Compiler diskette instructions

just copy this diskette to your harddrive. You DO NOT NEED TO RUN THE INSTALL PROGRAM. I suggest you create a subdirectory called FORTRAN IN WHICH YOU HAVE YOUR COMPILER AND THE programs from the diskette in the back of the book.

to run a fortran program that is on the diskette from the book:

1) use at the > prompt: fl /c program name.for
   This compiles the program

   EXAMPLE: $fortran/> fl /c search.for

2) then at the next > prompt type: link program name.obj

   EXAMPLE: $fortran/> link search.obj

3) then do several carriage returns at run file, list file and lib file lines
4) then type in the name that appears on the run file line (that is the default name of the executable file), that is at the > prompt: type in the program name

   EXAMPLE: $fortran/> search

5) input the data as required.

BEFORE you run the program check to see if there is an $INCLUDE command at the end of the file. If so line number one must be altered at the prompt to read:

>fl /c program name.for /i name of include file

   EXAMPLE: $fortran/> fl /c examp2-4.for /i diff.sub

points 2 3 4 and 5 remain the same as before.

README.FOR

USING THE FORTRAN COMPUTER PROGRAMS

A backup copy of this diskette should be made before attempting to compile and link any of the programs to obtain executable programs. If your PC computer has a hard disk drive, all programs, subroutine subprograms, and data files on this diskette should be copied to a directory <DIR> containing the FORTRAN compiler software.

To facilitate the use of these programs along with the text, the user may find it helpful to store the files by chapters in SUBDIRECTORIES of a main DIRECTORY. The use of files on this diskette as they are used by chapter is outlined at the end of this file README.FOR.

The programs are written using FORTRAN 77, and can be used on any computer that is IBM compatible. In using some versions of FORTRAN, it may be necessary to modify some of the statements slightly. The programs have been made user friendly by a generous use of comment statements in the program listings which provide the algorithms that should help the user follow the general programming steps used.

The programs can be easily modified by using some type of editing program such as is available in word-processing software. When it is desired to temporarily replace a statement in the program, such a statement can be bypassed by simply typing C in column 1, as in a comment statement, and then typing in the temporary statement. Return to the original program is then made by deleting the C and the temporary statement. The use of a batch program (created by user) can facilitate the compiling and linking of a source program by the FORTRAN software being used to obain the executable program. If the files have been stored by chapters in SUBDIRECTORIES, the batch program should be in each SUBDIRECTORY for compiling and linking source programs listed in that particular SUBDIRECTORY. The data to be entered for running an executable program are requested on the console screen by WRITE statements in the program.

MAIN PROGRAMS AND SUBPROGRAMS

All of the main programs in the text that are in file on the diskette have the extension FOR on their names. All of the subroutine subprograms which are in file on the diskette have the extension SUB on their names. The subroutine subprograms shown in the text and listed in the index that do not have the extension SUB will be found listed following the main programs in which they are called, and are on the diskette in the files of these main programs. The subroutine subprograms in file are compiled and linked with the main programs by an $INCLUDE statement. For example, the metacommand $INCLUDE:'NAME.SUB', beginning in column 1, would be at the end of the calling program. If the files have been stored by chapters in SUBDIRECTORIES, the subprograms in files with the extension SUB need to be stored in every chapter where they are used with the $INCLUDE statement.

DATA FILES

In some of the programs, data is entered by a READ statement in which the Logical Unit Specifier is the number of the data file to be read. The data files referred to in the text which are in file on this diskette are as follows:

FORT03.DAT	
FORT04.DAT	Hold permanent tables of data
FORT05.DAT	for the GAUSSQD.FOR program.
FORT06.DAT	
FORT07.DAT	
FORT10.DAT	General purpose data file.

The data files above with Logical Unit Specifier numbers 3 through 7 must be maintained as is on the diskette for input to the GAUSSQD.FOR program (Gaussian quadrature integration). The data file with the Logical Unit Specifier 10 can be used and changed as needed for different problems.

OPEN Statements

The OPEN statements associated with READ and WRITE statements in the programs are as follows:

```
OPEN(2,FILE='PRN')          (Printer)
OPEN(n,FILE='FORTn.DAT')    (Data Files)
```

where n is the Logical Unit Specifier number for data files previously shown.

The use of an asterisk * as the Logical Unit Specifier in READ and WRITE statements specifies, respectively, the keyboard and console screen. The use of an asterisk * in place of a FORMAT statement number means free format.

LIST OF FORTRAN FILE NAMES

The file names of the FORTRAN programs and subprograms in the text and on this diskette by chapters are as follows:

Chapter 1

BISECT.FOR	NEWTNR.SUB
BAIRSTOW.FOR	
EXAMP2-4.FOR	
NRPOLYRT.FOR	
SEARCH.FOR	
NEWTRAPH.FOR	

Chapter 2

FADDEEV.FOR		CHLSKY.SUB	FORT10.DAT
MATINV.FOR		TRIDI.SUB	
DEFLATE.FOR	(a)	JACOBI.SUB	

IGAUSSJO.FOR
GAUSSEL.FOR
JACOBI.FOR

(a) The subroutine subprogram JACOBI (in file JACOBI.SUB) for use on standard eigenvalue problems is not shown in the text but is on this diskette.

