2x + 3) \tag{5-11}$$

is shown for comparison.

The foregoing program can be used to solve second-order differential equations. In fact, by eliminating u from Eqs. 5-10a,b, we find that we have already solved the differential equation

$$y'' - 3y' + 2y = x \tag{5-12}$$

with the initial conditions

$$x = 0 \qquad y = 1 \qquad y' = 1$$

If we are primarily interested in solving second-order differential equations directly, the foregoing solution can be organized a little more neatly. Consider the general second-order differential equation

$$y'' = f(x,y,y') \tag{5-13}$$

We reconcile this with Eqs. 5-8a,b by setting $y' = u = f_a$ and $f = f_b$. Also $q_a = u$, and q_b may be denoted simply as q. We also let $S = S_b$ and $T = S_a$. Equation 5-9f now reduces to the identity $u = u$. The other equations of the set (Eq. 5-9) become

$$C_{(r)} = \frac{r}{2}(3 - r) + 1 \tag{5-14a}$$

$$E_{(r)} = \frac{h}{C} \operatorname{sgn} r \tag{5-14b}$$

$$x_{(r)} = x_j + E_{(r)} \tag{5-14c}$$

$$y_{(r)} = y_j + E_{(r)} u_{(r-1)} \tag{5 -14d}$$

$$u_{(r)} = u_j + E_{(r)} q_{(r-1)} \tag{5-14e}$$

$$q_{(r)} = f(x_{(r)}, y_{(r)}, u_{(r)}) \tag{5-14f}$$

$$T = \sum_{r=0}^{3} C_{(r)} u_{(r)} \tag{5-14g}$$

$$S = \sum_{r=0}^{3} C_{(r)} q_{(r)} \tag{5-14h}$$

$$y_{j+1} = y_j + \frac{hT}{6} \tag{5-14i}$$

$$u_{j+1} = u_j + \frac{hS}{6} \tag{5-14j}$$

We have not replaced u by y' because y' cannot be used as a program variable in most versions of BASIC.

The program follows. It is organized in exactly the same way as the first program of this section, except that now there is only one equation line—line 160—to be filled in by the user. The data line 250, which contains the initial values, is also filled in by the user.

```
 1  REM: SECOND ORDER D.E. (RUNGE-KUTTA)
10  READ X0,Y0,U0                          Reads initial values.
20  PRINT "INITIAL VALUES:"                Prints heading.
30  PRINT "X=";X0,"Y=";Y0,                 } Prints x, y, y'.
    "Y'=";U0
40  PRINT                                  Generates blank line.
50  INPUT "ENTER XN,N";XN,N                Calls for values of x_n and n.
60  H=(XN-X0)/N                            Calculates length of subinterval.
70  FOR J=0 TO N-1                         }
80  S=0                                    }
90  T=0                                    }
100 FOR R=0 TO 3                           } Analyzes one increment.
110 C=R*(3-R)/2+1                          }
120 E=H/C*SGN(R)                           }
130 X=X0+E                                 }
140 Y=Y0+E*U                               }
150 U=U0+E*Q                               }
160 Q=3*U-2*Y+X                            }
170 T=T+C*U                                }
180 S=S+C*Q                                }
190 NEXT R                                 }
200 X0=X                                   }
210 Y0=Y0+H*T/6                            }
220 U0=U0+H*S/6                            }
230 NEXT J                                 } Analyzes entire interval.
240 GOTO 30                                Returns for next interval.
250 DATA 0,1,1                             Data line for initial values.
```

Numerical results are identical to those given in the table earlier in this section.

THE ADAMS METHOD

Predictor-corrector methods can also be applied to systems of differential equations or to higher-order differential equations. We consider the second-order differential Eq. 5-13. The same Adams relations (Eq. 5-7) that connect y with y' for a first-order equation now connect y with $y' = u$ and u with $u' = f$. The equations are

$$u_{j+1} = u_j + \frac{h}{24}(55f_j - 59f_{j-1} + 37f_{j-2} - 9f_{j-3}) \tag{5-15a}$$

$$y_{j+1} = y_j + \frac{h}{24}(qu_{j+1} + 19u_j - 5u_{j-1} + u_{j-2}) \tag{5-15b}$$

$$u_{j+1} = u_j + \frac{h}{24}(qf_{j+1} + 19f_j - 5f_{j-1} + f_{j-2}) \tag{5-15c}$$

Equation 5-15a is a predictor equation; Eqs. 5-15b and 5-15c are corrector-type equations. A predictor equation is not needed for y_{j+1}. After a preliminary estimate of u_{j+1} is obtained from Eq. 5-15a, an accurate value of y_{j+1} is obtained directly from Eq. 5-15b. The value of f_{j+1} is then found from the subroutine for Eq. 5-13, and an accurate value of u_{j+1} is obtained from Eq. 5-15c.

The program follows. It is organized in the same way as the second program of Sec. 5-1. One new problem is worth mentioning. Because of Eq. 5-15b, it is necessary to store the last few us as well as the fs. This is done in line 260 of the program. The values of the fs and us are reassigned on the Runge-Kutta cycles $j = 0,1,2$. However, the last Runge-Kutta cycle $j = 3$ is incomplete; it calculates a new value of f, but no new u or y. Therefore only the value of f is reassigned on the last Runge-Kutta cycle; the other reassignments are skipped by line 250. The program operates in exactly the same way as the Adams program of Sec. 5-1. Two lines are filled by the user: the subroutine 410, which is the differential equation, and the data line 430, which contains the initial values. As the program stands, these lines contain the same differential Eq. 5-12 and initial conditions that were used for the Runge-Kutta program.

```
  1 REM: SECOND ORDER D.E. (RUNGE-KUTTA & ADAMS)
 10 READ X0,Y0,U0                          Reads initial values.
 20 PRINT "INITIAL VALUES:                 }
    X=";X0,"Y=";Y0,"Y'=";U0                } Prints initial values.
 30 PRINT                                  Generates blank line.
 40 INPUT "ENTER XN,N";XN,N                Calls for values of x_n and n.
 50 H=(XN-X0)/N                            Calculates length of subinterval.
 60 FOR J=0 TO 3                           }
 70 T=0                                    }
 80 S=0                                    }
 90 FOR R=0 TO 3                           }
100 C=R*(3-R)/2+1                          }
110 E=H/C*SGN(R)                           }
120 X=X0+E                                 }
130 Y=Y0+E*U                               }
140 U=U0+E*Q                               } Runge-Kutta loop.
150 GOSUB 410                              }
160 IF R=0 THEN 240                        }
170 T=T+C*U                                }
180 S=S+C*Q                                }
190 NEXT R                                 }
200 X0=X                                   }
210 Y0=Y0+H*T/6                            }
220 U0=U0+H*S/6                            }
230 NEXT J                                 }
```

```
240  F3=F2:F2=F1:F1=F0:F0=Q
250  IF J=3 THEN 290
260  U2=U1:U1=U0:U0=U
270  Y0=Y
280  IF J<3 THEN 170
290  J=J+1
300  U=U0+H*(55*F0-59*F1+37*F2-9*F3)/24
310  Y=Y0+H*(9*U+19*U0-5*U1+U2)/24
320  X=X+H
330  GOSUB 410
340  U=U0+H*(9*Q+19*F0-5*F1+F2)/24
350  IF J<N THEN 240
360  PRINT "X=";X,"Y=";Y,
     "Y'=";U
370  PRINT
380  INPUT "ENTER N: ";N
390  N=J+N
400  GOTO 240
410  Q=3*U-2*Y+X
420  RETURN
430  DATA 0,1,1
```

Reassignments.

Controls execution.

Adams loop.

Prints results.

Generates blank line.

Calls for new value of n.

Adjusts value of n.

Returns for next interval.

Subroutine.

RETURN statement.

Data line for initial values.

Numerical results appear in the following table. Exact results from Eq. 5-11 are shown for comparison.

x_n \ y_n	Exact	h=.1	h=.05
0.5	1.679570	1.679565	1.679570
1.0	3.097264	3.097153	3.097261
1.5	6.521384	6.520581	6.521360
2.0	15.399538	15.395600	15.399415

SINGULAR POINTS OF DIFFERENTIAL EQUATIONS

We now consider the nonlinear differential equation

$$y'' = -\frac{2y'}{x} - y^5 \tag{5-16}$$

This is known as Emden's equation; it occurs in astrophysics. (The exponent of the second term on the right may be any positive number; we have chosen the value 5 because there is a simple analytical solution for this case that can be used to check the numerical evaluation.) The initial conditions are

$x = 0 \qquad y = 1 \qquad y' = 0$

We may use either the Runge-Kutta program for second-order differential equations or the Adams program. With the Runge-Kutta program, the obvious revisions are

```
160  Q=-2*U/X-Y^5
250  DATA 0,1,0
```

With these revisions only, the program does not work. There is an x in the denominator of the first term on the right side of Eq. 5-16. The point $x = 0$ is said to be a *singular point* of the differential equation, and this causes a malfunction in line 160. Singular points often cause difficulty in numerical solutions of differential equations, but in this case the trouble can easily be corrected by a procedure similar to that used for integrals in Sec. 4-1. We shall find the value of y'' at $x = 0$ by an analytical calculation, then insert this into the program and bypass the GOSUB statement at this point. We refer to the differential Eq. 5-16. Since $y' = 0$ when $x = 0$, the first term on the right is indeterminate. By applying l'Hospital's rule to this term and also using the fact that $y_0 = 1$, we find that

$$y_0'' = -2 \lim_{x \to 0} \frac{y'}{x} - y_0^5 = -2y_0'' - 1 = -\frac{1}{3}$$

We now amend the program to use this value and bypass the GOSUB statement when $x = 0$. We add the following two lines:

```
 65  Q=-1/3
155  IF X=0 THEN 170
```

With the new lines 65, 155, 160 and 250, the program will work successfully. Numerical results can be checked against the analytical solution

$$y = \left(\frac{3}{x^2 + 3}\right)^{1/2} \tag{5-17}$$

There are various methods of dealing with singular points. Unfortunately no one straightforward method is uniformly successful. However, the foregoing procedure also works with the equation

$$4xy'' + 2y' + y = 0 \tag{5-18}$$

and the initial conditions

$$x = 0 \qquad y = 1 \qquad y' = -\frac{1}{2}$$

We rewrite the differential equation as

$$y'' = -\frac{2y' + y}{4x}$$

The point $x = 0$ is a singular point. We observe that the term on the right side of the equation is an indeterminate expression of the type 0/0. An application of l'Hospital's rule leads to

$$y_0'' = -\left[\frac{2y'' + y'}{4}\right]_0 = -\frac{y_0''}{2} + \frac{1}{8} = \frac{1}{12}$$

and the numerical evaluation follows directly.

It is sometimes possible to eliminate a singularity by a substitution. Although it is by no means obvious, the appropriate substitution for Eq. 5-18 is $x = t^2$. This transforms the equation into

$$\frac{d^2y}{dt^2} + y = 0 \tag{5-19}$$

and the initial conditions become

$$t = 0 \qquad y = 1 \qquad \frac{dy}{dt} = 0$$

This leads to an even simpler numerical evaluation than the first method. Results obtained by either method can be checked against the analytical solution

$$y = \cos t = \cos \sqrt{x} \tag{5-20}$$

which follows from Eq. 5-19.

A third method of dealing with a singular point is to expand the function into an infinite series. The procedure is discussed at length in books on differential equations. It is more complicated than the methods used here.

5-3 Fourth-Order Differential Equations

THE RUNGE-KUTTA METHOD

The solutions of Secs. 5-1 and 5-2 can be extended indefinitely to differential equations of any order. Consider the general fourth-order differential equation

$$y^{IV} = f(x,y,y',y'',y''') \tag{5-21}$$

We denote the derivatives y', y'', and y''' by u, v, and w, respectively. The Runge-Kutta equations are, for $r = 0,1,2,3$,

$$E_{(r)} = \frac{h}{12} r(13 - r(q - 2r)) \tag{5-22a}$$

$$x_{(r)} = x_j + E_{(r)} \tag{5-22b}$$

$$y_{(r)} = y_j + E_{(r)} u_{(r-1)} \tag{5-22c}$$

$$u_{(r)} = u_j + E_{(r)} v_{(r-1)} \tag{5-22d}$$

$$v_{(r)} = v_j + E_{(r)} w_{(r-1)} \tag{5-22e}$$

$$w_{(r)} = w_j + E_{(r)} q_{(r-1)} \tag{5-22f}$$

$$q_{(r)} = f(x_{(r)}, y_{(r)}, u_{(r)}, v_{(r)}, w_{(r)}) \tag{5-22g}$$

$$y_{j+1} = y_j + \frac{h}{6}(u_{(0)} + 2u_{(1)} + 2u_{(2)} + u_{(3)}) \tag{5-22h}$$

$$u_{j+1} = u_j + \frac{h}{6}(v_{(0)} + 2v_{(1)} + 2v_{(2)} + v_{(3)}) \tag{5-22i}$$

$$v_{j+1} = v_j + \frac{h}{6}(w_{(0)} + 2w_{(1)} + 2w_{(2)} + w_{(3)}) \tag{5-22j}$$

$$w_{j+1} = w_j + \frac{h}{6}(q_{(0)} + 2q_{(1)} + 2q_{(2)} + q_{(3)}) \tag{5-22k}$$

The present format differs slightly from those of Secs. 5-1 and 5-2. We have calculated $E_{(r)}$ without first calculating $C_{(r)}$, because the latter parameter is not used. We use Eqs. 5-22a,d–f to rewrite Eqs. 5-22h–k as

$$y_{j+1} = y_j + h\left(u_j + \frac{h}{2}\left(v_j + \frac{h}{3}\left(w_j + \frac{h}{4} q_{(0)}\right)\right)\right)$$

$$u_{j+1} = u_j + h\left(v_j + \frac{h}{2}\left(w_j + \frac{h}{6}\left(q_{(0)} + q_{(1)}\right)\right)\right)$$

$$v_{j+1} = v_j + h\left(w_j + \frac{h}{6}\left(q_{(0)} + q_{(1)} + q_{(2)}\right)\right)$$

$$w_{j+1} = w_j + \frac{h}{6}(q_{(0)} + 2q_{(1)} + 2q_{(2)} + q_{(3)})$$

Finally, we rewrite these in the form

$$S_p = \sum_{r=0}^{p-1} q_{(r)} \tag{5-23a}$$

$$y_{j+1} = y_j + h\left(u_j + \frac{h}{2}\left(v_j + \frac{h}{3}\left(w_j + \frac{h}{4} S_1\right)\right)\right) \tag{5-23b}$$

$$u_{j+1} = u_j + h\left(v_j + \frac{h}{2}\left(w_j + \frac{h}{6}S_2\right)\right) \tag{5-23c}$$

$$v_{j+1} = v_j + h\left(w_j + \frac{h}{6}S_3\right) \tag{5-23d}$$

$$w_{j+1} = w_j + \frac{h}{6}(S_4 + S_3 - S_1) \tag{5-23e}$$

A program follows for the equation

$$y^{IV} - 2y''' + 3y'' - 5y' + 3y = 0 \tag{5-24}$$

with the initial conditions

$$x = 0 \qquad y = y' = y'' = y''' = 1$$

The program is organized in the same way as the earlier Runge-Kutta programs, with the exceptions just noted. Two lines are to be filled in by the user: the equation line 160 and the data line 260, which contains the initial values.

```
  1 REM: FOURTH ORDER D.E. (RUNGE-KUTTA)
 10 READ X0,Y0,U0,V0,W0                    Reads initial values.
 20 PRINT "INITIAL VALUES:"                Prints heading.
 30 PRINT "X=";X0,"Y=",Y0,
    "Y'=";U0,"Y''=";V0,                    Prints x, y, y',y'', y'''.
    "Y'''=";W0
 40 PRINT                                  Generates blank line.
 50 INPUT "ENTER XN,N";XN,N                Calls for values of x_n and n.
 60 H=(XN-X0)/N                            Calculates length of subinterval.
 70 S(0)=0                                 Assigns value of S0.
 80 FOR J=0 TO N-1                                              }
 90 FOR R=0 TO 3                           }                    }
100 E=R*(13-R*(9-2*R))/12*H                }                    }
110 X=X0+E                                 }                    }
120 Y=Y0+E*U                               }  Analyzes          }
130 U=U0+E*V                               }  one               }
140 V=V0+E*W                               }  increment.        }
150 W=W0+E*Q                               }                    }  Analyzes
160 Q=2*W-3*V+5*U-3*Y                      }                    }  entire
170 S(R+1)=S(R)+Q                          }                    }  interval.
180 NEXT R                                 }                    }
190 X0=X                                                        }
200 Y0=Y0+H*(U0+H/2*(V0+H/3*(W0+H/4*S(1))))                     }
210 U0=U0+H*(V0+H/2*(W0+H/6*S(2)))                              }
220 V0=V0+H*(W0+H/6*S(3))                                       }
230 W0=W0+H/6*(S(4)+S(3)-S(1))                                  }
240 NEXT J                                                      }
```

```
250 GOTO 30                    Returns for next interval.
260 DATA 0,1,1,1,1             Data line for initial values.
```

The program operates in exactly the same way as the earlier Runge-Kutta programs. Numerical results for y appear in the following table:

x_n \ y_n	Exact	h=.1	h=.05
0.5	1.6487213	1.6487206	1.6487212
1.0	2.7182818	2.7182797	2.7182817
1.5	4.4816891	4.4816839	4.4816887
2.0	7.3890 561	7.3890 448	7.3890 554

The exact solution

$$y = e^x \tag{5-25}$$

is shown for comparison.