Chapter 3 and 4

LSTSQR.FOR	CHLSKY.SUB	FORT10.DAT
EXPON.FOR	TRIDI.SUB	
SPLINE.FOR		

Chapter 4

SIMPSON.FOR	DIFF.SUB	(b) FORT03.DAT
DIFFER.FOR	TRAPZ.SUB	(b) FORT04.DAT
GAUSSQD.FOR		(b) FORT05.DAT
KINEMAT.FOR		(b) FORT06.DAT
ROMBERG.FOR		(b) FORT07.DAT
		FORT10.DAT

(b) Data files which must be maintained as is for use with the program GAUSSQD.FOR (Gaussian quadrature integration).

Chapter 5 and 6

RATTRAP.FOR	MEULER.SUB
VIB.FOR	RKSFX4.SUB
VANDERPL.FOR	ADMOUL.SUB
PARTROOP.FOR	START.SUB
(c)	RKSFX5.SUB

(c) The subroutine subprogram RKSFX5 (in file RKSFX5.SUB) for the fifth-order Runge-Kutta method is not in the text but is on this diskette.

Chapter 6

HEATRAD.FOR	RKSFX4.SUB
BEAMDEFL.FOR	CHLSKY.SUB

Chapter 7-9

PLATETMP.FOR	TRIDI.SUB
STRNGVIB.FOR	
RODTEMP.FOR	
RODTEMP2.FOR	

Elementary Numerical Analysis by Conte & de Boor

CHAPTER
ONE

NUMBER SYSTEMS AND ERRORS

In this chapter we consider methods for representing numbers on computers and the errors introduced by these representations. In addition, we examine the sources of various types of computational errors and their subsequent propagation. We also discuss some mathematical preliminaries.

1.1 THE REPRESENTATION OF INTEGERS

In everyday life we use numbers based on the decimal system. Thus the number 257, for example, is expressible as

$$\begin{aligned} 257 &= 2 \cdot 100 + 5 \cdot 10 + 7 \cdot 1 \\ &= 2 \cdot 10^2 + 5 \cdot 10^1 + 7 \cdot 10^0 \end{aligned}$$

We call 10 the **base** of this system. Any integer is expressible as a polynomial in the base 10 with integral coefficients between 0 and 9. We use the notation

$$\begin{aligned} N &= (a_n a_{n-1} \cdots a_0)_{10} \\ &= a_n 10^n + a_{n-1} 10^{n-1} + \cdots + a_0 10^0 \end{aligned} \tag{1.1}$$

to denote any positive integer in the base 10. There is no intrinsic reason to use 10 as a base. Other civilizations have used other bases such as 12, 20, or 60. Modern computers read pulses sent by electrical components. The state of an electrical impulse is either *on* or *off*. It is therefore convenient to represent numbers in computers in the **binary** system. Here the base is 2, and the integer coefficients may take the values 0 or 1.

A nonnegative integer N will be represented in the binary system as

$$N = (a_n a_{n-1} \cdots a_1 a_0)_2$$
$$= a_n 2^n + a_{n-1} 2^{n-1} + \cdots + a_1 2^1 + a_0 2^0 \qquad (1.2)$$

where the coefficients a_k are either 0 or 1. Note that N is again represented as a polynomial, but now in the base 2. Many computers used in scientific work operate internally in the binary system. Users of computers, however, prefer to work in the more familiar decimal system. It is therefore necessary to have some means of converting from decimal to binary when information is submitted to the computer, and from binary to decimal for output purposes.

Conversion of a binary number to decimal form may be accomplished directly from the definition (1.2). As examples we have

$$(11)_2 = 1 \cdot 2^1 + 1 \cdot 2^0 = 3$$
$$(1101)_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13$$

The conversion of integers from a base β to the base 10 can also be accomplished by the following algorithm, which is derived in Chap. 2.

Algorithm 1.1 Given the coefficients $a_n, \ldots, a_0$ of the polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \qquad (1.3)$$

and a number β. Compute recursively the numbers $b_n, b_{n-1}, \ldots, b_0$:

$$b_n := a_n$$
$$b_{n-1} := a_{n-1} + b_n \beta$$
$$b_{n-2} := a_{n-2} + b_{n-1} \beta$$
$$b_{n-3} := a_{n-3} + b_{n-2} \beta$$
$$\cdots\cdots\cdots\cdots\cdots$$
$$b_0 := a_0 + b_1 \beta$$

Then $b_0 = p(\beta)$.

Since, by the definition (1.2), the binary integer $(a_n a_{n-1} \cdots a_0)_2$ represents the value of the polynomial (1.3) at $x = 2$, we can use Algorithm 1.1, with $\beta = 2$, to find the decimal equivalents of binary integers.

Thus the decimal equivalent of $(1101)_2$ computed using Algorithm 1.1 is

$$b_3 = 1$$
$$b_2 = 1 + 1 \cdot 2 = 3$$
$$b_1 = 0 + 3 \cdot 2 = 6$$
$$b_0 = 1 + 6 \cdot 2 = 13$$

and the decimal equivalent of $(10000)_2$ is

$$b_4 = 1$$
$$b_3 = 0 + 1 \cdot 2 = 2$$
$$b_2 = 0 + 2 \cdot 2 = 4$$
$$b_1 = 0 + 4 \cdot 2 = 8$$
$$b_0 = 0 + 8 \cdot 2 = 16$$

Converting a decimal integer N into its binary equivalent can also be accomplished by Algorithm 1.1 if one is willing to use *binary* arithmetic. For if $N = (a_n a_{n-1} \ldots a_0)_{10}$, then by the definition (1.1), $N = p(10)$, where $p(x)$ is the polynomial (1.3). Hence we can calculate the binary representation for N by translating the coefficients $a_n, \ldots, a_0$ into binary integers and then using Algorithm 1.1 to evaluate $p(x)$ at $x = 10 = (1010)_2$ in binary arithmetic. If, for example, $N = 187$, then

$$187 = (187)_{10} = 1 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0$$
$$= (1)_2 (1010)_2^2 + (1000)_2 (1010)_2^1 + (111)_2 (1010)_2^0$$

and using Algorithm 1.1 and binary arithmetic,

$$b_2 = (1)_2$$
$$b_1 = (1000)_2 + (1)_2 (1010)_2 = (1000)_2 + (1010)_2 = (10010)_2$$
$$b_0 = (111)_2 + (10010)_2 (1010)_2 = (111)_2 + (10110100)_2 = (10111011)_2$$

Therefore $187 = (10111011)_2$.

Binary numbers and binary arithmetic, though ideally suited for today's computers, are somewhat tiresome for people because of the number of digits necessary to represent even moderately sized numbers. Thus eight binary digits are necessary to represent the three-decimal-digit number 187. The **octal** number system, using the base 8, presents a kind of compromise between the computer-preferred binary and the people-preferred decimal system. It is easy to convert from octal to binary and back since three binary digits make one octal digit. To convert from octal to binary, one merely replaces all octal digits by their binary equivalent; thus

$$(347)_8 = (011 \quad 100 \quad 111)_2 = (11100111)_2$$

Conversely, to convert from binary to octal, one partitions the binary digits in groups of three (starting from the right) and then replaces each three-group by its octal digit; thus

$$(10111011)_2 = (010 \quad 111 \quad 011)_2 = (273)_8$$

If a decimal integer has to be converted to binary by hand, it is usually fastest to convert it first to octal using Algorithm 1.1, and then from octal to binary. To take an earlier example,

$$187 = (187)_{10} = (1)_8 (12)_8^2 + (10)_8 (12)_8^1 + (7)_8 (12)_8^0$$

Hence, using Algorithm 1.1 [with 2 replaced by $10 = (12)_8$, and with *octal* arithmetic],

$$b_2 = (1)_8$$
$$b_1 = (10)_8 + (1)_8(12)_8 = (22)_8$$
$$b_0 = (7)_8 + (22)_8(12)_8 = (7)_8 + (264)_8 = (273)_8$$

Therefore, finally,

$$187 = (273)_8 = (010111011)_2$$

EXERCISES

1.1-1 Convert the following binary numbers to decimal form:
$(1010)_2$ $(100101)_2$ $(10000001)_2$

1.1-2 Convert the following decimal numbers to binary form:
82, 109, 3433

1.1-3 Carry out the conversions in Exercises 1.1-1 and 1.1-2 by converting first to octal form.

1.1-4 Write a FORTRAN subroutine which accepts a number to the base BETIN with the NIN digits contained in the one-dimensional array NUMIN, and returns the NOUT digits of the equivalent in base BETOUT in the one-dimensional array NUMOUT. For simplicity, restrict both BETIN and BETOUT to 2, 4, 8, and 10.

1.2 THE REPRESENTATION OF FRACTIONS

If x is a positive real number, then its **integral** part x_I is the largest integer less than or equal to x, while

$$x_F = x - x_I$$

is its **fractional** part. The fractional part can always be written as a *decimal fraction*:

$$x_F = \sum_{k=1}^{\infty} b_k 10^{-k} \tag{1.4}$$

where each b_k is a nonnegative integer less than 10. If $b_k = 0$ for all k greater than a certain integer, then the fraction is said to *terminate*. Thus

$$\tfrac{1}{4} = 0.25 = 2 \cdot 10^{-1} + 5 \cdot 10^{-2}$$

is a terminating decimal fraction, while

$$\tfrac{1}{3} = 0.333 \cdots = 3 \cdot 10^{-1} + 3 \cdot 10^{-2} + 3 \cdot 10^{-3} + \cdots$$

is not.