THE ADAMS METHOD

The Adams equations for a fourth-order differential equation are

$$w_{j+1} = w_j + \frac{h}{24}(55f_j - 59f_{j-1} + 37f_{j-2} - 9f_{f-3}) \tag{5-26a}$$

$$v_{j+1} = v_j + \frac{h}{24}(9w_{j+1} + 19w_j - 5w_{j-1} + w_{j-2}) \tag{5-26b}$$

$$u_{j+1} = u_j + \frac{h}{24}(9v_{j+1} + 19v_j - 5v_{j-1} + v_{j-2}) \tag{5-26c}$$

$$y_{j+1} = y_j + \frac{h}{24}(9u_{j+1} + 19u_j - 5u_{j-1} + u_{j-2}) \tag{5-26d}$$

$$w_{j+1} = w_j + \frac{h}{24}(9f_{j+1} + 19f_j - 5f_{j-1} + f_{j-2}) \tag{5-26e}$$

The first four equations give a rough estimate of w_{j+1} and accurate values of v_{j+1}, u_{j+1}, and y_{j+1}. After these results have been found, f_{j+1} is evaluated by the subroutine and an accurate value of w_{j+1} is found from the last equation.

The program follows. The first half is almost identical to the Runge-Kutta program given earlier in this section, and the second half is organized in the same way as the Adams programs of Secs. 5-1 and 5-2. The equation subroutine 460 and the data line 480 are filled in by the user. As the program stands, these lines contain the same differential Eq. 5-24 and initial conditions that were used for the Runge-Kutta program earlier in this section.

```
1   REM: FOURTH ORDER D.E. (RUNGE-KUTTA & ADAMS)
10  READ X0,Y0,U0,V0,W0                          Reads initial values.
20  PRINT "INITIAL VALUES:
    X=";X0,"Y=";Y0,"Y'=";U0,                     } Prints initial values.
    "Y''=";V0,"Y'''=";W0
30  PRINT                                        Generates blank line.
40  INPUT "ENTER XN,N";XN,N                      Calls for values of x_n and n.
50  H=(XN-X0)/N                                  Calculates length of subinterval.
60  S(0)=0
70  FOR J=0 TO 3
80  FOR R=0 TO 3
90  E=R*(13-R*(9-2*R))/12*H
100 X=X0+E
110 Y=Y0+E*U
120 U=U0+E*V
130 V=V0+E*W
140 W=W0+E*Q
150 GOSUB 460                                    } Runge-Kutta loop.
160 IF R=0 THEN 250
170 S(R+1)=S(R)+Q
180 NEXT R
190 X0=X
200 Y0=Y0+H*(U0+H/2*(V0
    +H/3*(W0+H/4*S(1))))
210 U0=U0+H*(V0+H/2*(W0+H/6*S(2)))
220 V0=V0+H*(W0+H/6*S(3))
230 W0=W0+H/6*(S(4)+S(3)-S(1))
240 NEXT J
250 F3=F2:F2=F1:F1=F0:F0=Q                       }
260 IF J=3 THEN 320                              }
270 W2=W1:W1=W0:W0=W                             } Reassignments.
280 V2=V1:V1=V0:V0=V                             }
290 U2=U1:U1=U0:U0=U                             }
300 Y0=Y                                         }
310 IF J<3 THEN 170                              Controls execution.
320 J=J+1                                        }
330 W=W0+H*(55*F0-59*F1+37*F2-9*F3)/24           }
340 V=V0+H*(9*W+19*W0-5*W1+W2)/24                }
350 U=U0+H*(9*V+19*V0-5*V1+V2)/24                } Adams loop.
360 Y=Y0+H*(9*U+19*U0-5*U1+U2)/24                }
370 X=X+H                                        }
380 GOSUB 460                                    }
390 W=W0+H*(9*Q+19*F0-5*F1+F2)/24                }
400 IF J<N THEN 250                              }
410 PRINT "X=";X,"Y=";Y,                         } Prints results.
    "Y'=";U, "Y''=";V,"Y'''=";W
420 PRINT                                        Generates blank line.
```

```
430 INPUT "ENTER N: ";N          Calls for new value of n.
440 N=J+N                        Adjusts value of n.
450 GOTO 250                     Returns for next interval.
460 Q=2*W-3*V+5*U-3*Y            Subroutine.
470 RETURN                       RETURN statement.
480 DATA 0,1,1,1,1               Data line for initial values.
```

Numerical results appear in the following table:

x_n \ y_n	Exact	h=.1	h=.05
0.5	1.6487213	1.6487217	1.6487213
1.0	2.7182818	2.7182856	2.7182822
1.5	4.4816891	4.4816994	4.4816899
2.0	7.3890 561	7.3890 769	7.3890 578

Exact results from Eq. 5-25 are shown for comparison.

5-4 Boundary Value Problems

In all the problems considered until now, all the required conditions on y and its derivatives have been specified at one point (that is, one value of the independent variable), which could be used as the starting point for the numerical evaluation. Any problem involving a first-order differential equation is of this type, since there is only one condition to be satisfied. However, when the differential equation is of second or higher order, two or more conditions must be satisfied. If all the conditions are specified at one point, the problem is known as an initial value problem. A problem in which conditions are specified at two points is known as a boundary value problem. The methods that we have considered apply directly to initial value problems. To solve a boundary value problem, an extension of the foregoing methods is necessary.

SECOND-ORDER DIFFERENTIAL EQUATIONS

For a linear second-order differential equation, it is possible to solve a boundary value problem by solving two initial value problems and then using superposition, that is, linear interpolation. The easiest way to understand the method is to consider an example. Suppose that we require the solution of Eq. 5-12 that satisfies the boundary conditions

$$x = 0 \quad y = 1 \qquad x = 1 \quad y' = 0$$

and, in particular, we require the value of y at the point $x = 1$. We guess two values of y_0' (say 0 and 1) and run calculations from one of the programs of Sec. 5-2, using the correct starting values $x_0 = 0$ and $y_0 = 1$ in both cases. Results from the Runge-Kutta program with $h = .05$ appear in the first two rows of the following table:

x_0	y_0	y_0'	x_n	y_n	y_n'
0	1	0	1	−1.573502	−7.865285
0	1	1	1	3.097261	4.194522
0	1	.6521899	1	1.472723	0

The values of y_0' and y_n in the third row are obtained by linear interpolation in the first two rows. Thus

$$y_0' = \frac{7.865285}{7.865285 + 4.194522} = .6521899$$

and

$$y_n = -1.573502(1 - .6521889) + 3.097261 \cdot .6521889$$
$$= 1.472723$$

It is not necessary to type in all the results from the first two rows to obtain the final results. The values of $y_n' = u_n$ and y_n are still in the computer after the second run. The following simple exercise in the prompt mode gives the desired results:

```
B=1/(1+U0/7.865285)
PRINT B
 .6521899
A=−1.573502*(1−B)+Y0*B
PRINT A
 1.472723
```

If the Adams program is used instead of the Runge-Kutta program, the procedure is exactly the same except that the program variables for y_n' and y_n are U and Y—not U0 and Y0.

On most microcomputers it is possible to do the calculation without retyping any numbers. After the first run, some convenient letters that are not used in the program are assigned to y_n and y_n'. Most computers will set these equal to zero if the RUN command is used to start the second run. However, with most computers it is possible to use GOTO as an alternate command that does not lose the variables.

The final result is checked by running the program again, starting with the correct initial values x_0, y_0, and y_0'. This leads to the result

$y_n = 1.472723$, which confirms the interpolation. It also agrees to the full number of digits shown with the exact result, which is

$$y_n = \frac{e(8 + e) - 3}{4(2e - 1)}$$

Ideally the value of y_n' should be zero, but some small nonzero result will be found because of the error of the numerical approximation and roundoff error.

In this example we have made the interpolations manually after running the basic Runge-Kutta program twice. If results are desired for a number of sets of input data, it may be advantageous to incorporate the interpolations into the program.

This procedure is theoretically correct only for linear differential equations. For a boundary value problem involving a nonlinear differential equation, linear interpolation may be used to obtain a first estimate of the result. This must then be refined by trial and error. One of the root-finding methods of Chapter 2 may be combined with the Runge-Kutta method or the Adams method.

FOURTH-ORDER DIFFERENTIAL EQUATIONS

For a fourth-order differential equation, four conditions have to be satisfied. If all these are specified at one point, we have an initial value problem of the type considered in Sec. 5-3. If three conditions are specified at one end of the interval and one condition is specified at the other end, we take $x = x_0$ at the end where three conditions are specified. The remaining boundary condition is handled in the same way as in the case of the second-order differential equation just discussed. If two boundary conditions are specified at each end, the analysis is a little more complicated. The solution of the boundary value problem is obtained by superposition of three initial value solutions. The procedure will be illustrated by an example.

We require the solution of Eq. 5-24 that satisfies the boundary conditions

$$x = 0 \quad y = 1 \quad y' = 0 \qquad x = 1 \quad y'' = 0 \quad y''' = 0$$

and, in particular, we require the value of y at the point $x = 1$. The calculations are set out in the following table. The first three rows of figures are obtained by running the Runge-Kutta program of Sec. 5-3 three times with the input data shown and $h = .05$. The fourth row is obtained by superposition of the first two, and the fifth is obtained by superposition of the first and third. The sixth row is derived from the fourth and fifth.

x_0	y_0	y_0'	y_0''	y_0'''	x_n	y_n''	y_n'''
0	1	0	0	0	1	−2.679810	−6.808803
0	1	0	0	1	1	−.410209	−2.824123
0	1	0	1	0	1	−3.234332	−9.934594
0	1	0	0	1.708745	1	1.198360	0
0	1	0	−2.178266	0	1	−1.471913	0
0	1	0	−.977558	.941898	1	0	0

The final value of y_n cannot be obtained by further interpolations because we have not recorded intermediate values of y_n. We run the program again, using the initial values in the last line of the table as input data. Then we find that $y_n = .702948$, which is the desired result. We also find that $y_n' = -.414188$. Ideally the values of y_n'' and y_n''' should be zero, but some small nonzero values will be found because of the error of the numerical approximation and roundoff error.

Problems

Solve the differential Eqs. 5-1 through 5-12 numerically at several points for the specified initial conditions. (Analytical solutions are given to make it easy to check the numerical results.)

5-1. $y' + 2y = x^2$ $\quad y(0) = \frac{1}{4}$ $\quad y = \frac{x^2}{2} - \frac{x}{2} + \frac{1}{4}$

5-2. $y' + y = \sin x$ $\quad y(0) = -\frac{1}{2}$ $\quad y = \frac{1}{2}(\sin x - \cos x)$

5-3. $y' + 2xy = x$ $\quad y(0) = 1$ $\quad y = \frac{1}{2}(1 - e^{-x^2})$

5-4. $y' + y = xy^2$ $\quad y(0) = 1$ $\quad y = \frac{1}{x+1}$

5-5. $y' + x^2(y - 3y^3) = 0$ $\quad y(0) = \frac{1}{2}$ $\quad y = \frac{1}{(e^{2x^3/3} + 3)^{1/2}}$

5-6. $y' + y \tan x = \sin 2x$ $\quad y(0) = 0$ $\quad y = 2\cos x(1 - \cos x)$

5-7. $y' + (y^2 - 1)\tan x = 0$ $\quad y(0) = 3$ $\quad y = \frac{2 + \cos^2 x}{2 - \cos^2 x}$

5-8. $y' = \frac{y+1}{x+1}$ $\quad y(0) = 0$ $\quad y = x$

5-9. $y' = \frac{y^2 - 1}{x^2 - 1}$ $\quad y(2) = .8$ $\quad y = \frac{x+2}{2x+1}$

5-10. $xy' + y = xy$ $\quad y(1) = \frac{1}{2}$ $\quad y = \frac{e^{x-1}}{2x}$

5-11. $xy' + y = xy^3$ $\quad y(1) = \frac{1}{2}$ $\quad y = \frac{1}{\sqrt{2x(x+1)}}$

5-12. $(1 + x^2)y' + xy = xy^2$ $\quad y(0) = \frac{1}{2}$ $\quad y = \frac{1}{\sqrt{1 + x^2} + 1}$

5-13. Modify the Adams program of Sec. 5-1 to obtain an iterative solution of the corrector Eq. 5-7b as discussed under "Accuracy of Numerical Solutions" in Sec. 5-1. Repeat the Adams solution of Eq. 5-5 with two iterative cycles in the solution of Eq. 5-7b, and verify the numerical results shown below for $h = .1$. Compare these results with the ones given in the table of Sec. 5-1 for the basic Adams method. Observe that in this problem the iterative process yields no improvement in accuracy; the error in solving Eq. 5-7b approximately is of the same order as the inherent error in the equation itself.

x_n	.5	1.0	1.5	2.0
y_n	1.797443	3.436571	6.463401	11.778167

5-14. An analysis of the Adams method shows that the error of the corrected result is approximately $-19/251$ times the error of the predicted result. Therefore the equation

$$y_{i+1} = \frac{251}{270} y^C_{i+1} + \frac{19}{270} y^P_{i+1}$$

tends to balance the errors of the predictor and the corrector and give a result that is more accurate than either. (The superscripts P and C refer to the predictor and the corrector, respectively.) The value of the corrector must be an accurate solution of Eq. 5-7b. Revise the program of Prob. 5-13 to include this equation, and use the new program to repeat the solution of Eq. 5-5 and verify the results shown below for $h = .1$. Observe that the new results are better than the basic Adams results shown in the table of Sec. 5-1.

x_n	.5	1.0	1.5	2.0
y_n	1.797442	3.436563	6.463379	11.778116

Solve the differential Eqs. 5-15 through 5-22 numerically at several points for the specified initial conditions. (Analytical solutions are given to make it easy to check the numerical results.)

5-15. $y'' - 3y' + 2y = 0$ $\quad y(0) = 1$ $\quad y'(0) = 1$
$y = e^x$

5-16. $y'' + 4y' + 5y = 0$ $\quad y(0) = 0$ $\quad y'(0) = 1$
$y = e^{-2x} \sin x$

5-17. $y'' + 3y' + 2y = \cos x$ $\quad y(0) = .1$ $\quad y'(0) = .3$
$y = \frac{1}{10}(3 \sin x + \cos x)$

5-18. $y'' - xy' + 2y = 0 \qquad y(0) = -1 \quad y'(0) = 1$
$y = x^2 - 1$

5-19. $y'' + x^2y - 4xy = 0 \qquad y(0) = 0 \quad y'(0) = 4$
$y = x^4 + 4x$

5-20. $y'' - x^2y + xy = x \qquad y(0) = 1 \quad y'(0) = 1$
$y = x + 1$

5-21. $yy'' + y'^2 + 2y^2 = 0 \qquad y(0) = 1 \quad y'(0) = 0 \quad x < \frac{\pi}{4}$

$$y = \frac{1}{\sqrt{\cos 2x}}$$

5-22. $yy'' + y'^2 = 1 \qquad y(0) = 0 \quad y'(0) = 1$
$y = x$

5-23. Solve Eq. 5-16, with its associated initial conditions, by using the Adams program for second-order differential equations. Check the results at a few points against Eq. 5-17.

5-24. Obtain a numerical solution of Eq. 5-18 with its associated initial conditions by the first procedure suggested in the text. Check the results at a few points against Eq. 5-20.

5-25. Show that Eq. 5-19, with its associated initial conditions, follows from Eq. 5-18, with its associated initial conditions. Use this fact to obtain a numerical solution of Eq. 5-18. Check the results at a few points against Eq. 5-20.

5-26. Obtain a numerical solution of the differential equation

$$y'' + \frac{y'}{x} + y = 0$$

with the initial conditions

$$y(0) = 1 \qquad y'(0) = 0$$

It can be shown that the analytical solution is a Bessel function, $y = J_0(x)$. Use the first table of Sec. 3-6 to check the numerical results at a few points.

5-27. Obtain a numerical solution of the differential equation

$$y'' + \frac{y'}{x} + \left(1 - \frac{1}{x^2}\right) y = 0$$

with the initial conditions

$$y(0) = 0 \qquad y'(0) = \frac{1}{2}$$

It can be shown that the analytical solution is a Bessel function, $y = J_1(x)$. Use the first table of Sec. 3-6 to check the numerical results at a few points.

5-28. Obtain a numerical solution of the differential equation

$$y'' + \frac{y'}{x} + \frac{y}{1-x^2} = 0$$

with the initial conditions

$$y(0) = \frac{\pi}{2} \qquad y'(0) = 0$$

It can be shown that the analytical solution is an elliptic integral, $y = E(x)$. Use the table of Sec. 3-4 to check the numerical results at a few points.

Solve the differential Eqs. 5-29 through 5-32 numerically at several points for the specified initial conditions. (Analytical solutions are given to make it easy to check the numerical results.)

5-29. $y^{IV} + 3y'' - 4y = 0$ $\quad y(0) = 1 \quad y'(0) = 0$
$y''(0) = 1 \quad y'''(0) = -5 \quad y = e^{-x} + \dfrac{\sin 2x}{2}$

5-30. $y^{IV} - 2y''' + 2y'' - 2y' + y = 0 \quad y(0) = 1 \quad y'(0) = 1$
$y''(0) = -1 \quad y'''(0) = -1 \quad y = \sin x + \cos x$

5-31. $y^{IV} + 2y''' - 3y'' - 4y' + 4y = 0 \quad y(0) = 0 \quad y'(0) = 1$
$y''(0) = 2 \quad y'''(0) = 3 \quad y = xe^x$

5-32. $y^{IV} - 2y'' + y = 4 \cos x \quad y(0) = 2 \quad y'(0) = -1$
$y''(0) = 0 \quad y'''(0) = -1 \quad y = e^{-x} + \cos x$

5-33. Write a Runge-Kutta program to solve the four simultaneous first-order differential equations

$$y' = f_a(x,y,u,v,w) \qquad u' = f_b(x,y,u,v,w)$$
$$v' = f_c(x,y,u,v,w) \qquad w' = f_d(x,y,u,v,w)$$

Use the program to solve Eq. 5-24, and check the results against those given in the text.