If the integral part of x is given as a decimal integer by

$$x_I = (a_n a_{n-1} \cdots a_0)_{10}$$

while the fractional part is given by (1.4), it is customary to write the two representations one after the other, separated by a point, the "decimal point":

$$x = (a_n a_{n-1} \cdots a_0.b_1 b_2 b_3 \cdots)_{10}$$

Completely analogously, one can write the fractional part of x as a *binary* fraction:

$$x_F = \sum_{k=1}^{\infty} b_k 2^{-k}$$

where each b_k is a nonnegative integer less than 2, i.e., either zero or one. If the integral part of x is given by the binary integer

$$x_I = (a_n a_{n-1} \cdots a_0)_2$$

then we write

$$x = (a_n a_{n-1} \cdots a_0.b_1 b_2 b_3 \cdots)_2$$

using a "binary point."

The binary fraction $(.b_1 b_2 b_3 \cdots)_2$ for a given number x_F between zero and one can be calculated as follows: If

$$x_F = \sum_{k=1}^{\infty} b_k 2^{-k}$$

then
$$2x_F = \sum_{k=1}^{\infty} b_k 2^{-k+1} = b_1 + \sum_{k=1}^{\infty} b_{k+1} 2^{-k}$$

Hence b_1 is the integral part of $2x_F$, while

$$(2x_F)_F = 2x_F - b_1 = \sum_{k=1}^{\infty} b_{k+1} 2^{-k}$$

Therefore, repeating this procedure, we find that b_2 is the integral part of $2(2x_F)_F$, b_3 is the integral part of $2(2(2x_F)_F)_F$, etc.

If, for example, $x = 0.625 = x_F$, then

$$2(0.625) = 1.25 \qquad \text{so } b_1 = 1$$
$$2(0.25) = 0.5 \qquad \text{so } b_2 = 0$$
$$2(0.5) = 1.0 \qquad \text{so } b_3 = 1$$

and all further b_k's are zero. Hence

$$0.625 = (.101)_2$$

This example was rigged to give a terminating binary fraction. Unhappily, not every terminating decimal fraction gives rise to a terminating binary fraction. This is due to the fact that the binary fraction for

$x_F = 10^{-1} = 0.1$ is not terminating. We have

$$\begin{aligned} 2(0.1) &= 0.2 \quad \text{so } b_1 = 0 \\ 2(0.2) &= 0.4 \quad \text{so } b_2 = 0 \\ 2(0.4) &= 0.8 \quad \text{so } b_3 = 0 \\ 2(0.8) &= 1.6 \quad \text{so } b_4 = 1 \\ 2(0.6) &= 1.2 \quad \text{so } b_5 = 1 \end{aligned}$$

and now we are back to a fractional part of 0.2, so that the digits cycle. It follows that

$$0.1 = (.0 \quad 0011 \quad 0011 \cdots)_2$$

The procedure just outlined is formalized in the following algorithm.

Algorithm 1.2 Given x between 0 and 1 and an integer β greater than 1. Generate recursively $b_1, b_2, b_3 \ldots$ by

$$\begin{aligned} c_0 &:= x \\ b_1 &:= (\beta c_0)_I, \qquad c_1 := (\beta c_0)_F \\ b_2 &:= (\beta c_1)_I, \qquad c_2 := (\beta c_1)_F \\ &\cdots\cdots\cdots\cdots\cdots \end{aligned}$$

Then

$$x = (.b_1 b_2 b_3 \cdots)_\beta = \sum_{k=1}^{\infty} b_k \beta^{-k}$$

We have stated this algorithm for a general base β rather than for the specific binary base $\beta = 2$, for two reasons. If this conversion to binary is carried out with pencil and paper, it is usually faster to convert first to octal, i.e., use $\beta = 8$, and then to convert from octal to binary. Also, the algorithm can be used to convert a binary (or octal) fraction to decimal, by choosing $\beta = 10$ and using binary (or octal) arithmetic.

To give an example, if $x = (.101)_2$, then, with $\beta = 10 = (1010)_2$ and binary arithmetic, we get from Algorithm 1.2

$$\begin{aligned} 10(.101)_2 &= (110.010)_2 \quad \text{so } b_1 = (110)_2 = 6, \quad c_1 = (.01)_2 \\ 10(.01)_2 &= (10.10)_2 \quad \text{so } b_2 = (10)_2 = 2, \quad c_2 = (.1)_2 \\ 10(.1)_2 &= (101.)_2 \quad \text{so } b_3 = (101)_2 = 5, \quad c_3 = 0 \end{aligned}$$

Hence subsequent b_k's are zero. This shows that

$$(.101)_2 = 0.625$$

confirming our earlier calculation. Note that if x_F is a terminating binary fraction with n digits, then it is also a terminating decimal fraction with n digits, since

$$(.1)_2 = 0.5$$

EXERCISES

1.2-1 Convert the following binary fractions to decimal fractions:
$(.110\dot{0}0\dot{1}1)_2$ $(.1\dot{1}1\dot{1}11\dot{1}1)_2$

1.2-2 Find the first 5 digits of .1 written as an octal fraction, then compute from it the first 15 digits of .1 as a binary fraction.

1.2-3 Convert the following octal fractions to decimal:
$(.614)_8$ $(.776)_8$
Compare with your answer in Exercise 1.2-1.

1.2-4 Find a binary number which approximates π to within 10^{-3}.

1.2-5 If we want to convert a decimal integer N to binary using Algorithm 1.1, we have to use *binary* arithmetic. Show how to carry out this conversion using Algorithm 1.2 and *decimal* arithmetic. (*Hint:* Divide N by the appropriate power of 2, convert the result to binary, then shift the "binary point" appropriately.)

1.2-6 If we want to convert a terminating binary fraction x to a decimal fraction using Algorithm 1.2, we have to use *binary* arithmetic. Show how to carry out this conversion using Algorithm 1.1 and *decimal* arithmetic.

1.3 FLOATING-POINT ARITHMETIC

Scientific calculations are usually carried out in floating-point arithmetic.

An n-digit floating-point number in base β has the form

$$x = \pm (.d_1 d_2 \cdots d_n)_\beta \beta^e \tag{1.5}$$

where $(.d_1 d_2 \cdots d_n)_\beta$ is a β-fraction called the **mantissa**, and e is an integer called the **exponent**. Such a floating-point number is said to be **normalized** in case $d_1 \neq 0$, or else $d_1 = d_2 = \cdots = d_n = 0$.

For most computers, $\beta = 2$, although on some, $\beta = 16$, and in hand calculations and on most desk and pocket calculators, $\beta = 10$.

The precision or length n of floating-point numbers on any particular computer is usually determined by the word length of the computer and may therefore vary widely (see Fig. 1.1). Computing systems which accept FORTRAN programs are expected to provide floating-point numbers of two different lengths, one roughly double the other. The shorter one, called **single precision**, is ordinarily used unless the other, called **double precision**, is specifically asked for. Calculation in double precision usually doubles the storage requirements and more than doubles running time as compared with single precision.

Figure 1.1 Floating-point characteristics.

Computer	β	n	$M = -m$
IBM 7094	2	27	2^7
Burroughs 5000 Series	8	13	2^6
IBM 360/370	16	6	2^6
CDC 6000 and Cyber Series	2	48	2^{10}
DEC 11/780 VAX	2	24	2^7
Hewlett Packard 67	10	10	99

The exponent e is limited to a range

$$m < e < M \tag{1.6}$$

for certain integers m and M. Usually, $m = -M$, but the limits may vary widely; see Fig. 1.1.

There are two commonly used ways of translating a given real number x into an n β-digit floating-point number $fl(x)$, rounding and chopping. In **rounding**, $fl(x)$ is chosen as the normalized floating-point number nearest x; some special rule, such as symmetric rounding (rounding to an even digit), is used in case of a tie. In **chopping**, $fl(x)$ is chosen as the nearest normalized floating-point number *between* x and 0. If, for example, two-decimal-digit floating-point numbers are used, then

$$fl\left(\tfrac{2}{3}\right) = \begin{cases} (0.67)10^0 & \text{rounded} \\ (0.66)10^0 & \text{chopped} \end{cases}$$

and

$$fl(-838) = \begin{cases} -(0.84)10^3 & \text{rounded} \\ -(0.83)10^3 & \text{chopped} \end{cases}$$

On some computers, this definition of $fl(x)$ is modified in case $|x| \geq \beta^M$ (**overflow**) or $0 < |x| \leq \beta^{m-n}$ (**underflow**), where m and M are the bounds on the exponents; either $fl(x)$ is not defined in this case, causing a stop, or else $fl(x)$ is represented by a special number which is not subject to the usual rules of arithmetic when combined with ordinary floating-point numbers.

The difference between x and $fl(x)$ is called the **round-off error**. The round-off error depends on the size of x and is therefore best measured relative to x. For if we write

$$fl(x) = x(1 + \delta) \tag{1.7}$$

where $\delta = \delta(x)$ is some number depending on x, then it is possible to *bound* δ *independently* of x, at least as long as x causes no overflow or underflow. For such an x, it is not difficult to show that

$$|\delta| < \tfrac{1}{2}\beta^{1-n} \quad \text{in rounding} \tag{1.8}$$

See Exercise 1.3-3. The maximum possible value for $|\delta|$ is often called the **unit roundoff** and is denoted by u.

When an arithmetic operation is applied to two floating-point numbers, the result usually fails to be a floating-point number of the same length. If, for example, we deal with two-decimal-digit numbers and

$$x = (0.20)10^1 = 2 \qquad y = (0.77)10^{-6} \qquad z = (0.30)10^1 = 3$$

then

$$x + y = (0.200000077)10^1 \qquad x \cdot y = (0.154)10^{-5}$$

$$\frac{x}{z} = (0.666\cdots)10^0$$

Hence, if ω denotes one of the arithmetic operations (addition, subtraction, multiplication, or division) and ω^* denotes the floating-point operation of the same name provided by the computer, then, however the computer may arrive at the result $x\omega^* y$ for two given floating-point numbers x and y, we can be sure that usually

$$x\omega^* y \neq x\omega y$$

Although the floating-point operation ω^* corresponding to ω may vary in some details from machine to machine, ω^* is usually constructed so that

$$x\omega^* y = fl(x\omega y) \tag{1.10}$$

In words, the floating-point sum (difference, product, or quotient) of two floating-point numbers usually equals the floating-point number which represents the exact sum (difference, product, or quotient) of the two numbers. Hence (unless overflow or underflow occurs) we have

$$x\omega^* y = (x\omega y)(1 + \delta) \qquad \text{for some } \delta \text{ with } |\delta| \leq u \tag{1.11a}$$

where u is the unit roundoff. In certain situations, it is more convenient to use the equivalent formula

$$x\omega^* y = (x\omega y)/(1 + \delta) \qquad \text{for some } \delta \text{ with } |\delta| \leq u \tag{1.11b}$$