6

Matrices and Simultaneous Equations

6-1. Simultaneous Linear Algebraic Equations

The solution of a set of simultaneous linear algebraic equations

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = c_1 \\ &a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = c_2 \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \ldots + a_{mn}x_n = c_3 \end{aligned} \tag{6-1}$$

is one of the most frequently occurring problems in applied mathematics. In this section we consider one very straightforward and reasonably efficient method of solution—the Gauss-Jordan method. The procedure will be introduced by an example. Consider the set of equations

$$5x_1 - 2x_2 + 3x_3 = -2 \tag{6-2a}$$

$$-2x_1 + 7x_2 + 5x_3 = 7 \tag{6-2b}$$

$$3x_1 + 5x_2 + 6x_3 = 9 \tag{6-2c}$$

We start by dividing through the first equation to make the coefficient of the leading term unity. We then add or subtract appropriate multiples

of the resulting equation to eliminate the leading terms of the other equations. These operations constitute the first cycle of the solution. At the end of the first cycle, we have

$$\begin{aligned} x_1 - .4x_2 + .6x_3 &= -.4 \\ 6.2x_2 + 6.2x_3 &= 6.2 \\ 6.2x_2 + 4.2x_3 &= 10.2 \end{aligned}$$

Throughout this chapter we use matrix notation. For the present, this simply amounts to writing the equations with detached coefficients; matrix algebra will not be introduced until the next section. We represent Eqs. 6-2 by the matrix

$$\left[\begin{array}{ccc:c} 5 & -2 & 3 & -2 \\ -2 & 7 & 5 & 7 \\ 3 & 5 & 6 & 9 \end{array}\right]$$

We have included both the square matrix of the as and the column matrix of the cs in a single matrix, separated by a dotted line. A matrix of this type is sometimes known as an *augmented matrix*. At the end of the first cycle, the matrix becomes

$$\left[\begin{array}{ccc:c} 1 & -.4 & .6 & -.4 \\ 0 & 6.2 & 6.2 & 6.2 \\ 0 & 6.2 & 4.2 & 10.2 \end{array}\right]$$

During the arithmetical operations, the first row is known as the pivotal row. The element that divides the row (5) is known as the pivotal element or the pivot.

Our objective is to reduce the square matrix to a diagonal matrix of unit elements. For the second cycle we use the second row as the pivotal row; that is, we normalize the second row by dividing through by the element in the second column and then use the resulting row to reduce the elements at the top and bottom of the second column to zero. The result is

$$\left[\begin{array}{ccc:c} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & -2 & 4 \end{array}\right]$$

For the third cycle we normalize the third row and use the result to clear the remaining elements of the third column. The result is

$$\left[\begin{array}{ccc:c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -2 \end{array}\right]$$

The problem is now solved. The last matrix represents the equations

$$x_1 = 2$$
$$x_2 = 3$$
$$x_3 = -2$$

which are the desired results.

We now consider the general problem of solving a set of n simultaneous linear algebraic equations in n unknowns. In other words, we shall write a program to solve Eqs. 6-1. We denote the general coefficients by a_{ij} and c_i. The subscripts i and j are the row number and column number, respectively, of the matrix element. The usual practice is to number the subscripts from 1 to n. However, most versions of BASIC allow only the upper limit to be dimensioned. Storage space is set aside for an array of numbers with subscripts running from 0 to the upper limit. If the subscripts actually start at 1, one row and one column of storage space are wasted. To make the most efficient possible use of the computer storage space, we shall run the subscripts from 0 to $n - 1$ in the program. We also number the cycles from $k = 0$ to $k = n - 1$. Also, for the sake of an efficient program, we shall denote the constants c_i as a_{in}. Since the same operations are performed on the cs as on the as, we then need only one set of instructions.

The program follows. Line 1 is the title. Line 10 assigns the value of n, and line 20 is a DIMension statement. Lines 30 through 130 read and print the values of the as and cs. The only lines that require any comment are 80 and 90. These generate a gap in each row between a_{in-1} and $a_{in} = c_i$. Thus the A and C matrices are separated in the display or printout. Lines 140 through 220 constitute a loop that solves the set of simultaneous equations. This segment is the core of the program; everything else is concerned with the organization of the data and the printout. The operation of this segment may be made a little more clear by imagining the following lines to be inserted:

```
162  NEXT J
164  FOR J=N TO K STEP -1
```

Lines 150 through 162 now constitute a loop that normalizes the pivotal row $i = k$. The calculations run from right to left so that the pivotal element is not reduced to unity until the last step; otherwise the other divisions would be meaningless. Lines 164 through 210 constitute a nested loop that adjusts the other rows to set the elements in the column $j = k$ equal to zero. The entire operation is repeated by lines 140 and 220 for the required number of cycles. The loops of lines 150 through 162 and of lines 164 through 210 may be combined by deleting lines 162 and 164, and this has been done in the program. Lines 230 through 270 print the resulting values of the xs. Lines 280 through 300 are the data lines; these contain the coefficients for Eqs. 6-2.

```
  1  REM: SIMULTANEOUS EQUATIONS
 10  N=3                                  Assigns value of n.
 20  DIM A(N-1,N)                         DIMension statement.
 30  PRINT                                }
 40  PRINT "THE A AND                     }
     C MATRICES ARE:"                     }
 50  FOR I=0 TO N-1                       }
 60  FOR J=0 TO N                         }
 70  READ A(I,J)                          }  Reads and prints
 80  IF J<N THEN 100                      }  matrix elements.
 90  PRINT "  ";                          }
100  PRINT A(I,J);                        }
110  NEXT J                               }
120  PRINT                                }
130  NEXT I                               }
140  FOR K=0 TO N-1                       }
150  FOR J=N TO K STEP -1                 }
160  A(K,J)=A(K,J)/A(K,K)                 }
170  FOR I=0 TO N-1                       }  Solves simultaneous
180  IF I=K THEN 200                      }  equations.
190  A(I,J)=A(I,J)-A(K,J)*A(I,K)          }
200  NEXT I                               }
210  NEXT J                               }
220  NEXT K                               }
230  PRINT                                }
240  PRINT "THE X MATRIX IS:"             }
250  FOR I=0 TO N-1                       }  Prints results.
260  PRINT A(I,N)                         }
270  NEXT I                               }
280  DATA 5,-2,3,-2                       }
290  DATA -2,7,5,7                        }  Data lines.
300  DATA 3,5,6,9                         }
```

To operate the program, line 10 is filled in by the user. The data lines at the end of the program are also filled in by the user. For clarity, we have used one data line for each row of coefficients in the equations. These may be combined if the user prefers. This program runs on almost any commonly used model of microcomputer as it stands, with the possible exception of two lines. Some computers do not allow a variable to be used as the argument in a DIMension statement. With any computer of this type, numbers must be used inside the parentheses in line 20. The most convenient way to do this is to insert some large numbers and leave them permanently, instead of inserting the true values of $n - 1$ and n each time the program is run. (Another possibility is to simply omit the DIMension statement; the computer automatically allows enough space for a set of ten equations.) Also, some computers such as the Apple do not automatically generate leading and trailing spaces with numerical out-

put. For any computer of this type, the expression " "; must be appended to line 100.

When the program is run, the display appears as follows:

```
THE A AND C MATRICES ARE:
 5  -2   3    -2
-2   7   5     7
 3   5   6     9

THE X MATRIX IS:
 2
 3
-2
```

With larger matrices, each row may occupy more than one line on the screen or printout. In this case it may be desirable for clarity to print blank spaces between rows. This can be accomplished by inserting PRINT statements as lines 55 and 255.

It is possible to simplify the input slightly. Inspection of the A matrix shows that the coefficients satisfy the relation $a_{ij} = a_{ji}$. The matrix is said to be symmetric, and it may be written in the form

$$\begin{bmatrix} 5 & -2 & 3 \\ -2 & 7 & 5 \\ 3 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 5 & -2 & 3 \\ & 7 & 5 \\ \text{Sym.} & & 6 \end{bmatrix}$$

The vast majority of matrices that occur in the solution of physical problems are symmetric. Therefore it is seldom necessary to enter a full set of coefficients in the data lines. The program may be amended as follows:

```
 62  IF J<I THEN 74
 72  GOTO 80
 74  A(I,J)=A(J,I)
280  DATA 5,-2,3,-2
290  DATA 7,5,7
300  DATA 6,9
```

The amended program bypasses the READ statement for elements below the diagonal and obtains values from symmetry. The data line 280 is identical to the original, but lines 290 and 300 are shorter.

As a second example, consider the equations

$$x_1 + 2x_2 + x_3 = 9 \tag{6-3a}$$

$$2x_1 + 4x_2 + 3x_3 = 16 \tag{6-3b}$$

$$x_1 + 3x_2 + 6x_3 = 3 \tag{6-3c}$$

We start with the matrix

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 9 \\ 2 & 4 & 3 & 16 \\ 1 & 3 & 6 & 3 \end{array}\right]$$

The first cycle leads to

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 9 \\ 0 & 0 & 1 & -2 \\ 0 & 1 & 5 & -6 \end{array}\right]$$

It is apparent that the present procedure cannot be continued without some modification. The pivotal element for the second row is zero. An attempt to normalize this row would require a division by zero. We can easily get around the difficulty by interchanging the last two rows. Thus

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 9 \\ 0 & 1 & 5 & -6 \\ 0 & 0 & 1 & -2 \end{array}\right]$$

The second cycle now leads to

$$\left[\begin{array}{ccc|c} 1 & 0 & -9 & 21 \\ 0 & 1 & 5 & -6 \\ 0 & 0 & 1 & -2 \end{array}\right]$$

After the third cycle, we have

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & -2 \end{array}\right]$$

The results are

$x_1 = 3$
$x_2 = 4$
$x_3 = -2$

We now amend Eqs. 6-3 to read:

$$x_1 + 2x_2 + x_3 = 9 \tag{6-4a}$$

$$2x_1 + 4x_2 + 3x_3 = 16 \tag{6-4b}$$

$$x_1 + 2x_2 + 2x_3 = 7 \tag{6-4c}$$

We start with the matrix

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 9 \\ 2 & 4 & 3 & 16 \\ 1 & 2 & 2 & 7 \end{array}\right]$$

The first cycle leads to

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 9 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 1 & -2 \end{array}\right]$$

It is impossible to continue, because both of the available pivotal elements in the second column are zero. This means that the equations are not linearly independent. We can easily see that Eq. 6-4b is the sum of Eqs. 6-4a and 6-4c. No solution (or at least no unique solution) can be found. It is clear that $x_3 = -2$, but x_1 and x_2 are indeterminate. It can be shown that this situation occurs if, and only if, the determinant of the matrix is equal to zero. A matrix that has this property is said to be singular.

The difficulties in the solutions of Eqs. 6-3 and 6-4 occurred when zero elements appeared on the diagonal in pivotal positions. In most physical applications this difficulty does not occur. The occurrence of a singular matrix such as that of Eqs. 6-4 in the solution to a physical problem means that there is some defect in the formulation of the solution. However, there are a few real problems in which the situation of Eqs. 6-3 can occur. A segment follows that may be inserted into the earlier program to allow for zero elements on the diagonal. At the same time that we allow for zero elements on the diagonal, the accuracy of the calculation can be enhanced slightly by adopting a broader viewpoint. The presence of any very small number in a pivotal position—even if not identically zero—causes a loss of accuracy in the calculations. The optimum result is obtained by shuffling the rows in each cycle in order to bring the available element with the greatest absolute value into the pivotal position.* The following segment does this.

```
141  Z=K
142  Y=ABS(A(K,K))
143  IF K=N-1 THEN 149
144  FOR I=K+1 TO N-1
145  IF ABS(A(I,K))<=Y THEN 148
146  Z=I
147  Y=ABS(A(I,K))
148  NEXT I
```

Finds largest pivot.

* For completeness, we point out that this rule is not quite rigorous. The choice of pivot can be changed arbitrarily by multiplying through one of the equations by some large number, even though this makes no fundamental change in the problem. Nevertheless, the present rule gives good results in practical problems. More comprehensive discussions of pivoting strategies can be found in books on numerical analysis, such as reference 10.

```
149 IF Y>10^-7 THEN 153
150 PRINT
151 PRINT "THE A MATRIX IS
    SINGULAR."
152 END
153 IF Z=K THEN 159
154 FOR J=K TO N
155 T=A(K,J)
156 A(K,J)=A(Z,J)
157 A(Z,J)=T
158 NEXT J
159 FOR J=N TO K STEP-1
```

Terminates calculation if matrix is singular.

Rearranges rows.

Formerly line 150.

The constant in line 149 must be adjusted to fit the accuracy of the computer. This allows for the fact that a calculated result that should be exactly zero may have some small nonzero value due to machine error. We adopt the same rule here as in Sec. 2-5: Any result that differs from zero only in the last two digits of the calculation is assumed to be exactly zero. The constant 10^{-7} is suitable for typical microcomputer accuracy of nine digits. For the TRS-80, with seven-digit accuracy, the constant should be 10^{-5}. For the TI-99/4, with thirteen-digit accuracy, a value of 10^{-11} may be used.

With the data of Eqs. 6-2, the amended program gives the same results as the original program. For the data of Eqs. 6-3, we revise the data lines to read

```
280 DATA 1,2,1,9
290 DATA 2,4,3,16
300 DATA 1,3,6,3
```

The results are the same as those found previously by carrying out the algebra: $x_1 = 3$, $x_2 = 4$, $x_3 = -2$. With the data of Eqs. 6-4, the program prints

```
THE A MATRIX IS SINGULAR.
```

Systems of equations of the type

$$\left.\begin{aligned} d_1x_1 + e_1x_2 &= c_1 \\ a_2x_1 + d_2x_2 + e_2x_3 &= c_2 \\ a_3x_2 + d_3x_3 + e_3x_4 &= c_3 \\ a_4x_3 + d_4x_4 + e_4x_5 &= c_4 \\ a_5x_4 + d_5x_5 &= c_5 \end{aligned}\right\} \tag{6-5}$$

occur in a number of physical applications. The coefficient matrix contains only three diagonals of nonzero elements. Equations of this type are said to have a tridiagonal matrix. We could, of course, use the Gauss-Jordan program given earlier, but this procedure would be very inefficient because,

for a large system of equations, most of the memory and most of the arithmetical operations would be devoted to zero elements. We shall use a different procedure for which the new notation is particularly suitable.

We start by using the last equation to eliminate the last term on the left side of the next-to-last equation. We then work our way upward through the set, applying the same process to each equation in turn. This leads to the following set of equations:

$$\left.\begin{aligned} d_1'x_1 &= c_1' \\ a_2x_1 + d_2'x_2 &= c_2' \\ a_3x_2 + d_3'x_3 &= c_3' \\ a_4x_3 + d_4'x_4 &= c_4' \\ a_5x_4 + d_5'x_5 &= c_5' \end{aligned}\right\} \tag{6-6}$$

The new parameters d_i' and c_i' are given by the equations

$$d_i' = d_i - \frac{e_i a_{i+1}}{d_{i+1}'}$$

$$c_i' = c_i - \frac{e_i c_{i+1}}{d_{i+1}'}$$

The value of x_1 is now found directly from the first equation of the set 6-6. We then substitute this result into the second equation and solve for x_2. In the same way, we work our way downward through the set. Each result x_i is found with the help of the last prior result x_{i-1}. The results are printed as they are found. The general formula is

$$x_i = \frac{c_i' - a_i x_{i-1}}{d_i'}$$

Sets of equations like Eqs. 6-5 that originate in physical problems almost invariably have symmetric matrices, that is, $e_i = a_{i+1}$. Therefore we do not need to enter both the as and the es. We shall work with the as. Then the three foregoing equations become

$$d_i' = d_i - \frac{a_{i+1}^2}{d_{i+1}'}$$

$$c_i' = c_i - \frac{a_{i+1} c_{i+1}}{d_{i+1}'}$$

$$x_i = \frac{c_i' - a_i x_{i-1}}{d_i'}$$

A program follows. It is set up to evaluate Eqs. 6-5 with

$d_1 = d_2 = d_3 = d_4 = d_5 = 2$
$a_2 = a_3 = a_4 = a_5 = -1$
$c_1 = 6,\ c_2 = c_3 = c_4 = c_5 = 0$

Line 1 is the title. Line 10 assigns the value of n, the number of equations. Lines 20 through 40 are DIMension statements. To take advantage of the full capacity of the computer, we again run the subscripts from 0 to $n - 1$. The loops of lines 50 through 70, 80 through 100, and 110 through 130 read the ds, as, and cs. The loop of lines 140 through 170 transforms Eqs. 6-5 into Eqs. 6-6. The loop of lines 180 through 210 evaluates the xs and prints the results as they are found. Lines 220, 230, and 240 are the data lines for the ds, as (or es), and cs.

```
  1 REM: SIMULTANEOUS EQUATIONS WITH TRIDIAGONAL
    MATRIX
 10 N=5
 20 DIM D(N-1)                          } DIMension statements.
 30 DIM A(N-1)
 40 DIM C(N-1)
 50 FOR I=0 TO N-1                      } Reads ds.
 60 READ D(I)
 70 NEXT I
 80 FOR I=1 TO N-1                      } Reads as.
 90 READ A(I)
100 NEXT I
110 FOR I=0 TO N-1                      } Reads cs.
120 READ C(I)
130 NEXT I
140 FOR I=N-2 TO 0 STEP -1              } Calculates coefficients
150 D(I)=D(I)-A(I+1)*A(I+1)/D(I+1)      } of Eqs. 6-6.
160 C(I)=C(I)-A(I+1)*C(I+1)/D(I+1)
170 NEXT I
180 FOR I=0 TO N-1                      } Calculates and prints xs.
190 X=(C(I)-A(I)*X)/D(I)
200 PRINT X
210 NEXT I
220 DATA 2,2,2,2,2                      } Data lines.
230 DATA -1,-1,-1,-1
240 DATA 6,0,0,0,0
```

The results are

$x_1 = 5 \quad x_2 = 4 \quad x_3 = 3 \quad x_4 = 2 \quad x_5 = 1$

To operate the program, line 10 is filled in by the user. The data lines at the end of the program are also filled in by the user. These are organized differently from the data lines in the earlier program; each data line repre-

sents the coefficients on a diagonal instead of a row. Lines 220 and 240 for the *d*s and *c*s have n entries each. Line 230 for the *a*s (or *e*s) has $n - 1$ entries. The same remark about the DIMension statements that followed the earlier program applies here also. This program does not print the original matrices. However, the coefficients can easily be obtained by listing the data lines.

The coefficients in systems of equations of this type usually follow a repetitive pattern. For a very large set of equations it is sometimes more convenient to generate the coefficients by equations instead of filling in a large amount of data. To do this for the present problem, we delete lines 60 through 130, then insert the following lines:

```
 60 D(I)=2
 70 A(I)=-1
 80 C(I)=0
 90 NEXT I
100 C(0)=6
110 A(0)=0
```

The data lines may also be deleted. The results are identical to those given by the original program.