Equation (1.11) expresses the basic idea of backward error analysis (see J. H. Wilkinson [24]†). Explicitly, Eq. (1.11) allows one to interpret a floating-point result as the result of the corresponding ordinary arithmetic, but performed on slightly perturbed data. In this way, the analysis of the effect of floating-point arithmetic can be carried out in terms of ordinary arithmetic.

For example, the value of the function $f(x) = x^{2^n}$ at a point x_0 can be calculated by n squarings, i.e., by carrying out the sequence of steps

$$x_1 := x_0^2, x_2 := x_1^2, \ldots, x_n := x_{n-1}^2$$

with $f(x_0) = x_n$. In floating-point arithmetic, we compute instead, according to Eq. (1.11a), the sequence of numbers

$$\hat{x}_1 = x_0^2(1 + \delta_1), \hat{x}_2 = (\hat{x}_1)^2(1 + \delta_2), \ldots, \hat{x}_n = (\hat{x}_{n-1})^2(1 + \delta_n)$$

†Numbers in brackets refer to items in the references at the end of the book.

with $|\delta_i| \le u$, all i. The computed answer is, therefore,

$$\hat{x}_n = x_0^{2^n}(1 + \delta_1)^{2^{n-1}} \cdots (1 + \delta_{n-1})^2(1 + \delta_n)$$

To simplify this expression, we observe that, if $|\delta_1|, \ldots, |\delta_r| \le u$, then

$$(1 + \delta_1) \cdots (1 + \delta_r) = (1 + \delta)^r$$

for some δ with $|\delta| \le u$ (see Exercise 1.3-6). Also then

$$(1 + \delta)^r = (1 + \eta)^{r+1}$$

for some η with $|\eta| \le u$. Consequently,

$$\hat{x}_n = x_0^{2^n}(1 + \delta)^{2^n} = f(x_0(1 + \delta))$$

for some δ with $|\delta| \le u$. In words, the computed value $\hat{x}_n$ for $f(x_0)$ is the exact value of $f(x)$ at the perturbed argument $x = x_0(1 + \delta)$.

We can now gauge the effect which the use of floating-point arithmetic has had on the accuracy of the computed value for $f(x_0)$ by studying how the value of the (exactly computed) function $f(x)$ changes when the argument x is perturbed, as is done in the next section. Further, we note that this error is, in our example, comparable to the error due to the fact that we had to convert the initial datum x_0 to a floating-point number to begin with.

As a second example, of particular interest in Chap. 4, consider calculation of the number s from the equation

$$a_1b_1 + \cdots + a_rb_r + a_{r+1}s = c \tag{1.12}$$

by the formula

$$s = \left(c - \sum_{k=1}^{r} a_kb_k\right)/a_{r+1}$$

If we obtain s through the steps

$$\begin{aligned} s_0 &:= c \\ s_i &:= s_{i-1} - a_ib_i, \qquad i = 1, \ldots, r \\ s &:= s_r/a_{r+1} \end{aligned}$$

then the corresponding numbers computed in floating-point arithmetic satisfy

$$\begin{aligned} \hat{s}_0 &= c \\ \hat{s}_i &= [\hat{s}_{i-1} - a_ib_i(1 + \delta)](1 + \delta), \qquad i = 1, \ldots, r \\ \hat{s} &= \hat{s}_r/[a_{r+1}(1 + \delta)] \end{aligned}$$

Here, we have used Eqs. (1.11a) and (1.11b), and have not bothered to distinguish the various δ's by subscripts. Consequently,

$$\begin{aligned} a_{r+1}(1 + \delta)\hat{s} &= \hat{s}_r \\ &= \hat{s}_{r-1}(1 + \delta) - a_rb_r(1 + \delta)^2 \\ &= \hat{s}_{r-2}(1 + \delta)^2 - a_{r-1}b_{r-1}(1 + \delta)^3 - a_rb_r(1 + \delta)^2 \\ &\vdots \\ &= \hat{s}_0(1 + \delta)^r - a_1b_1(1 + \delta)^{r+1} - \cdots - a_rb_r(1 + \delta)^2 \end{aligned}$$

This shows that the computed value $\hat{s}$ for s satisfies the perturbed equation

$$a_1b_1(1 + \delta)^{r+1} + \cdots + a_rb_r(1 + \delta)^2 + a_{r+1}(1 + \delta)\hat{s} = c(1 + \delta)^r \tag{1.13}$$

Note that we can reduce all exponents by 1 in case $a_{r+1} = 1$, that is, in case the last division need not be carried out.