6-2. Matrix Algebra

This section is concerned with elementary matrix algebra. Most readers are probably familiar with elementary matrix nomenclature and operations, and we have used a few simple matrix representations in Sec. 6-1. However, we shall give a brief outline of the essential points. A matrix is a rectangular array of elements arranged in rows and columns, usually enclosed in brackets. In this book a matrix is denoted by a capital letter. The elements are denoted by the same letter in lower case, with subscripts to indicate the row and column. Thus

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

is a matrix. The general element is a_{ij}, where i is the row number and j is the column number. A matrix with m rows and n columns is known as an $m \times n$ *matrix*. Practical applications are usually concerned with square matrices ($m = n$), column matrices ($n = 1$), and row matrices ($m = 1$). A square matrix is said to be symmetric if and only if all of the elements satisfy the equation $a_{ij} = a_{ji}$. The group of elements $i = j$ of a square matrix is known as the *diagonal* or the *principal diagonal*. A square matrix in which all off-diagonal elements are equal to zero is

known as a *diagonal matrix*. A diagonal matrix with all diagonal elements equal to one is known as a *unit matrix* or an *identity matrix*. It is usually denoted by the symbol I.

The statement that two matrices are equal, that is,

$$A = B$$

means that they have the same number of rows, the same number of columns, and that corresponding elements are equal, that is, all elements satisfy the equation

$$a_{ij} = b_{ij}$$

Two matrices may be added provided that they have the same number of rows and the same number of columns. Each element of the resulting matrix is the sum of the corresponding elements of the original matrices. Thus the statement that

$$S = A + B$$

means that all elements satisfy the equation

$$s_{ij} = a_{ij} + b_{ij}$$

It is clear that matrix addition is *commutative,* that is,

$$A + B = B + A$$

and that it is *associative,* that is,

$$(A + B) + C = A + (B + C)$$

Two matrices may be multiplied, provided that the number of columns of the first is equal to the number of rows of the second. (Two matrices that satisfy this requirement are said to be *conformable* in the order given.) The product has the same number of rows as the first matrix and the same number of columns as the second. Let A be an $m \times n$ matrix and B be an $n \times q$ matrix. Then the product

$$P = AB$$

is an $m \times q$ matrix. We say that B is premultiplied by A or that A is postmultiplied by B. The elements of P are given by the equation

$$p_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

For example

$$\begin{bmatrix} 5 & -2 & 3 \\ -2 & 7 & 5 \\ 3 & 5 & 6 \end{bmatrix}\begin{bmatrix} 3 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 14 & 8 & 10 \\ 13 & 34 & 27 \\ 25 & 38 & 31 \end{bmatrix} \tag{6-7}$$

Matrix multiplication is not generally commutative. In fact, it is not possible to carry out both multiplications AB and BA unless the matrices are conformable in either order. Even in this case, the products AB and BA are not usually equal. For example,

$$\begin{bmatrix} 3 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 3 \end{bmatrix}\begin{bmatrix} 5 & -2 & 3 \\ -2 & 7 & 5 \\ 3 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 14 & 13 & 25 \\ 8 & 34 & 38 \\ 10 & 27 & 31 \end{bmatrix} \tag{6-8}$$

which is not the same as the preceding result. It can also be seen that the product of two symmetric square matrices is in general not symmetric.

We state two other results that will be needed subsequently. Matrix multiplication is associative, that is,

$$(AB)C = A(BC)$$

If any matrix is premultiplied or postmultiplied by the conformable unit matrix, the result is identical to the original matrix. Thus

$$IA = A \qquad AI = A \tag{6-9a,b}$$

However, the I matrices in these equations are not the same. They are of different sizes unless the A matrix is square. If the A matrix is square, the Is are the same, and this case is an exception to the general rule that matrix multiplication is not commutative.

We will need another matrix operation: *transposition*. The transpose of a matrix is the matrix obtained by interchanging its rows and columns, and is denoted by the superscript T. Thus, for example

$$\begin{bmatrix} 2 & 7 \\ 5 & 3 \\ 4 & 1 \end{bmatrix}^T = \begin{bmatrix} 2 & 5 & 4 \\ 7 & 3 & 1 \end{bmatrix}$$

It is clear that the transpose of a transpose is the original matrix.

In general, let A be an $m \times n$ matrix. Then

$$B = A^T$$

is an $n \times m$ matrix, and the elements of the two matrices are related by the equation

It follows from the definition of matrix multiplication that the transpose of a product is equal to the product of the transposes in reverse order, that is,

$$(AB)^T = B^T A^T$$

It is clear that the transpose of a symmetric square matrix is identical to the original matrix. Hence if A and B are symmetric square matrices, we have

$$(AB)^T = BA$$

Equations 6-7 and 6-8 provide an example of this result.

Many large computers have built-in programs for the most important matrix operations. However, few if any microcomputers have this feature, so we shall develop several programs. The one that follows evaluates the matrix product AB, where A is an $m \times n$ matrix and B is an $n \times q$ matrix. Line 1 is the title. Lines 10, 20, and 30 assign the values of the matrix dimensions m, n, and q. Lines 40 through 60 are DIMension statements. To save space in the computer memory, we run the subscripts from 0 to $m - 1$, $n - 1$, and $q - 1$. Lines 70 through 150 read and print the values of the as. Lines 160 through 240 do the same thing for the bs. Lines 250 through 360 calculate and print the elements of the product matrix. The data lines at the end contain the values of the as and bs for Eq. 6-7.

```
  1  REM: MATRIX MULTIPLICATION
 10  M=3                                  } Assigns matrix dimensions.
 20  N=3
 30  Q=3
 40  DIM A(M-1,N-1)                       } DIMension statements.
 50  DIM B(N-1,Q-1)
 60  DIM P(M-1,Q-1)
 70  PRINT                                } Reads and prints as.
 80  PRINT "THE A MATRIX IS:"
 90  FOR I=0 TO M-1
100  FOR J=0 TO N-1
110  READ A(I,J)
120  PRINT A(I,J);
130  NEXT J
140  PRINT
150  NEXT I
```

```
160 PRINT
170 PRINT "THE B MATRIX IS:"
180 FOR I=0 TO N-1
190 FOR J=0 TO Q-1
200 READ B(I,J)
210 PRINT B(I,J);
220 NEXT J
230 PRINT
240 NEXT I
250 PRINT
260 PRINT "THE PRODUCT AB IS:"
270 FOR I=0 TO M-1
280 FOR J=0 TO Q-1
290 P(I,J)=0
300 FOR K=0 TO N-1
310 P(I,J)=P(I,J)+A(I,K)*B(K,J)
320 NEXT K
330 PRINT P(I,J);
340 NEXT J
350 PRINT
360 NEXT I
370 DATA 5,-2,3
380 DATA -2,7,5
390 DATA 3,5,6
400 DATA 3,2,1
410 DATA 2,4,2
420 DATA 1,2,3
```

Reads and prints *b*s.

Calculates and prints product matrix.

Data lines for *a*s.

Data lines for *b*s.

To operate the program, lines 10, 20, and 30 are filled in by the user. The data lines at the end of the program are also filled in by the user. We have used one data line for each row of the A matrix and again for each row of the B matrix. It is not feasible to combine the data for the two matrices row by row as in Sect. 6-1, because the two matrices do not necessarily have the same number of rows. However, other condensed formats are possible; we might put all of the data for matrix A in one line and all of the data for matrix B in another line. The program verifies Eq. 6-7.

We repeat three remarks that followed the first program of Sec. 6-1. With some computers it is necessary to use numbers instead of algebraic expressions in the DIMension statements of lines 40, 50, and 60. Also, with computers that do not automatically print leading and trailing spaces with numeric output, the expression " "; must be appended to lines 120, 210, and 330. Finally, with larger matrices, each row may occupy more than one line on the screen or printout. In this case it may be desirable for clarity to print blank spaces between rows. This can be accomplished by inserting PRINT statements as lines 95, 185, and 275.

There is no operation of matrix division. However, for a nonsingular square matrix (Sec. 6-1), there is a somewhat similar operation known as *inversion*. The inverse of a square matrix A is known as A^{-1}, and is defined by the equation

$$AA^{-1} = I \tag{6-10a}$$

It can be shown that matrix multiplication is commutative in this special case; it is also true that

$$A^{-1}A = I \tag{6-10b}$$

or that A is the inverse of A^{-1}.

The same Gauss-Jordan procedure that we used in Sec. 6-1 to solve simultaneous equations can also be used to invert a matrix. Let us invert

$$A = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 8 & 5 \\ 2 & 5 & 4 \end{bmatrix} \tag{6-11}$$

We start by writing the augmented matrix

$$\left[\begin{array}{ccc|ccc} 1 & 3 & 2 & 1 & 0 & 0 \\ 3 & 8 & 5 & 0 & 1 & 0 \\ 2 & 5 & 4 & 0 & 0 & 1 \end{array}\right]$$

The left half is the original matrix A; the right half is the unit matrix I. We now apply the Gauss-Jordan process, performing the same operations on the entire augmented matrix. For the first cycle, we use the first row as the pivotal row. The result is

$$\left[\begin{array}{ccc|ccc} 1 & 3 & 2 & 1 & 0 & 0 \\ 0 & -1 & -1 & -3 & 1 & 0 \\ 0 & -1 & 0 & -2 & 0 & 1 \end{array}\right]$$

For the second cycle, we use the second row as the pivotal row. The result is

$$\left[\begin{array}{ccc|ccc} 1 & 0 & -1 & -8 & 3 & 0 \\ 0 & 1 & 1 & 3 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 & 1 \end{array}\right]$$

Finally, after the third cycle

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & -7 & 2 & 1 \\ 0 & 1 & 0 & 2 & 0 & -1 \\ 0 & 0 & 1 & 1 & -1 & 1 \end{array}\right]$$

The left half is now the unit matrix, and the right half is the inverse matrix A^{-1}. Thus the final result is

$$A^{-1} = \left[\begin{array}{ccc} -7 & 2 & 1 \\ 2 & 0 & -1 \\ 1 & -1 & 1 \end{array}\right] \tag{6-12}$$

This result can easily be verified by multiplying the matrices A and A^{-1}, in either order. It can be proved that the method is valid in general, but we shall confine ourselves to a simple heuristic observation. The effect of the Gauss-Jordan operations on the A matrix is that of a premultiplication by A^{-1}, and the same effect may be expected on the I matrix. Therefore the result is A^{-1}.

A critical review of the foregoing process shows that it is extremely inefficient. We have carried $2n = 6$ columns of the augmented matrix through the entire calculation, but only $n + 1 = 4$ columns are ever used at one time. In the first cycle we use only the first four columns, in the second cycle we use only the middle four, and in the third cycle we use only the last four. The remaining $n - 1 = 2$ columns take up useless storage space and running time. We now give an improved procedure, again starting with Eq. 6-11. To form the starting augmented matrix, we use only the original matrix plus one column of the unit matrix, that is,

$$\left[\begin{array}{ccc|c} 1 & 3 & 2 & 1 \\ 3 & 8 & 5 & 0 \\ 2 & 5 & 4 & 0 \end{array}\right]$$

We now use the Gauss-Jordan process to carry out the first cycle, using the first row as the pivotal row. The result is

$$\left[\begin{array}{c|ccc} 1 & 3 & 2 & 1 \\ 0 & -1 & -1 & -3 \\ 0 & -1 & 0 & -2 \end{array}\right]$$

We have shifted the separating line to show that it is now the last three columns that are of interest. To start the second cycle, we discard the first column and add the second column of the unit matrix. Thus

$$\left[\begin{array}{ccc|c} 3 & 2 & 1 & 0 \\ -1 & -1 & -3 & 1 \\ -1 & 0 & -2 & 0 \end{array}\right]$$

We again apply the Gauss-Jordan process, now using the second row as the pivotal row. The result is

$$\left[\begin{array}{c|ccc} 0 & -1 & -8 & 3 \\ 1 & 1 & 3 & -1 \\ 0 & 1 & 1 & -1 \end{array}\right]$$

By this time the procedure is obvious. We start the third cycle with the matrix

$$\left[\begin{array}{ccc|c} -1 & -8 & 3 & 0 \\ 1 & 3 & -1 & 0 \\ 1 & 1 & -1 & 1 \end{array}\right]$$

and finish with the matrix

$$\left[\begin{array}{c|ccc} 0 & -7 & 2 & 1 \\ 0 & 2 & 0 & -1 \\ 1 & 1 & -1 & 1 \end{array}\right]$$

Finally we discard the first column and obtain a result that is identical to Eq. 6-12.

A program for matrix inversion follows. It is very similar to the first program for simultaneous equations in Sec. 6-1. Line 1 is the title. Line 10 assigns the value of n, and line 20 is a DIMension statement. Lines 30 through 110 read and print the values of the as. Lines 120 through 340 perform the inversion and print the result. The k loop of lines 140 through 340 carries out the n cycles. The little i loop of lines 150 through 170 generates the last column of the augmented matrix to start each cycle, using a relational expression. The j loop of lines 180 through 240 carries out one cycle of the inversion process; this is very similar to the corresponding loop of the earlier program for simultaneous equations. However, we now start the operations with the second column $j = 1$, since the first column $j = 0$ will be discarded to start the next cycle. The loop of lines 250 through 330 shifts all the elements one step to the left at the end of each cycle. It also prints the results after the last cycle $k = n$. The data lines at the end contain the elements of the matrix. Each line represents one row of the matrix.

```
  1 REM: MATRIX INVERSION
 10 N=3                                  Assigns value of n.
 20 DIM A(N-1,N)                         DIMension statement.
 30 PRINT                                }
 40 PRINT "THE ORIGINAL                  }
    MATRIX IS:"                          }
 50 FOR I=0 TO N-1                       } Reads and prints
 60 FOR J=0 TO N-1                       } elements of
 70 READ A(I,J)                          } original matrix.
 80 PRINT A(I,J);                        }
 90 NEXT J                               }
100 PRINT                                }
110 NEXT I                               }
120 PRINT                                                          }
130 PRINT "THE INVERTED                                            }
    MATRIX IS:"                                                    }
140 FOR K=0 TO N-1                                                 }
150 FOR I=0 TO N-1                       } Generates               }
160 A(I,N)=ABS(I=K)                      } last column             }
170 NEXT I                               } of augmented            }
180 FOR J=1 TO N                         } matrix.                 }
190 A(K,J)=A(K,J)/A(K,0)                                           }
200 FOR I=0 TO N-1                       } One cycle               }
210 IF I=K THEN 230                      } of inversion.           }
220 A(I,J)=A(I,J)-A(K,J)*A(I,0)                                    } Inverts
230 NEXT I                                                         } matrix.
240 NEXT J                                                         }
250 FOR I=0 TO N-1                       }                         }
260 FOR J=0 TO N-1                       }                         }
270 A(I,J)=A(I,J+1)                      } Shifts                  }
280 IF K<N-1 THEN 300                    } numbers                 }
290 PRINT A(I,J);                        } to left;                }
300 NEXT J                               } prints                  }
310 IF K<N-1 THEN 330                    } results.                }
320 PRINT                                }                         }
330 NEXT I                               }                         }
340 NEXT K                                                         }
350 DATA 1,3,2                           } Data lines              }
360 DATA 3,8,5                           } for matrix              }
370 DATA 2,5,4                           } elements.               }
```

To operate the program, line 10 is filled in by the user. The data lines at the end of the program are also filled in by the user. We have used the matrix of Eq. 6-11 as an example. Each data line represents one row of the matrix. The result verifies Eq. 6-12. If spaces are needed between the elements in the display, the expression " "; may be appended

to lines 80 and 290. If spaces are desired between the rows, PRINT statements may be inserted as lines 55, 135, and 325.

The accuracy of a matrix inversion depends on the accuracy of the computer. An inversion can be checked very easily by reinverting the result and comparing it with the original matrix. With most computers this can be done manually after the program is run by pressing the ENTER key, then entering GOTO 140. As an alternative, the following lines may be incorporated into the program:

```
350  R=R+1
360  IF R>1 THEN 400
370  PRINT
380  PRINT "THE REINVERTED MATRIX IS:"
390  GOTO 140
400  END
```

The data lines must, of course, be renumbered.

The inversion of a matrix is seldom of interest in itself; it is usually performed as an intermediate step in obtaining some other result. We return to the simultaneous Eqs. 6-1. These can be written in matrix form as

$$AX = C \tag{6-13}$$

and we wish to solve for X. We premultiply both sides by A^{-1}. This leads to

$$A^{-1}(AX) = A^{-1}C$$

Since matrix multiplication is associative, the expression on the left side may be rewritten as $(A^{-1}A)X$. By Eq. 6-10b, this becomes IX, and, by Eq. 6-9a, it becomes X. It follows that

$$X = A^{-1}C \tag{6-14}$$

This furnishes an alternative to the solution that we developed in Sec. 6-1. We can obtain X by inverting A and then postmultiplying the result by C. At first glance this procedure seems pointless; it is longer and more complicated than the one that we have already used successfully. Nevertheless, it is sometimes highly advantageous. In many applications, a single A matrix is used repeatedly with a number of C matrices. The most efficient procedure for a problem of this type is to start by inverting the A matrix; the desired result for each C matrix is then obtained by multiplication, which takes very little computer time.