EXERCISES

1.3-1 The following numbers are given in a decimal computer with a four-digit normalized mantissa:

(a) $0.4523 \cdot 10^4$ (b) $0.2115 \cdot 10^{-3}$ (c) $0.2583 \cdot 10^1$

Perform the following operations, and indicate the error in the result, assuming symmetric rounding:

a. $(a) + (b) + (c)$ *d.* $(a) - (b) - (c)$
b. $(a)/(c)$ *e.* $(a)(b)/(c)$
c. $(a) - (b)$ *f.* $(b)/(c) \cdot (a)$

1.3-2 Let $fl(x)$ be given by chopping. Show that $fl(-x) = -fl(x)$, and that $fl(\beta^r x) = \beta^r fl(x)$ (unless overflow or underflow occurs).

1.3-3 Let $fl(x)$ be given by chopping and let $\delta = \delta(x)$ be such that $fl(x) = x(1 + \delta)$. (If $x = 0$, pick $\delta = 0$.) Show that then δ is bounded as in (1.9).

1.3-4 Give examples to show that most of the laws of arithmetic fail to hold for floating-point arithmetic. (*Hint:* Try laws involving three operands.)

1.3-5 Write a FORTRAN FUNCTION FL(X) which returns the value of the n-decimal-digit floating-point number derived from X by rounding. Take n to be 4 and check your calculations in Exercise 1.3-1. [Use ALOG10(ABS(X)) to determine e such that $10^{e-1} \le |x| < 10^e$.]

1.3-6 Let $U = \{1 + \delta : |\delta| \le u\}$. Show that for all $\alpha_1, \ldots, \alpha_r \in U$, there exists $\alpha \in U$ so that $\alpha_1\alpha_2 \cdots \alpha_r = \alpha^r$. Show also that $a_1\alpha_1 + a_2\alpha_2 + \cdots + a_r\alpha_r = (a_1 + a_2 + \cdots + a_r)\alpha$ for some $\alpha \in U$, provided $a_1, a_2, \ldots, a_r$ all have the same sign.

1.3-7 Carry out a backward error analysis for the calculation of the scalar product $s = a_1b_1 + a_2b_2 + \cdots + a_nb_n$. Redo the analysis under the assumption that double-precision accumulation is used. This means that the double-precision results of each multiplication are retained and added to the sum in double precision, with the resulting sum rounded only at the end to single precision.

1.4 LOSS OF SIGNIFICANCE AND ERROR PROPAGATION; CONDITION AND INSTABILITY

If the number x^* is an approximation to the exact answer x, then we call the difference $x - x^*$ the **error** in x^*; thus

$$\text{Exact} = \text{approximation} + \text{error} \tag{1.14}$$

The **relative error** in x^*, as an approximation to x, is defined to be the number $(x - x^*)/x$. Note that this number is close to the number $(x - x^*)/x^*$ if it is at all small. [Precisely, if $\alpha = (x - x^*)/x$, then $(x - x^*)/x^* = \alpha/(1 - \alpha)$.]

Every floating-point operation in a computational process may give rise to an error which, once generated, may then be amplified or reduced in subsequent operations.

One of the most common (and often avoidable) ways of increasing the importance of an error is commonly called **loss of significant digits**. If x^* is an approximation to x, then we say that x^* approximates x to r significant β-digits provided the absolute error $|x - x^*|$ is at most $\frac{1}{2}$ in the rth significant β-digit of x. This can be expressed in a formula as

$$|x - x^*| \leq \tfrac{1}{2}\beta^{s-r+1} \tag{1.15}$$

with s the largest integer such that $\beta^s \leq |x|$. For instance, $x^* = 3$ agrees with $x = \pi$ to one significant (decimal) digit, while $x^* = \frac{22}{7} = 3.1428\cdots$ is correct to three significant digits (as an approximation to π). Suppose now that we are to calculate the number

$$z = x - y$$

and that we have approximations x^* and y^* for x and y, respectively, available, each of which is good to r digits. Then

$$z^* = x^* - y^*$$

is an approximation for z, which is also good to r digits unless x^* and y^* agree to one or more digits. In this latter case, there will be cancellation of digits during the subtraction, and consequently z^* will be accurate to fewer than r digits.

Consider, for example,

$$x^* = (0.76545421)10^1 \qquad y^* = (0.76544200)10^1$$

and assume each to be an approximation to x and y, respectively, correct to seven significant digits. Then, in eight-digit floating-point arithmetic,

$$z^* = x^* - y^* = (0.12210000)10^{-3}$$

is the *exact* difference between x^* and y^*. But as an approximation to $z = x - y$, z^* is good only to three digits, since the fourth significant digit of z^* is derived from the eighth digits of x^* and y^*, both possibly in error.

Hence, while the *error* in z^* (as an approximation to $z = x - y$) is at most the sum of the errors in x^* and y^*, the *relative* error in z^* is possibly 10,000 times the relative error in x^* or y^*. Loss of significant digits is therefore dangerous only if we wish to keep the relative error small.