A program follows for the solution of simultaneous equations by matrix inversion. Line 1 is the title. Line 10 assigns the value of n, and the following three lines are DIMension statements. Lines 50 through 130 read and print the A matrix. Lines 140 through 300 invert it. The C matrix is entered by the INPUT statement in lines 310 through 360. (Line 330 requires a comment. The subscripts in the program start at 0, but, as far as the user is concerned, they start at 1.) The product $A^{-1}\,C$ is evaluated in lines 370 through 440, which also print the C and X matrices. Line 450 sends the execution back to the INPUT statement to call for a new set of cs. The last few lines are DATA statements; these represent the rows of the A matrix of Eqs. 6-2.

```
  1 REM: SIMULTANEOUS EQUATIONS (MATRIX IN-
    VERSION)
 10 N=3                                  Assigns value of n.
 20 DIM A(N-1,N)                        ┐
 30 DIM C(N-1)                          ├ DIMension statements.
 40 DIM X(N-1)                          ┘
 50 PRINT                               ┐
 60 PRINT "THE A MATRIX IS:"            │
 70 FOR I=0 TO N-1                      │
 80 FOR J=0 TO N-1                      │
 90 READ A(I,J)                         ├ Reads and prints as.
100 PRINT A(I,J);                       │
110 NEXT J                              │
120 PRINT                               │
130 NEXT I                              ┘
140 FOR K=0 TO N-1                      ┐
150 FOR I=0 TO N-1                      │
160 A(I,N)=ABS(I=K)                     │
170 NEXT I                              │
180 FOR J=1 TO N                        │
190 A(K,J)=A(K,J)/A(K,0)                │
200 FOR I=0 TO N-1                      │
210 IF I=K THEN 230                     │
220 A(I,J)=A(I,J)-A(K,J)*A(I,0)         ├ Inverts A matrix.
230 NEXT I                              │
240 NEXT J                              │
250 FOR I=0 TO N-1                      │
260 FOR J=0 TO N-1                      │
270 A(I,J)=A(I,J+1)                     │
280 NEXT J                              │
290 NEXT I                              │
300 NEXT K                              ┘
```

```
310 PRINT
320 FOR I=0 TO N-1
330 PRINT "I=";I+1
340 INPUT "ENTER C(I) ";C(I)
350 PRINT
360 NEXT I
```

Calls for values of cs.

```
370 PRINT "THE C AND X
    MATRICES ARE:"
380 FOR I=0 TO N-1
390 X(I)=0
400 FOR J=0 TO N-1
410 X(I)=X(I)+A(I,J)*C(J)
420 NEXT J
430 PRINT C(I),X(I)
440 NEXT I
```

Calculates product matrix $X=A^{-1}C$; prints C and X matrices.

```
450 GOTO 310
```

Returns for new input.

```
460 DATA 5,-2,3
470 DATA -2,7,5
480 DATA 3,5,6
```

Data lines for as.

To operate the program, line 10 is filled in by the user. The data lines at the end of the program are also filled by the user. These represent the as of Eq. 6-2. The cs are entered as input; $c_1 = -2$, $c_2 = 7$, $c_3 = 9$. The results are the same as those found in Sec. 6-1: $x_1 = 2$, $x_2 = 3$, $x_3 = -2$. If spaces are needed between the elements in the display, the expression " "; may be appended to line 100. If spaces are desired between the rows, PRINT statements may be inserted as lines 75 and 385.

6-3. Determinants

Determinants are not used as frequently as matrices; the methods of Secs. 6-1 and 6-2 for solving simultaneous equations are more efficient than methods based on determinants. However, it occasionally happens that a determinant must be evaluated. The best procedures are similar to that given in Sec. 6-1 for solving simultaneous equations, and we shall use some of the same examples. It is assumed that the reader is familiar with the elementary properties of determinants.

We start with the determinant

$$D = \begin{vmatrix} 5 & -2 & 3 \\ -2 & 7 & 5 \\ 3 & 5 & 6 \end{vmatrix} \tag{6-15}$$

The evaluation is accomplished by reducing all the elements below the diagonal to zero. To do so, we use the fact that any row—or any multiple

of any row—may be added to any other row without changing the value of the determinant. Using the first row as the pivotal row, we find that

$$D = \begin{vmatrix} 5 & -2 & 3 \\ 0 & 6.2 & 6.2 \\ 0 & 6.2 & 4.2 \end{vmatrix}$$

Now using the second row as the pivotal row, we obtain the result

$$D = \begin{vmatrix} 5 & -2 & 3 \\ 0 & 6.2 & 6.2 \\ 0 & 0 & -2 \end{vmatrix}$$

The value of this determinant is the product of the diagonal elements. To show this we expand by minors as follows:

$$D = 5 \begin{vmatrix} 6.2 & 6.2 \\ 0 & -2 \end{vmatrix} = 5 \cdot 6.2|-2| = 5 \cdot 6.2(-2) = -62$$

We also observe that the number of cycles is $n - 1 = 2$, where $n = 3$ is the order of the determinant.

As a second example, we consider the determinant

$$D = \begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 3 \\ 1 & 3 & 6 \end{vmatrix} \tag{6-16}$$

The first cycle leads to

$$D = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 5 \end{vmatrix}$$

We now run into the same difficulty that occurred with the corresponding matrix in Sec. 6-1: a zero pivotal element in the second row. Again, we get around the difficulty by interchanging rows. This changes the sign of the determinant, and we allow for this by changing the sign of one row. Thus

$$D = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 1 & 5 \\ 0 & 0 & -1 \end{vmatrix} = 1 \cdot 1(-1) = -1$$

As a third example, consider

$$D = \begin{vmatrix} 1 & 2 & 3 & 1 \\ 2 & 4 & 6 & 4 \\ 3 & 6 & 2 & 1 \\ 2 & 4 & 3 & 2 \end{vmatrix} \tag{6-17}$$

The first cycle leads to

$$D = \begin{vmatrix} 1 & 2 & 3 & 1 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & -7 & -2 \\ 0 & 0 & -3 & 0 \end{vmatrix}$$

We have not yet cleared all the elements below the diagonal. Nevertheless it is impossible to continue, because all the available pivotal elements in the second column are zero. It can be shown that this situation occurs only if the value of the determinant is zero, so the evaluation is completed at this point. The result is $D = 0$.

A program follows. Line 1 is the title. Line 10 assigns the value of n, the order of the determinant. Line 20 is a DIMension statement. To save space in the computer memory, we run the subscripts from 0 to $n - 1$. Lines 30 through 110 read and print the elements of the determinant. Lines 120 through 340 clear the elements below the diagonal. (The segment of lines 130 through 280 shuffles the rows when necessary to get rid of zero pivotal elements—or small pivotal elements. For many applications this segment may not be necessary.) Lines 350 through 380 calculate the value of the determinant by multiplying the diagonal elements. Lines 390 and 400 print the result. The last few lines are data lines for Eq. 6-15.

```
  1  REM: EVALUATION OF A DETERMINANT
 10  N=3                                  Assigns value of n.
 20  DIM A(N-1,N-1)                       DIMension statement.
 30  PRINT                                }
 40  PRINT "THE DETERMINANT               }
     IS:"                                 }
 50  FOR I=0 TO N-1                       }
 60  FOR J=0 TO N-1                       }  Reads and prints
 70  READ A(I,J)                          }  elements of determinant.
 80  PRINT A(I,J):                        }
 90  NEXT J                               }
100  PRINT                                }
110  NEXT I                               }
```

```
120 FOR K=0 TO N-2
130 Z=K
140 Y=ABS(A(K,K))
150 FOR I=K+1 TO N-1
160 IF ABS(A(I,K))<=Y THEN 190
170 Z=I
180 Y=ABS(A(I,K))
190 NEXT I
200 IF Y>10^-7 THEN 230
210 D=0
220 GOTO 390
230 IF Z=K THEN 290
240 FOR J=K TO N-1
250 T=A(K,J)
260 A(K,J)=A(Z,J)
270 A(Z,J)=-T
280 NEXT J
290 FOR I=K+1 TO N-1
300 FOR J=K+1 TO N-1
310 A(I,J)=A(I,J)
    -A(K,J)*A(I,K)/A(K,K)
320 NEXT J
330 NEXT I
340 NEXT K
350 D=1
360 FOR I=0 TO N-1
370 D=D*A(I,I)
380 NEXT I
390 PRINT
400 PRINT "THE VALUE OF THE
    DETERMINANT IS ";D
410 DATA 5,-2,3
420 DATA -2,7,5
430 DATA 3,5,6
```

Finds largest pivot.

Terminates calculation if D=0.

Rearranges rows.

Clears elements below diagonal.

Generates triangular matrix.

Multiplies diagonal elements.

Prints results.

Data lines.

To operate the program, line 10 is filled in by the user. The data lines at the end are also filled in by the user. The result verifies the value −62. If spaces are needed between the elements in the display, the expression " "; may be appended to line 80. If spaces are desired between the rows, a PRINT statement may be inserted as line 55. The same remarks that were made in Sec. 6-1 apply here to the DIMension statement of line 20 and the numerical constant in line 200.

6-4. Matrix Eigenvalues

We begin this section with a physical application. We consider the problem of finding the natural frequencies of vibration of the system of springs and masses sketched in Fig. 6-1.

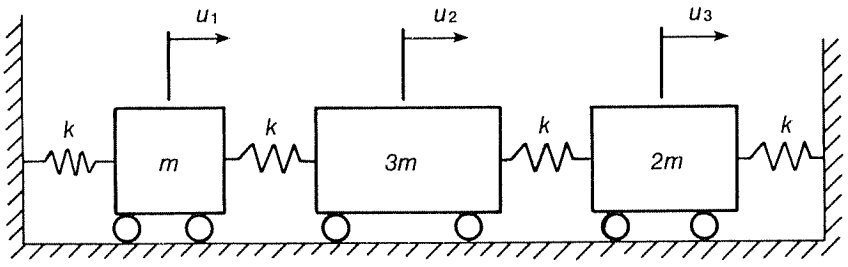


FIG. 6-1

The differential equations of motion are

$$-ku_1 + k(u_2 - u_1) = m\frac{d^2u_1}{dt^2}$$

$$-k(u_2 - u_1) + k(u_3 - u_2) = 3m\frac{d^2u_2}{dt^2}$$

$$-k(u_3 - u_2) - ku_3 = 2m\frac{d^2u_3}{dt^2}$$

where u_1, u_2, and u_3 are the displacements of the three masses from their rest positions, m is a unit mass, k is the spring stiffness, and t is the time. We let

$$u_j = x_j e^{i\omega t} \qquad j = 1,2,3$$

The parameter ω is known as the angular frequency, and x_j is the amplitude of motion of the jth mass. The equations of motion now become

$$2x_1 - x_2 = \lambda x_1 \tag{6-18a}$$

$$-x_1 + 2x_2 - x_3 = 3\lambda x_2 \tag{6-18b}$$

$$-x_2 + 2x_3 = 2\lambda x_3 \tag{6-18c}$$

where

$$\lambda = \frac{m\omega^2}{k}$$

We wish to solve Eqs. 6-18 for λ. The most obvious (although usually not the easiest) way to proceed is to work with the determinantal equation

$$\begin{bmatrix} (2-\lambda) & -1 & 0 \\ -1 & (2-3\lambda) & -1 \\ 0 & -1 & (2-2\lambda) \end{bmatrix} = 0 \tag{6-19}$$

Expansion of the determinant by elementary algebra leads to the equation

$$6\lambda^3 - 22\lambda^2 + 21\lambda - 4 = 0$$

By using the program of Sec. 2-6, we obtain the three roots

$$\lambda = .2528214993 \qquad 1.180920530 \qquad 2.232924637$$

This procedure is often used to solve a set of two equations, and is occasionally used to solve a set of three equations. However, for large sets of equations the algebraic labor is prohibitive. A second method is to solve the determinantal Eq. 6-19 by the program of Sec. 6-3, finding all the roots by trial and error. This procedure is also rather clumsy.

We now adopt a different approach. Equations 6-18 may be rewritten in matrix form as

$$AX = \lambda BX \tag{6-20}$$

where

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{6-21}$$

From the standpoint of the determinantal expansion, it is clear that the solution of a set of n simultaneous equations for λ is equivalent to the solution of an nth degree polynomial equation. Hence there are n values of λ that will satisfy the equations. These roots are known as eigenvalues or characteristic values. The corresponding values of the xs are known as eigenvectors.

We shall obtain a solution by iteration. To do so, it is necessary to have the X matrix alone on one side of the equation. There are two possible ways to accomplish this. We may premultiply both sides by A^{-1}. Then by the same exercise that led from Eq. 6-13 to Eq. 6-14, we find that

$$X = \lambda GX \quad \text{where } G = A^{-1}B \tag{6-22}$$

On the other hand, we may premultiply by B^{-1} to obtain

$$\lambda X = HX \quad \text{where } H = B^{-1}A \tag{6-23}$$

For a reason that will become apparent subsequently, the first formulation is usually preferable. By using either matrix algebra or the inversion program of Sec. 6-2, we find that

$$A^{-1} = \frac{1}{4}\begin{bmatrix} 3 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

It follows that

$$G = \frac{1}{4}\begin{bmatrix} 3 & 6 & 2 \\ 2 & 12 & 4 \\ 1 & 6 & 6 \end{bmatrix}$$

We have now reduced the problem to the problem of solving the matrix equation

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \frac{\lambda}{4}\begin{bmatrix} 3 & 6 & 2 \\ 2 & 12 & 4 \\ 1 & 6 & 6 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

or equivalently

$$\frac{x_1}{\lambda} = .75x_1 + 1.5x_2 + .5x_3 \tag{6-24a}$$

$$\frac{x_2}{\lambda} = .5x_1 + 3x_2 + x_3 \tag{6-24b}$$

$$\frac{x_3}{\lambda} = .25x_1 + 1.5x_2 + 1.5x_3 \tag{6-24c}$$

To solve by iteration, we choose a set of values for the xs and substitute it into the right sides of these equations. The simplest choice is $x_1 = x_2 = x_3 = 1$. The results are

$$x_1 = 2.75\lambda \qquad x_2 = 4.5\lambda \qquad x_3 = 3.25\lambda$$

We have a system of n equations in $n + 1$ unknowns—the n xs and λ. Any one of the xs may be assigned a value arbitrarily. We set $x_1 = 1$. Then the results of the first iteration are

$$\lambda = .3636 \qquad x_1 = 1 \qquad x_2 = 1.6364 \qquad x_3 = 1.1818$$

A second iteration leads to

$$\lambda = .2635 \qquad x_1 = 1 \qquad x_2 = 1.7365 \qquad x_3 = 1.1796$$

After repeated iterations, the solution eventually converges to

$$\lambda = .2528214993$$
$$x_1 = 1$$
$$x_2 = 1.747178501$$
$$x_3 = 1.169184137$$

In presenting the final results, it is customary to set the eigenvector with the greatest absolute value equal to unity. With this convention, the results become

$$\lambda = .2528214993$$
$$x_1 = .5723513651$$
$$x_2 = 1$$
$$x_3 = .6691841368$$

We have seen that, for a set of n simultaneous equations, there are n eigenvalues, that is, n values of λ. It can be shown that, if the iteration process based on Eq. 6-22 converges, it yields the root with the smallest absolute value. In most physical applications, this is the root of primary interest, and it is often the only root of interest. (A solution based on Eq. 6-23 would yield the root with the greatest absolute value. For this reason, Eq. 6-22 was chosen.) However, it sometimes happens that higher eigenvalues are required, and we shall now develop a procedure to find these.

It will be assumed throughout the remainder of this chapter that the A and B matrices are symmetric. To find solutions for higher modes of vibration, we need a relation between the eigenvectors for different modes. Equation 6-20 is valid for any mode; we rewrite it for two distinct modes, say p and q, with the mode numbers shown as superscripts:

$$AX^{(p)} = \lambda^{(p)} BX^{(p)} \tag{6-25}$$

$$AX^{(q)} = \lambda^{(q)} BX^{(q)} \tag{6-26}$$

We now take the transpose of both sides of Eq. 6-25, recalling from Sec. 6-2 that the transpose of a product is equal to the product of the transposes taken in reverse order. This leads to

$$X^{(p)T} A = \lambda^{(p)} X^{(p)T} B \tag{6-27}$$

We have also used the fact that the transpose of a symmetric square matrix is identical to the original matrix. We now premultiply Eq. 6-26 by $X^{(p)T}$ and postmultiply Eq. 6-27 by $X^{(q)}$, then subtract. The result is

$$[\lambda^{(q)} - \lambda^{(p)}] X^{(p)T} B X^{(q)} = 0$$

We assume that the solution contains no multiple eigenvalues. Then $\lambda^{(p)} \neq \lambda^{(q)}$, and it follows that

$$X^{(p)T} B X^{(q)} = 0 \tag{6-28}$$

This result is known as an *orthogonality condition*; it connects the eigenvectors for two distinct modes. This may be broken into the two equations

$$F^{(p)} = X^{(p)T} B \qquad F^{(p)} X = 0$$

(We have dropped the superscript q. Mode numbers are not needed for the xs, because only current values are used at each stage of the calculations. It is clear that B is a square matrix, X is a column matrix, $X^{(p)T}$ is a row matrix, and $F^{(p)}$ is a row matrix. The product $F^{(p)}X$ is a matrix consisting of a single element. For the subsequent analysis, we need the expanded forms

$$f_j^{(p)} = \sum_{k=1}^{n} b_{kj} x_k^{(p)} \tag{6-29}$$

$$\sum_{j=1}^{n} f_j^{(p)} x_j = 0 \tag{6-30}$$

The procedure is to start by finding the smallest root of λ in Eq. 6-20 by iteration. This root is then eliminated from the system of equations by using Eqs. 6-29 and 6-30. We then solve the reduced set by iterating as before to obtain the smallest remaining root (the second root of the original set). This procedure may be repeated as many times as necessary to obtain higher eigenvalues.

We have already found the eigenvectors $x_k^{(1)}$ for the first mode of vibration of the problem of Fig. 6-1. It now follows from Eq. 6-29 that

$$\begin{aligned} f_1^{(1)} &= 1 \cdot .5723513651 = .5723513651 \\ f_2^{(1)} &= 3 \cdot 1 \qquad\qquad\quad = 3 \\ f_3^{(1)} &= 2 \cdot .6691841368 = 1.338368274 \end{aligned}$$

Substitution of these results into Eq. 6-30 leads to the result

$$0 = .5723513651x_1 + 3x_2 + 1.338368274x_3 \tag{6-31}$$

We use this equation to eliminate the last term on the right side of each of Eqs. 6-24. The results are

$$\frac{x_1}{\lambda} = .5361756826x_1 + .3792322490x_2 \tag{6-32a}$$

$$\frac{x_2}{\lambda} = .0723513651x_1 + .7584644980x_2 \tag{6-32b}$$

$$\frac{x_3}{\lambda} = -.3914729523x_1 + 1.862303253x_2 \tag{6-32c}$$

We solve these equations by iteration to find the results for the second mode. If only λ is required, the first two equations are sufficient. However, if we want the xs, all three equations are needed. The results are

$$\begin{aligned} \lambda &= 1.180920530 \\ x_1 &= -.4417655105 \\ x_2 &= -.3618410602 \\ x_3 &= 1 \end{aligned}$$

To find the results for the third mode, we need another orthogonality condition. Using Eqs. 6-29 and 6-30 with the new set of xs, we find that

$$0 = -.4417655104x_1 - 1.085523181x_2 + 2x_3 \tag{6-33}$$

We use this result to eliminate the last term on the right side of Eq. 6-31. It follows that

$$0 = .8679738369x_1 + 3.726414893x_2$$

We use this equation to eliminate the last term on the right side of each of the Eqs. 6-32. The results are

$$\frac{x_1}{\lambda} = .4478431486x_1 \tag{6-34a}$$

$$\frac{x_2}{\lambda} = -.1043137029x_1 \tag{6-34b}$$

$$\frac{x_3}{\lambda} = .0423033572x_1 \tag{6-34c}$$

There is no need for another iterative process; the problem is solved at this point. The results for the third mode are

$$\lambda = 2.232924637$$
$$x_1 = 1$$
$$x_2 = -.2329246372$$
$$x_3 = .0944602086$$

The same procedure can easily be applied to a set of any number of equations.

The program follows. The first few parts are very similar to parts of the programs of Sec. 6-2. Line 1 is the title. Line 10 assigns the value of n, the number of equations. The next five lines are DIMension statements. Lines 70 through 180 read the as and bs and print the A matrix. (We again number the subscripts from 0 in the program.) Lines 190 through 260 print the B matrix. Lines 270 through 430 invert the A matrix, and lines 440 through 510 calculate the product $G = A^{-1}B$. Line 520 assigns the value $p = 1$ for the first mode. Lines 530 through 700 carry out the iterative evaluation of λ and print the result. The ts are the values of x_i/λ found from the iterative equations; these are divided through by t_1 (t_0 in the program) to obtain the next higher set of approximations for the xs. Line 720 is a dummy input statement of the type used in Sec. 2-1. Successive approximations for λ are obtained by pressing the ENTER key repeatedly. Line 730 then sends the execution back to iterate again. After satisfactory convergence has been obtained, the operator enters X. (Line 710 bypasses this operation if $p = n$, because no iteration is needed

for the last mode.) Lines 740 through 840 normalize the xs with respect to the greatest absolute value and print the results.

The remainder of the program is concerned with the calculation of results for higher modes. Line 850 tests whether the current mode number p is less than n, the number of modes. If $p = n$, execution ends at line 860. Otherwise, execution proceeds to the orthogonality calculations in preparation for the next higher mode. Lines 870 through 920 calculate the fs of Eq. 6-29. Instead of introducing a new subscripted variable F into the program, we save memory space by reusing the variable A. (In the program, there is no distinction between subscripts and superscripts.) Lines 940 through 980 eliminate the last x between the new orthogonality relation and the old orthogonality relations. (This segment is skipped by line 930 on the first cycle.) Lines 990 through 1030 use the resulting orthogonality relation to eliminate the last x from the iterative equations. Line 1040 adjusts the value of p for the next higher mode, and line 1050 resets the dummy input variable to a null string. Line 1060 sends the execution back to line 530 to start the iteration for the next higher mode. The last few lines are the data lines for Eq. 6-21. Each line contains one row of the A matrix and one row of the B matrix.

```
  1 REM: MATRIX EIGENVALUES
 10 N=3                                   Assigns value of n.
 20 DIM A(N-1,N)                        ┐
 30 DIM B(N-1,N-1)                      │
 40 DIM G(N-1,N-1)                      ├ DIMension statements.
 50 DIM X(N-1)                          │
 60 DIM T(N-1)                          ┘
 70 PRINT                               ┐
 80 PRINT "THE A MATRIX IS:"            │
 90 FOR I=0 TO N-1                      │
100 FOR J=0 TO N-1                      │
110 READ A(I,J)                         │
120 PRINT A(I,J);                       ├ Reads as and bs;
130 NEXT J                              │ prints as.
140 FOR J=0 TO N-1                      │
150 READ B(I,J)                         │
160 NEXT J                              │
170 PRINT                               │
180 NEXT I                              ┘
190 PRINT                               ┐
200 PRINT "THE B MATRIX IS:"            │
210 FOR I=0 TO N-1                      │
220 FOR J=0 TO N-1                      ├ Prints bs.
230 PRINT B(I,J);                       │
240 NEXT J                              │
250 PRINT                               │
260 NEXT I                              ┘
```

```
270  FOR K=0 TO N-1
280  FOR I=0 TO N-1
290  A(I,N)=ABS(I=K)
300  NEXT I
310  FOR J=1 TO N
320  A(K,J)=A(K,J)/A(K,0)
330  FOR I=0 TO N-1
340  IF I=K THEN 360
350  A(I,J)=A(I,J)-A(K,J)*A(I,0)
360  NEXT I
370  NEXT J
380  FOR J=0 TO N-1
390  FOR I=0 TO N-1
400  A(I,J)=A(I,J+1)
410  NEXT I
420  NEXT J
430  NEXT K
```

Inverts A matrix.

```
440  FOR I=0 TO N-1
450  FOR J=0 TO N-1
460  G(I,J)=0
470  FOR K=0 TO N-1
480  G(I,J)=G(I,J)+A(I,K)*B(K,J)
490  NEXT K
500  NEXT J
510  NEXT I
```

Calculates matrix product $G=A^{-1}B$.

```
520  P=1
```

Initializes p, the mode number.

```
530  FOR I=0 TO N-1
540  X(I)=1
550  NEXT I
560  PRINT
570  PRINT "THE VALUE OF LAMBDA
     FOR MODE";P;"IS ";
580  IF P=N THEN 600
590  PRINT "FOUND BY ITERATION
     AS FOLLOWS:";
600  PRINT
610  FOR I=0 TO N-1
620  T(I)=0
630  FOR J=0 TO N-P
640  T(I)=T(I)+G(I,J)*X(J)
650  NEXT J
660  NEXT I
670  FOR I=0 TO N-1
680  X(I)=T(I)/T(0)
690  NEXT I
700  PRINT 1/T(0)
```

Iterates for λ and prints result.

```
710  IF P=N THEN 740
```

Bypasses iteration for last mode.

```
720 INPUT Q$                                  Interrupts execution.
730 IF Q$<>"X" THEN 610                       Returns for next iteration.
740 Y=T(0)                                    ⎫
750 FOR I=1 TO N-1                            ⎪
760 IF ABS(T(I))<=ABS(Y) THEN 780             ⎪
770 Y=T(I)                                    ⎪
780 NEXT I                                    ⎪ Normalizes xs and
790 PRINT                                     ⎪ prints results.
800 PRINT "THE X MATRIX IS:"                  ⎪
810 FOR I=0 TO N-1                            ⎪
820 X(I)=T(I)/Y                               ⎪
830 PRINT X(I)                                ⎪
840 NEXT I                                    ⎭
850 IF P<N THEN 870                           Proceeds to next mode if p<n.
860 END                                       Terminates execution if p=n.
870 FOR J=0 TO N-1                            ⎫
880 A(P,J)=0                                  ⎪
890 FOR K=0 TO N-1                            ⎪ Calculates orthogonality factors.
900 A(P,J)=A(P,J)+B(K,J)*X(K)                 ⎪
910 NEXT K                                    ⎪
920 NEXT J                                    ⎭
930 IF P=1 THEN 990                           ⎫
940 FOR I=P-1 TO 1 STEP -1                    ⎪
950 FOR J=0 TO N-P+I                          ⎪
960 A(I,J)=A(I,J)-A(I+1,J)                    ⎪
    *A(I,N-P+I)/A(I+1,N-P+I)                  ⎪
970 NEXT J                                    ⎪ Solves orthogonality
980 NEXT I                                    ⎪ equations.
990 FOR I=0 TO N-1                            ⎪
1000 FOR J=0 TO N-P                           ⎪
1010 G(I,J)=G(I,J)                            ⎪
    -G(I,N-P)*A(1,J)/A(1,N-P)                 ⎪
1020 NEXT J                                   ⎪
1030 NEXT I                                   ⎭
1040 P=P+1                                    Adjusts value of p.
1050 Q$=""                                    Resets Q$ to null string.
1060 GOTO 530                                 Returns for next higher mode.
1070 DATA 2,-1,0,1,0,0                        ⎫
1080 DATA -1,2,-1,0,3,0                       ⎪ Data lines for matrix elements.
1090 DATA 0,-1,2,0,0,2                        ⎭
```

To operate the program, line 10 is filled in by the user. The data lines at the end are also filled in by the user. When the program is run, the computer performs one iteration for the lowest value of λ and displays the result on the screen, followed by a question mark. This represents the dummy INPUT statement of line 720. To iterate again, the operator presses the ENTER key, and the iteration is repeated. This operation is

repeated until the results on the screen show satisfactory convergence. The operator then enters X. The X matrix for the first mode is displayed, and the computer starts the iterative process for the second mode. This procedure is repeated until all the desired modes have been analyzed. The evaluation may be carried through all the n modes, or it may be terminated at any intermediate point by pressing the BREAK key.

This program runs as it stands on almost any commonly used model of microcomputer, with one reservation. The comment about the DIMension statement that followed the first program of Sec. 6-1 also obtains here. The dummy input statement of line 720 also requires a comment. With most microcomputers the act of pressing the ENTER key in response to a string input question without entering anything enters a null string, that is, a string consisting of nothing. The TRS-80 and the Commodore 64 are exceptions. With these computers the entry of nothing simply restarts the execution and any preexisting string is retained. This will work satisfactorily for the first mode. However, the X that is entered to print the X matrix for the first mode will be retained permanently unless the program contains an instruction to clear it, and no iterations will be performed for the higher modes. This trouble is eliminated by line 1050, which assigns a null value to the string variable Q$ to start the iteration for each higher mode, thus erasing X. This line is needed only for the TRS-80 and Commodore 64, but may be left in the program for any other model, since it does not increase the length of the program appreciably.

Three remarks about the display may be helpful. If spaces are needed between the elements, the expression " "; may be appended to lines 120 and 230. If spaces are desired between the rows, PRINT statements may be inserted as lines 95, 215, and 815. Line 710 also requires a comment. Since no iteration is required for the highest mode $p = n$, this line bypasses the iterative process for $p = n$ and prints the results automatically following the mode $p = n - 1$. For a large matrix, the results for two modes may not fit onto the screen at the same time. If the computer is used without a printer, it may be necessary to delete line 710.

This program can be made a little more convenient to use by revising it to include the extrapolation process of Sec. 2-1. The following amendment accomplishes this. The convergence is greatly improved, and it is not necessary for the operator to press the ENTER key for each iteration. The iterations are repeated automatically until the estimated error of x_n is within the limit allowed by line 735. Double subscripts are now used for the xs; the first subscript is the eigenvalue number as before, and the second represents the stage of the extrapolation process.

```
 50  DIM X(N-1,2)
525  INPUT Q$
540  X(I,0)=1
580  PRINT
```

```
600 FOR K=1 TO 2
640 T(I)=T(I)+G(I,J)*X(J,K-1)
680 X(I,K)=T(I)/T(0)
700 NEXT K
705 FOR I=1 TO N-1
710 IF X(I,1)=X(I,2) THEN 740
715 R=(X(I,2)-X(I,1))/(X(I,1)-X(I,0))
720 E=(X(I,2)-X(I,1))/(1-1/R)
725 X(I,0)=X(I,2)-E
730 NEXT I
735 IF ABS(E)>10^-7 THEN 600
785 PRINT 1/T(0)
820 X(I,0)=T(I)/Y
830 PRINT X(I,0)
900 A(P,J)=A(P,J)+B(K,J)*X(K,0)
1060 GOTO 525
```

To operate the amended program, the operator enters RUN. The A and B matrices appear on the screen, followed by a question mark. The operator then presses the ENTER key, and the final iterated value of λ for the first mode appears on the screen, followed by the X matrix. Corresponding results for higher modes are obtained by pressing the ENTER key once for each mode. If the intermediate iterative values of λ are desired, they can be obtained by adding the following line to the program:

```
732 PRINT 1/T(0)
```

The allowable error in line 735 may have to be adjusted to fit the accuracy of the computer, as discussed in Chapter 2.

Problems

Solve the sets of simultaneous equations 6-1 through 6-4. (Answers are given in order $x_1, x_2, \ldots$.)

6-1.
$$\begin{aligned} 3x_1 - 4x_2 + 2x_3 &= -1 \\ 4x_1 + 3x_2 - 6x_3 &= 1 \\ 2x_1 - 6x_2 + x_3 &= -10 \end{aligned}$$
Ans. 5 3 −2

6-2.
$$\begin{aligned} 2x_1 + x_2 + 3x_3 &= 15 \\ x_1 + 5x_2 + x_3 &= -3 \\ 3x_1 + x_2 + 2x_3 &= 16 \end{aligned}$$
Ans. 4 −2 3

6-3.
$$\begin{aligned} x_1 - 3x_2 + 2x_3 - 5x_4 &= 6 \\ -3x_1 + 2x_2 - x_3 + 2x_4 &= -3 \\ 2x_1 - x_2 + 4x_3 - 2x_4 &= 11 \\ -5x_1 + 2x_2 - 2x_3 + 6x_4 &= -15 \end{aligned}$$
Ans. 1 3 2 −2

6-4. $2x_1 - x_2 = 5$
$-x_1 + 3x_2 - 2x_3 = -4$
$-2x_2 + 5x_3 - x_4 = 6$
$-x_3 + 4x_4 = 6$ *Ans.* 3 1 2 2

Verify the matrix operations 6-5 through 6-10.

6-5. $$\begin{bmatrix} 3 & -4 & 2 \\ -4 & 3 & -6 \\ 2 & -6 & 1 \end{bmatrix}\begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & 1 \\ 3 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 8 & -15 & 9 \\ -23 & 5 & -21 \\ 1 & -27 & 2 \end{bmatrix}$$

6-6. $$\begin{bmatrix} 3 & -4 & 2 \\ -4 & 3 & -6 \\ 2 & -6 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 19 \\ -24 \\ 19 \end{bmatrix}$$

6-7. $$\begin{bmatrix} 3 & 2 \\ -4 & 3 \\ 2 & -1 \end{bmatrix}\begin{bmatrix} 2 & 8 & -3 & 5 \\ 3 & 7 & 1 & 4 \end{bmatrix} = \begin{bmatrix} 12 & 38 & -7 & 23 \\ 1 & -11 & 15 & -8 \\ 1 & 9 & -7 & 6 \end{bmatrix}$$

6-8. $$\begin{bmatrix} 3 & -4 & 2 \\ -4 & 3 & -6 \\ 2 & -6 & 1 \end{bmatrix}^{-1} = \frac{1}{31}\begin{bmatrix} 33 & 8 & -18 \\ 8 & 1 & -10 \\ -18 & -10 & 7 \end{bmatrix}$$

6-9. $$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & 1 \\ 3 & 1 & 2 \end{bmatrix}^{-1} = \frac{1}{23}\begin{bmatrix} -9 & -1 & 14 \\ -1 & 5 & -1 \\ 14 & -1 & -9 \end{bmatrix}$$

6-10. $$\begin{bmatrix} 1 & -3 & 2 & -5 \\ -3 & 2 & -1 & 2 \\ 2 & -1 & 4 & -2 \\ -5 & 2 & -2 & 6 \end{bmatrix}^{-1} = \frac{1}{117}\begin{bmatrix} -26 & -26 & 0 & -13 \\ -26 & 64 & 9 & -40 \\ 0 & 9 & 36 & 9 \\ -13 & -40 & 9 & 25 \end{bmatrix}$$

6-11. Invert the matrices on the right sides of Prob. 6-8 through 6-10, and compare the results with the original matrices.

6-12. Evaluate the determinants of the three square matrices in Prob. 6-5. Observe that the value of the determinant of the product is equal to the product of the values of the determinants of the factors. It can be shown that this is true in general.

Ans. −31 −23 713.

6-13. Find the eigenvalues of the following system of equations:

$3u_1 - u_2 = 3\lambda u_1$
$-u_1 + 2u_2 - u_3 = \lambda u_2$
$-u_2 + u_3 = 2\lambda u_3$ *Ans.* .15521997 .86653745 2.4782426

6-14. Find the eigenvalues of the following matrix equation:

$$\begin{bmatrix} 24 & 0 & -12 \\ 0 & 8 & -6 \\ -12 & -6 & 12 \end{bmatrix} = \lambda \begin{bmatrix} 72 & 0 & -36 \\ 0 & 8 & -3 \\ -36 & -3 & 36 \end{bmatrix}$$

Ans. .08286539 1/3 1.072690

Appendix: Numerical Methods

In this appendix we give derivations of three numerical methods: the Gauss method of numerical integration, used in Chapter 4, and the Runge-Kutta and Adams methods of solving differential equations, used in Chapter 5.

A-1. Gauss Integration

The basic formula for numerical integration is

$$I = \int_a^b y\,dx = w_1 y_1 + w_2 y_2 + \ldots + w_n y_n \tag{A-1}$$

where the y_js are the values of the function at n base points x_j, and the w_js are appropriate weighting factors. We represent the function y by a polynomial

$$y = a_0 + a_1 x + a_2 x^2 + \ldots \tag{A-2}$$

To obtain numerical results from Eq. A-1, it is necessary to specify the values of x at the n base points at which the function is to be evaluated.

One obvious possibility is to use uniformly spaced base points. Then the use of n points will determine the function y exactly and uniquely, provided that y is a polynomial of degree not greater than $n - 1$, and the numerical integration will be exact. Instead of arbitrarily choosing to place the base points $x_1, x_2, \ldots, x_n$ at equal intervals, we may choose to find the values of the x_js that will lead to the most accurate numerical evaluation. If the x_js are considered to be adjustable, as well as the w_js, we have $2n$ adjustable parameters, and it is possible to obtain an evaluation that will give exact results for a polynomial of degree $\leq 2n - 1$. This is the basic idea of Gauss integration.

Legendre polynomials play a major part in the theory of Gauss integration. These are discussed in advanced calculus and have been considered briefly in Sec. 1-5. To take advantage of the orthogonality properties of the Legendre polynomials, we change the interval of integration in Eq. A-1, using limits -1 and 1 instead of a and b. We also change the independent variable to ξ, reserving the symbol x for the general interval a to b. After the analysis is completed on this basis, the results can easily be applied to the more general interval by using Eq. 4-9b. The basic formula now becomes

$$I = \int_{-1}^{1} y d\xi = w_1 y_1 + w_2 y_2 + \ldots + w_n y_n \tag{A-3}$$

We shall now find the values of the ξ_js and w_js, which lead to an exact numerical integration provided that y is a polynomial of degree not greater than $2n - 1$. Let y be a polynomial of degree $2n - 1$. Then y may be written in the form

$$y = P_n(\xi) q_{n-1}(\xi) + r_{n-1}(\xi) \tag{A-4}$$

where $P_n(\xi)$ is the Legendre polynomial of degree n, $q_{n-1}(\xi)$ is the quotient obtained by dividing $P_n(\xi)$ into y, and $r_{n-1}(\xi)$ is the remainder. $q_{n-1}(\xi)$ and $r_{n-1}(\xi)$ are polynomials of degree $n - 1$. Substitution of Eq. A-4 into the middle member of Eq. A-3 leads to

$$I = \int_{-1}^{1} P_n(\xi) q_{n-1}(\xi) d\xi + \int_{-1}^{1} r_{n-1}(\xi) d\xi$$

Since the function $q_{n-1}(\xi)$ is a polynomial of degree $n - 1$, it can be expressed as a linear combination of Legendre polynomials of degree not greater than $n - 1$. Each of these is orthogonal to $P_n(\xi)$ in the interval -1 to 1. Therefore the first integral on the right side of the foregoing equation is zero, and it follows that

$$I = \int_{-1}^{1} r_{n-1}(\xi) d\xi \tag{A-5}$$

Substitution of Eq. A-4 into the last member of Eq. A-3 leads to

$$I = \sum_{j=1}^{n} w_j [P_n(\xi_j) q_{n-1}(\xi_j) + r_{n-1}(\xi_j)] \tag{A-6}$$

We want the numerical evaluation of Eq. A-6 to give exactly the same result as the exact expression of Eq. 6-5. Since the latter expression is independent of $q_{n-1}(\xi)$, the former one must be also. This is true if and only if

$$P_n(\xi_j) = 0 \tag{A-7}$$

that is, the ξ_js are the zeros of the Legendre polynomial $P_n(\xi)$.

We now have an equation for the ξ_js; we still need an equation for the w_js. To obtain this, we use the Lagrange interpolatory expression for y. This has appeared previously as Eq. 1-17. With ξ substituted for x, the equation is

$$y = \sum_{j=1}^{n} \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{\xi - \xi_i}{\xi_j - \xi_i} y_j \tag{A-8}$$

By integrating both sides of this equation between the limits -1 and 1, then comparing the result with Eq. A-3 and matching coefficients of corresponding terms, we find that

$$w_j = \int_{-1}^{1} \prod_{\substack{i=1 \\ i \neq j}}^{n} \frac{\xi - \xi_i}{\xi_j - \xi_i} \, d\xi$$

$$= \frac{1}{(\xi_j - \xi_1) \cdots (\xi_j - \xi_{j-1})(\xi_j - \xi_{j+1}) \cdots (\xi_j - \xi_n)} \int_{-1}^{1} \frac{\prod_{i=1}^{n} (\xi - \xi_i)}{\xi - \xi_j} \, d\xi \tag{A-9}$$

It is clear that $P_n(\xi)$ can be expressed as

$$P_n(\xi) = c \prod_{i=1}^{n} (\xi - \xi_i)$$

where c is the coefficient of ξ^n and the ξ_is are the roots of Eq. A-7. It follows that

$$P_n'(\xi_j) = c(\xi_j - \xi_1) \cdots (\xi_j - \xi_{j-1})(\xi_j - \xi_{j+1}) \cdots (\xi_j - \xi_n)$$

Equation A-9 now becomes

$$w_j = \frac{1}{P_n'(\xi_j)} \int_{-1}^{1} \frac{P_n(\xi)}{\xi - \xi_j}\, d\xi \tag{A-10}$$

To evaluate this integral we need Christoffel's summation formula (reference 7, page 101). This is

$$\frac{n}{\xi - t}\,[P_{n-1}(t)P_n(\xi) - P_n(t)P_{n-1}(\xi)]$$
$$= [P_0(t)P_0(\xi) + 3P_1(t)P_1(\xi) + \ldots + (2n-1)P_{n-1}(t)P_{n-1}(\xi)] \tag{A-11}$$

We set $t = \xi_j$ and use Eq. A-7. Then it follows that

$$\frac{P_n(\xi)}{\xi - \xi_j} = \frac{1}{nP_{n-1}(\xi_j)}\,[P_0(\xi_j)P_0(\xi) + 3P_1(\xi_j)P_1(\xi) + \ldots$$
$$+ (2n-1)P_{n-1}(\xi_j)P_{n-1}(\xi)] \tag{A-12}$$

We substitute Eq. A-12 into the right side of A-10. The integrals of all of the terms in brackets except the first are equal to zero, and we find that

$$w_j = \frac{2}{nP_{n-1}(\xi_j)P_n'(\xi_j)} \tag{A-13}$$

The parameter $P_n(\xi_j)$ will be evaluated by the procedure of Chapter 1, using a recurrence formula. During this process we will obtain the value of $P_{n-1}(\xi_j)$ as an intermediate step. To use Eq. A-13, we also need the value of $P_n'(\xi_j)$. It is shown on page 100 of reference 7 that

$$P_n'(\xi) = \frac{n}{1-\xi^2}\,[P_{n-1}(\xi) - \xi P_n(\xi)] \tag{A-14}$$

With the help of Eq. A-7, it follows that

$$P_n'(\xi_j) = \frac{nP_{n-1}(\xi_j)}{1-\xi_j^2}$$

Equation A-13 can now be rewritten as

$$w_j = \frac{2(1-\xi_j^2)}{[nP_{n-1}(\xi_j)]^2} \tag{A-15}$$

We now have the necessary equations to evaluate the Gauss coefficients. The ξ_js are found from Eq. A-7, with the help of the equations

of Sec. 1-5. The w_js are found from Eq. A-15. Only half of the required ξ_js and w_js actually have to be calculated. Since each Legendre polynomial contains only even powers or only odd powers, the ξ_js given by Eq. A-7 occur in $\pm$ pairs. Also, the corresponding w_js occur in positive pairs, since only squares of the parameters appear on the right side of Eq. A-15. Hence it is sufficient to consider the $n/2$ positive values of ξ_j if n is even, or the $(n+1)/2$ positive values (including zero) if n is odd. For $n = 1$ through 5, the evaluations can easily be made algebraically. The results are

$$n=1 \quad \xi_1 = 0 \qquad w_1 = 2$$

$$n=2 \quad \xi_1 = \frac{1}{\sqrt{3}} \qquad w_1 = 1$$

$$n=3 \quad \xi_1 = 0 \qquad w_1 = \frac{8}{9}$$

$$\xi_2 = \sqrt{\frac{3}{5}} \qquad w_2 = \frac{5}{9}$$

$$n=4 \quad \xi_{1,2} = \left[\frac{1}{7}\left(3 \mp \sqrt{\frac{24}{5}}\right)\right]^{1/2} \qquad w_{1,2} = \frac{1}{6}\left(3 \pm \sqrt{\frac{5}{6}}\right)$$

$$n=5 \quad \xi_1 = 0 \qquad w_1 = \frac{128}{225}$$

$$\xi_{2,3} = \frac{1}{3}\left(5 \mp \sqrt{\frac{40}{7}}\right)^{1/2} \qquad w_{2,3} = \frac{1}{900}(322 \pm 13\sqrt{70})$$

For larger values of n, the best way to evaluate the Gauss coefficients is to write a program. We shall use a program segment based on Sec. 1-5, to evaluate $P_n(\xi)$. We will need the recurrence Eq. 1-15, which is

$$P_{n+1}(\xi) = \frac{1}{n+1}\left[(2n+1)\xi P_n(\xi) - nP_{n-1}(\xi)\right] \tag{A-16}$$

We shall then find the root ξ_j of Eq. A-7 by using the Newton-Raphson method of Sec. 2-2. The weighting factor w_j will be found from Eq. A-15. There is one essential element that we do not yet have; an estimate of ξ_j is needed to start the iteration process. We can get this from the equation

$$\xi_j = \left(\frac{n-1}{n-.5}\right)^{1/2} \sin\left[\frac{\pi}{n}(j-.5)\right] \tag{A-17a}$$

n even, $j = 1,2,3, \ldots, \dfrac{n}{2}$

or

$$\xi_j = \left(\frac{n-1}{n-.5}\right)^{1/2} \sin\left[\frac{\pi}{n}(j-1)\right] \qquad \text{(A-17b)}$$

$$n \text{ odd}, \quad j = 1,2,3, \ldots, \frac{n+1}{2}$$

The number of iterations required for convergence to ten significant figures is $j + 3$.

The program follows. Line 1 is the title. Line 10 generates a blank line between successive sets of output. Line 20 calls for the value of n. Line 30 prints the headings for the output, which will appear as a table. Line 40 calculates a parameter L, which will be used to distinguish between even and odd values of n in equations A–17. Lines 50 through 190 constitute a triple FOR-NEXT loop that calculates and prints the values of the ξ_js and w_js. Line 60 calculates the starting estimate of ξ_j, using Eq. A-17. The innermost loop of lines 100 through 140 calculates $P_n(\xi_j)$, using Eq. A-16. The middle loop of lines 70 through 160 performs a Newton-Raphson iteration to find the value of ξ_j that satisfies Eq. A-7. Equation A-14 is used in line 150. The outer loop of lines 50 through 190 calculates w_j in line 170, using Eq. A-15, prints the final values of ξ_j and w_j, and repeats the entire process for all required values of j. Line 200 returns the execution to the beginning in preparation for further calculations with other values of n.

```
  1  REM: COEFFICIENTS FOR GAUSS INTEGRATION
 10  PRINT
 20  INPUT "N = ";N
 30  PRINT "   XI","   W"
 40  L=N/2-INT(N/2)
 50  FOR J=1 TO (N+1)/2
 60  X=SQR((N-1)/(N-.5))*SIN(3.141592654*(J-.5-L)/N)
 70  FOR K=1 TO J+3
 80  P0=1
 90  P1=X
100  FOR I=1 TO N-1
110  P2=((2*I+1)*X*P1-I*P0)/(I+1)
120  P0=P1
130  P1=P2
140  NEXT I
150  X=X-(1-X*X)*P1/N/(P0-X*P1)
160  NEXT K
170  W=2*(1-X*X)/(N*P0)^2
180  PRINT X,W
190  NEXT J
200  GOTO 10
```

Results found from the program agree with those given in the table of Sec. 3-2.

THE RUNGE-KUTTA METHOD

To derive the Runge-Kutta formulas, we start with Eq. 5-1, which is

$$y' = f(x,y) \tag{A-18}$$

The Taylor series expansion of y is

$$y_{i+1} = y_i + hy_i' + \frac{h^2}{2} y_i'' + \ldots \tag{A-19}$$

where $h = x_{i+1} - x_i$. This may be rewritten as

$$y_{i+1} = y_i + hf_i + \frac{h^2}{2}(f_{xi} + f_i f_{yi}) + \ldots \tag{A-20}$$

where $f_i = f(x_i, y_i)$ and the subscripts x and y denote partial derivatives. We assume an approximation of the form

$$y_{i+1} = y_i + \alpha_1 hf(x_i, y_i) + \alpha_2 hf(x_i + \beta_1 h, y_i + \beta_2 hf_i) \tag{A-21}$$

and proceed to determine the constants α_1, α_2, β_1, and β_2 so that a Taylor series expansion of the right side of Eq. A-21 will agree with the expansion Eq. A-20 through terms of second degree in h.

The Taylor series expansion of $f(x_i + \Delta_1, y_i + \Delta_2)$ is

$$f(x_i + \Delta_1, y_i + \Delta_2) = f_i + \Delta_1 f_{xi} + \Delta_2 f_i f_{yi} + \ldots$$

By setting $\Delta_1 = \beta_1 h$, $\Delta_2 = \beta_2 h$, and substituting the result into the last term on the right side of Eq. A-21, we find that

$$y_{i+1} = y_i + (\alpha_1 + \alpha_2)hf_i + \alpha_2 h^2(\beta_1 f_{xi} + \beta_2 f_i f_{yi}) \tag{A-22}$$

By equating coefficients of corresponding terms on the right sides of Eqs. A-20 and A-22, we find that

$$\alpha_1 + \alpha_2 = 1 \qquad \beta_1 = \beta_2 = \frac{1}{2\alpha_2}$$

There are only three equations for four unknowns, so we have some freedom in choosing the constants. The simplest equations are obtained by setting

$$\alpha_1 = \alpha_2 = \frac{1}{2} \qquad \beta_1 = \beta_2 = 1$$

Equation A-21 now becomes

$$y_{i+1} = y_i + \frac{h}{2}(q_1 + q_2) \tag{A-23a}$$

where

$$q_i = f(x_i, y_i) \qquad q_2 = f(x_i + h, y_i + q_i) \tag{A-23b,c}$$

More accurate formulas are obtained by considering higher-order terms in the Taylor series. By considering terms through h^4, we obtain Eqs. 5-2. More complete derivations can be found in books on numerical analysis. (See, for example, reference 5 or 10.)

THE ADAMS METHOD

We shall give a very simple derivation of the Adams method; more sophisticated derivations can be found in books on numerical analysis. We start by rewriting Eq. A-18 in integral form as

$$y_{i+1} = y_i + \int_{x_i}^{x_{i+1}} f[x, y(x)]dx = y_i + \int_{x_i}^{x_{i+1}} f(x)dx \tag{A-24}$$

The subsequent algebra can be simplified by taking the origin at the point $x = x_i$. This will not affect the generality of the results. Then Eq. A-24 becomes

$$y_{i+1} = y_i + \int_0^h f(x)dx \tag{A-25}$$

We need an approximate expression for $f(x)$. We choose the cubic polynomial

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{A-26}$$

Then Eq. A-25 becomes

$$y_{i+1} = y_i + a_0 h + \frac{1}{2} a_1 h^2 + \frac{1}{3} a_2 h^3 + \frac{1}{4} a_3 h^4 \tag{A-27}$$

We assume that values of y are available at the four points $x_i = 0$, $x_{i-1} = -h$, $x_{i-2} = -2h$, $x_{i-3} = -3h$. By fitting the polynomial of Eq. A-26 to these four points, we arrive at the set of simultaneous equations

$$f_i = a_0$$

$$f_{i-1} = a_0 - a_1 h + a_2 h^2 - a_3 h^3$$

$$f_{i-2} = a_0 - 2a_1 h + 4a_2 h^2 - 8a_3 h^3$$

$$f_{i-3} = a_0 - 3a_1 h + 9a_2 h^2 - 27a_3 h^3$$

where we have written f for $f(x)$. The solutions are

$$a_0 = f_i$$

$$a_1 = \frac{1}{h}\left(\frac{11}{6} f_i - 3f_{i-1} + \frac{3}{2} f_{i-2} - \frac{1}{3} f_{i-3}\right)$$

$$a_2 = \frac{1}{h^2}\left(f_i - \frac{5}{2} f_{i-1} + 2f_{i-2} - \frac{1}{2} f_{i-3}\right)$$

$$a_3 = \frac{1}{h^3}\left(\frac{1}{6} f_i - \frac{1}{2} f_{i-1} + \frac{1}{2} f_{i-2} - \frac{1}{6} f_{i-3}\right)$$

Substitution of these results into Eq. A-27 leads to

$$y_{i+1} = y_i + \frac{h}{24}(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}) \tag{A-28}$$

This is the predictor Eq. 5-7a.

Equation A-28 has been obtained by extrapolation. The more accurate corrector Eq. 5-7b is obtained by using values of f at the points $x_{i+1} = h$, $x_i = 0$, $x_{i-1} = -h$, $x_{i-2} = -2h$. The simultaneous equations are

$$f_{i+1} = a_0 + a_1 h + a_2 h^2 + a_3 h^3$$

$$f_i = a_0$$

$$f_{i-1} = a_0 - a_1 h + a_2 h^2 - a_3 h^3$$

$$f_{i-2} = a_0 - 2a_1 h + 4a_2 h^2 - 8a_3 h^3$$

The solutions are

$$a_0 = f_i$$

$$a_1 = \frac{1}{h}\left(\frac{1}{3} f_{i+1} - \frac{1}{2} f_i - f_{i-1} + \frac{1}{6} f_{i-2}\right)$$

$$a_2 = \frac{1}{h^2}\left(\frac{1}{2} f_{i+1} - f_i + \frac{1}{2} f_{i-1}\right)$$

$$a_3 = \frac{1}{h^3}\left(\frac{1}{6} f_{i+1} - \frac{1}{2} f_i + \frac{1}{2} f_{i-1} - \frac{1}{6} f_{i-2}\right)$$

Substitution of these results into Eq. A-27 leads to

$$y_{i+1} = y_1 + \frac{h}{24}(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}) \tag{A-29}$$

which is the desired result.

References

1. ABRAMOWITZ, M. AND I. A. STEGUN, *Handbook of Mathematical Functions*, National Bureau of Standards, Applied Mathematics Series 55, Washington, D.C.: U.S. Government Printing Office, 1964. (The first edition of this book contains a number of misprints. Most of these have been corrected in subsequent printings.)
2. BAUER, F. L., H. RUTISHAUSER, AND E. STIEFEL, "New Aspects in Numerical Quadrature," from *Proceedings of Symposia in Applied Mathematics*, 15, pp. 199-218, Providence, R.I.: American Mathematical Society, 1963.
3. FRANKLIN, P., *A Treatise on Advanced Calculus*, Chapter XVI, New York: Dover Publications, 1948.
4. HANCOCK, H., *Elliptic Integrals*, pp. 69, 81, New York: Dover Publications, 1917.
5. HENRICI, P., *Discrete Variable Methods in Ordinary Differential Equations*, New York: Wiley-Interscience, 1962.
6. HORNBECK, R. W., *Numerical Methods*, New York: Quantum Publishers, Inc., 1975.
7. MACROBERT, T. M., *Spherical Harmonics*, Chapter V (2nd ed.), New York: Dover Publications, 1948.

8. MECK, H. R., *Scientific Analysis for Programmable Calculators*, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1981.
9. NATIONAL BUREAU OF STANDARDS APPLIED MATHEMATICS SERIES 41, *Tables of the Error Function and its Derivatives* (2nd ed.), Washington, D.C.: U.S. Government Printing Office, 1954.
10. RALSTON, A. AND P. RABINOWITZ, *A First Course in Numerical Analysis* (2nd ed.), New York: McGraw-Hill, 1978.
11. STROUD, A. H. AND D. SECREST, *Gaussian Quadrature Formulas*, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1966.
12. TRANTER, C. J., *Integral Transforms in Mathematical Physics* (2nd ed), pp. 67-72, London: Methuen, 1956.

Suggested Solutions to Selected Problems

Chapter 1

1-3. There are many possible ways of programming this calculation. An example follows using relational expressions.

```
10 INPUT N
20 S=.1*N-.01*ABS((N>5)*(N-5)+(N>10)*(N-10)+(N>15)*
   (N-15))
30 PRINT S
```

1-5. a. The general term is

$$\frac{1}{[n(n+1)(n+2)]^2} = \frac{1}{[n(2+3n+n^2)]^2} = \frac{1}{[n(2+n(3+n)]^2}$$

A program follows:

```
10 S=0
20 J=1
30 PRINT
40 INPUT "N=";N
50 FOR J=J TO N
60 S=S+1/(J*(2+J*(3+J)))^2
70 NEXT J
```

```
80 PRINT "S=";S
90 GOTO 30
```

Varying levels of accuracy are obtained by entering increasing values of n as input. Some results are shown in the following table:

n	10	20	30
S	.0299001	.0299010	.0299011

The last result is correct to the full number of digits shown.

c. The nested format works very well with this series. We rewrite it as

$$S = 1 - \frac{1}{2}\left(\frac{1}{2} - \frac{1}{2}\left(\frac{1}{3} - \frac{1}{2}\left(\frac{1}{4} - \frac{1}{2}\left(\frac{1}{5} - \cdots\right)\right)\right)\right)$$

A program follows.

```
10 INPUT "N=";N
20 S=1/N
30 FOR J=N-1 TO 1 STEP -1
40 S=1/J-S/2
50 NEXT J
60 PRINT S
```

The number in line 20 is the number at the extreme right of the nested equation. The FOR-NEXT loop executes $n - 1$ cycles, proceeding from right to left and summing n terms of the original series. With $n = 30$, we obtain the result $S = .8109302162$, which is correct to ten significant figures.

1-6. In expanded form, the equation for the binomial coefficient is

$$\binom{p}{q} = \frac{p(p-1)(p-2)\cdots(p-q+1)}{q(q-1)(q-2)\cdots 1} \qquad p \geq q \geq 1$$

$$= 1 \qquad p \geq q = 0$$

A program follows. A GOTO statement has been included at the end so the program can be used repeatedly without entering RUN each time.

```
10 PRINT
20 INPUT "ENTER P,Q";P,Q
30 B=1
40 IF Q=0 THEN 80
50 FOR J=0 TO Q-1
60 B=(P-J)/(Q-J)*B
70 NEXT J
80 PRINT B
90 GOTO 10
```

1-7. The program may be revised as follows:

```
 45 J=3
 50 FOR J=J TO N
110 INPUT N
120 GOTO 50
130 DATA 1,2
```

1-11. A program follows:

```
10 INPUT X(1),X(2),X(3)
20 FOR J=1 TO 3
30 F(J)=SQR(3-X(J)*(5-X(J)*(2+X(J))))
40 NEXT J
50 Y=F(1)-3*F(2)+2*F(3)
60 PRINT Y
```

1-12. A program follows:

```
  1 REM: SORTING NUMBERS
 10 DIM X(100)
 20 PRINT
 30 N=N+1
 40 INPUT X(N)
 50 IF N=1 THEN 130
 60 FOR J=N TO 2 STEP -1
 70 IF X(J)<=X(J-1) THEN 110
 80 T=X(J)
 90 X(J)=X(J-1)
100 X(J-1)=T
110 PRINT X(J);",";
120 NEXT J
130 PRINT X(1)
140 GOTO 20
```

Chapter 2

2-1. 3.69344 1359 **2-2.** 1.91967 5341 **2-3.** .62401 75637
2-4. 2.61172 0144 **2-5.** ±1.89549 4267 **2-6.** ±.82376 78331
2-7. 1.16556 1185
2-8. The equation to be solved is

$$y = x^2 - 5 = 0$$

The Newton-Raphson equation is

$$x = x_0 - \frac{x_0^2 - 5}{2x_0} = \frac{1}{2}\left(x_0 + \frac{5}{x_0}\right)$$

The result is 2.23606 7978.
2-9. 1.70997 5947

2-12. a. −.24697 96037, 1.44504 1868, 2.80193 7736
b. 5.00526 5097, −2.50263 2549±.83036 67988i
c. 1.36880 8108, −1.68440 4054±3.43133 1350i

2-13. Using only the first subscript for the τs, the program may be revised as follows:

```
20 INPUT "ENTER SX,SY,SZ";SX,SY,SZ
30 INPUT "ENTER TX,TY,TZ";TX,TY,TZ
40 A=1
50 B=-SX-SY-SZ
60 C=SX*SY+SY*SZ+SZ*SX-TX*TX-TY*TY-TZ*TZ
70 D=SX*TY*TY+SY*TZ*TZ+SZ*TX*TX-SX*SY*SZ
   -2*TX*TY*TZ
```

Also, line 2 may be deleted.

For the input of the problem, the results are

−89.07759 661 55.95553 186 103.12206 47

This program can also be used for the calculation of principal moments of inertia. The tensile stresses correspond to moments of inertia and the shear stresses correspond to products of inertia with reversed signs.

2-15. a. $\pm\sqrt{2}$, $\frac{1}{2}(1 \pm \sqrt{5})$
b. $-1 \pm \sqrt{3}$, $\frac{1}{2}(1 \pm \sqrt{13})$
c. $\frac{1}{2}(1 \pm \sqrt{5})$, $1 \pm 2i$
d. $\frac{1}{2}(3 \pm \sqrt{5})$, $1 \pm i\sqrt{3}$

2-16. Rough preliminary calculations show that the equation has only two real roots, with approximate values 2.7 and −.7. By one of the iterative methods of this chapter, we obtain the improved values 2.73205 0808 and −.73205 0808. We then divide out the factor

$$(x + .73205\ 0808)(x - 2.73205\ 0808) = x^2 - 2x - 2$$

from the sixth-degree equation to obtain

$$x^4 + x^3 + 5x^2 + 5x + 12 = 0$$

which can be solved by the program of Sec. 2-7. The complete results are

$$1 \pm \sqrt{3} \quad -1 \pm i\sqrt{2} \quad \tfrac{1}{2}(1 \pm i\sqrt{15})$$

Chapter 3

3-3. The following amendments to the program for $E_1(x)$ are needed. We use P for n, because N has already been used in the program.

```
 20 INPUT "ENTER X,N";X,P
140 EP=E1
150 IF P=1 THEN 190
160 FOR J=2 TO P
170 EP=(EXP(-X) -X*EP)/(J-1)
180 NEXT J
```

```
190 PRINT EP
200 GOTO 10
```

3-4. The nested form of this equation is

$$\operatorname{erf} x = \frac{2}{\sqrt{\pi}}\, xe^{-x^2}\left(1+\frac{2x^2}{3}\left(1+\frac{2x^2}{5}\left(1+\frac{2x^2}{7}\left(1+\ldots\right)\right)\right)\right)$$

The following amendments to the program of Sec. 3-3 are needed:

```
40 S=1
60 S=1+2*X*X/(2*J+1)*S
80 ERF=X/SQR(ATN(1))/EXP(X*X)*S
```

3-6. This problem resembles the asymptotic evaluation of $E_1(x)$ in Sec. 3-2. The nested form of the equation is

$$\operatorname{erfc} x = \frac{e^{-x^2}}{\sqrt{\pi}\, x}\left(1-\frac{1}{2x^2}\left(1-\frac{3}{2x^2}\left(1-\frac{5}{2x^2}\left(1-\ldots\left(\frac{1}{2}\right)\right)\right)\right)\right)$$

A program follows:

```
 10 PRINT
 20 INPUT "X=";X
 30 N=INT(X*X+1.5)
 40 S=.5
 50 FOR J=N-1 TO 1 STEP -1
 60 S=1-(2*J-1)/2/X/X*S
 70 NEXT J
 80 T=S/SQR(ATN(1))/2
 90 PRINT "ERFC(X)=";T/X/EXP(X*X)
100 PRINT "XEXP(X*X)*ERFC(X)=";T
110 GOTO 10
```

3-7. The following substitutions may be used:

a. $x = b \sin\theta$ b. $x = b\cos\theta$ c. $x = b\tan\theta$
d. $x = (a^2\cos^2\theta + b^2\sin^2\theta)^{1/2}$

3-8. The required amendment follows. In this case the nested format is more complicated—and less efficient—than direct summation.

```
110 S=1+2*S/Q
130 E=P*(1-K*K*R*S/2)
```

With some computers, the amended program generates an overflow for small values of k.

3-9. This problem resembles Prob. 1-6. However, it cannot be solved by the same method, because p and q are not necessarily integers. The best procedure is to use the factorial program of Sec. 3-5 as a subroutine to evaluate $(p-1)!$, $(q-1)!$, and $(p+q-1)!$. A program follows:

```
  1 REM: BETA FUNCTION
 10 PRINT
 20 INPUT "ENTER P,Q ";P,Q
 30 Z=P-1
 40 GOSUB 160
 50 U1=U
 60 Z=Q-1
 70 GOSUB 160
 80 U2=U
 90 Z=P+Q-1
100 GOSUB 160
110 U3=U
120 PRINT "P=";P
130 PRINT "Q=";Q
140 PRINT "B(P,Q)";U1*U2/U3
150 GOTO 10
160 R=1
170 IF Z>=5 THEN 210
180 Z=Z+1
190 R=Z*R
200 GOTO 170
210 S=1/99
220 FOR J=1 TO 4
230 READ C
240 S=1/C-S/Z/Z
250 NEXT J
260 RESTORE
270 T=(Z+.5)*LOG(Z)-Z+.5*LOG(8*ATN(1))+S/Z/12
280 U=EXP(T)/R
290 RETURN
300 DATA 140,105,30,1
```

3-10. We rewrite the equation in nested form as

$$H_p(x) = \frac{\frac{2}{\pi}x^{p+1}}{1\cdot 3\cdot 5\cdot\ \ldots\ \cdot(2p+1)}\left(1-\frac{\left(\frac{x}{2}\right)^2}{\frac{3}{2}\left(p+\frac{3}{2}\right)}\left(1-\frac{\left(\frac{x}{2}\right)^2}{\frac{5}{2}\left(p+\frac{5}{2}\right)}\left(1-\ldots\right)\right)\right)$$

A program follows. It is organized in the same way as the program for $J_p(x)$ in Sec. 3-6.

```
  1 REM: STRUVE FUNCTION HP(X)
 10 PRINT
 20 INPUT "ENTER P,X ";P,X
 30 N=INT(2*X+5)
 40 S=1
 50 FOR J=N-1 TO 1 STEP -1
```

```
 60 S=1-X*X/4/(J+.5)/(P+J+.5)*S
 70 NEXT J
 80 FOR J=0 TO P
 90 S=S/(2*J+1)
100 NEXT J
110 PRINT "P=";P
120 PRINT "X=";X
130 PRINT "HP(X)=";S/2/ATN(1)*X^(P+1)
140 GOTO 10
```

Chapter 4

4-9. See Prob. 1-9.

4-10. The integrand is infinite at the lower limit. Integrate by parts to show that

$$\int_0^{\pi/2} \ln \sin x \, dx = -\int_0^{\pi/2} \frac{x}{\tan x} \, dx$$

The new integral is proper. It has been evaluated in Sec. 4-2.

4-11. The integrand is infinite at the upper limit. By writing $\pi/2 - x$ for x, show that

$$\int_0^{\pi} x \ln \sin x \, dx = \pi \int_0^{\pi/2} \ln \sin x \, dx$$

Then use the result of Prob. 4-10.

4-12-14. Gauss-Chebyshev integration works very well for these three integrals.

4-17. We observe that

$$\int_0^{\infty} \frac{x \, dx}{e^{\alpha x} - 1} = \frac{1}{\alpha^2} \int_0^{\infty} \frac{x \, dx}{e^x - 1}$$

The desired results now follow from Prob. 4-16.

4-18. A numerical evaluation is not necessary; we observe that

$$\int_0^{\infty} \frac{x \, dx}{e^x + 1} = \int_0^{\infty} \frac{x \, dx}{e^x - 1} - \int_0^{\infty} \frac{2x \, dx}{e^{2x} - 1} = \frac{1}{2} \int_0^{\infty} \frac{x \, dx}{e^x - 1}$$

The desired result now follows from Prob. 4-16.

4-19. This integral can be evaluated numerically as it stands, since it is proper. However, the process converges very slowly. It is preferable to write $e^{-x/2}$ for x, then use the result of Prob. 4-16.

4-20. This integral can be evaluated by writing $\tan x$ for x, but the resulting program has a long running time due to a long subroutine and slow convergence. It is preferable to start with the elementary identity

$$\int_0^{\infty} \left[\frac{1}{e^x - 1} - \frac{1}{x(x+1)} \right] dx = \ln \left[\frac{x+1}{x} \left(1 - e^{-x} \right) \right]_0^{\infty} = 0$$

The desired integral now becomes

$$I = \int_0^\infty \left(\frac{1}{1+x} - e^{-x} \right) \frac{dx}{x}$$

We break the interval at the point $x = 1$ and write $1/x$ for x in the second of the new integrals. This leads to

$$I = \int_0^1 (1 - e^{-x} - e^{-1/x}) \frac{dx}{x}$$

which is well suited to numerical integration.

4-21. The substitution $t = \tan \theta$ leads to

$$E_1(x) = \int_{\arctan x}^{\pi/2} \frac{e^{-\tan \theta}}{\sin \theta \cos \theta} d\theta$$

$$= \int_{\arctan x}^{\pi/2} \frac{2e^{-\tan \theta}}{\sin 2\theta} d\theta$$

It follows that

$$xe^x E_1(x) = \int_{\arctan x}^{\pi/2} \frac{2xe^{x-\tan \theta}}{\sin 2\theta} d\theta$$

There is a conflict between the nomenclature of this chapter and that of Chapter 3; the variable of integration is now x. Instead of typing in the value of the arc tangent for each value of x, we edit this conversion into the program. The new lines are

```
 42  INPUT "X=";C
 44  A=ATN(C)
200  Y=2*C*EXP(C-TAN(X))/SIN(2*X)
230  DATA 0,1.570796327
```

The first entry in line 230 is immaterial, since this is read as a in line 10 and then replaced by an assignment in line 44. For $x = 2$, we obtain the following results:

m	1	2	3	4
I	.772673	.772657229	.772657223	.772657234

The last result is correct to the full number of digits shown. Since the value of x is entered by an INPUT statement, the program can be used for other values of x without further editing by simply pressing the BREAK key, then entering RUN.

Chapter 5 **5-13.** The program may be amended as follows:

```
248  FOR K=1 TO 2
260  Y=Y0+H*(9*Q+19*F0-5*F1+F2)/24
262  NEXT K
264  Y0=Y
```

5-14. The following changes may be added to those of Prob. 5-13.

```
246 YP=Y
264 Y0=(251*Y+19*YP)/270
```

5-23. The program may be amended as follows:

```
 55 Q = -1/3
145 IF X=0 THEN 240
410 Q= -2*U/X-Y^5
430 DATA 0,1,0
```

5-26. Observe that there is a singular point at $x = 0$. Follow the procedure used for Eq. 5-16, first showing that $y_0'' = -\frac{1}{2}$.

5-33. The easiest way to accomplish this is to start with the program of Sec. 5-2 for two simultaneous equations. The following revisions are needed:

```
 10 READ X0,Y0,U0,V0,W0
 92 SC=0
 94 SD=0
152 V=V0+E*QC
154 W=W0+E*QD
162 QB=V
164 QC=W
170 QD=2*W-3*V+5*U-3*Y
192 SC=SC+C*QC
194 SD=SD+C*QD
232 V0=V0+H*SC/6
234 W0=W0+H*SD/6
260 DATA 0,1,1,1,1
```

Index

Technical

If scientific and numerical analysis is an integral part of your profession, your whole day shouldn't be spent on calculations!

Now, with this book, you'll learn the sound and speedy programming principles and computational techniques that let you successfully perform everyday numerical analysis tasks on almost any microcomputer—all using the simplified, practical form of BASIC shown here! From the fundamental to the highly complex, you can now use your computer to solve

- **roots of equations**
- **transcendental functions**
- **numerical integration**
- **matrices and simultaneous equations**
- **differential equations**

With these techniques, plus the ready-to-run BASIC programs, practice problems, and suggested programming solutions in the book, learning and doing numerical analysis can be as easy as you've always wished.

H. R. MECK received his B.S. from Lehigh University and a masters degree from Harvard. He first pursued a career in industry and government solving engineering problems in atomic energy, then went on to become a freelance writer. The author of many published journal articles in the field of solid mechanics, Mr. Meck also wrote the successful book *Scientific Analysis for Programmable Calculators.*

Cover design by Linda Huber

PRENTICE-HALL, Inc., Englewood Cliffs, New Jersey 07632

P

ISBN 0-13-626631-2