Such loss can often be avoided by anticipating its occurrence. Consider, for example, the evaluation of the function

$$f(x) = 1 - \cos x$$

in six-decimal-digit arithmetic. Since $\cos x \approx 1$ for x near zero, there will be loss of significant digits for x near zero if we calculate $f(x)$ by first finding $\cos x$ and then subtracting the calculated value from 1. For we cannot calculate $\cos x$ to more than six digits, so that the error in the calculated value may be as large as $5 \cdot 10^{-7}$, hence as large as, or larger than, $f(x)$ for x near zero. If one wishes to compute the value of $f(x)$ near zero to about six significant digits using six-digit arithmetic, one would have to use an alternative formula for $f(x)$, such as

$$f(x) = 1 - \cos x = \frac{1 - \cos^2 x}{1 + \cos x} = \frac{\sin^2 x}{1 + \cos x}$$

which can be evaluated quite accurately for small x; else, one could make use of the Taylor expansion (see Sec. 1.7) for $f(x)$,

$$f(x) = \frac{x^2}{2} - \frac{x^4}{24} + \cdots$$

which shows, for example, that for $|x| \leq 10^{-3}$, $x^2/2$ agrees with $f(x)$ to at least six significant digits.

Another example is provided by the problem of finding the roots of the quadratic equation

$$ax^2 + bx + c = 0 \tag{1.16}$$

We know from algebra that the roots are given by the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.17}$$

Let us assume that $b^2 - 4ac > 0$, that $b > 0$, and that we wish to find the root of smaller absolute value using (1.17); i.e.,

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{1.18}$$

If $4ac$ is small compared with b^2, then $\sqrt{b^2 - 4ac}$ will agree with b to several places. Hence, given that $\sqrt{b^2 - 4ac}$ will be calculated correctly only to as many places as are used in the calculations, it follows that the numerator of (1.18), and therefore the calculated root, will be accurate to fewer places than were used during the calculation. To be specific, take the

equation

$$x^2 + 111.11x + 1.2121 = 0 \tag{1.19}$$

Using (1.18) and five-decimal-digit floating-point chopped arithmetic, we calculate

$$\begin{aligned} b^2 &= 12{,}345 \\ b^2 - 4ac &= 12{,}340 \\ \sqrt{b^2 - 4ac} &= 111.09 \\ x_1 &= \frac{-b + \sqrt{b^2 - 4ac}}{2a} = -0.01000 \end{aligned}$$

while in fact, $x_1 = -0.010910$

is the correct root to the number of digits shown. Here too, the loss of significant digits can be avoided by using an alternative formula for the calculation of the absolutely smaller root, viz.,

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}} \tag{1.20}$$

Using this formula, and five-decimal-digit arithmetic, we calculate

$$x_1 = -0.010910$$

which is accurate to five digits.

Once an error is committed, it contaminates subsequent results. This **error propagation** through subsequent calculations is conveniently studied in terms of the two related concepts of *condition* and *instability*.

The word **condition** is used to describe the sensitivity of the function value $f(x)$ to changes in the argument x. The condition is usually measured by the maximum relative change in the function value $f(x)$ caused by a unit relative change in the argument. In a somewhat informal formula,

$$\text{condition of } f \text{ at } x = \max\left\{ \left| \frac{f(x) - f(x^*)}{f(x)} \right| \Big/ \left| \frac{x - x^*}{x} \right| : \quad |x - x^*| \text{ "small"} \right\} \approx \left| \frac{f'(x)x}{f(x)} \right| \tag{1.21}$$

The larger the condition, the more ill-conditioned the function is said to be. Here we have made use of the fact (see Sec. 1.7) that

$$f(x) - f(x^*) \approx f'(x)(x - x^*)$$

i.e., the change in argument from x to x^* changes the function value by approximately $f'(x)(x - x^*)$.

If, for example,

$$f(x) = \sqrt{x}$$

then $f'(x) = \frac{1}{2}/\sqrt{x}$, hence the condition of f is, approximately,

$$\left| \frac{f'(x)x}{f(x)} \right| = \frac{\left[\frac{1}{2}/\sqrt{x} \right] x}{\sqrt{x}} = \frac{1}{2}$$

This says that taking square roots is a well-conditioned process since it actually reduces the relative error. By contrast, if

$$f(x) = \frac{10}{1 - x^2}$$

then $f'(x) = 20x/(1 - x^2)^2$, so that

$$\left| \frac{f'(x)x}{f(x)} \right| = \left| \frac{\left[20x/(1 - x^2)^2 \right] x}{10/(1 - x^2)} \right| = \frac{2x^2}{|1 - x^2|}$$

and this number can be quite large for $|x|$ near 1. Thus, for x near 1 or -1, this function is quite ill-conditioned. It very much magnifies relative errors in the argument there.

The related notion of **instability** describes the sensitivity of a numerical process for the calculation of $f(x)$ from x to the inevitable rounding errors committed during its execution in finite precision arithmetic. The precise effect of these errors on the accuracy of the computed value for $f(x)$ is hard to determine except by actually carrying out the computations for particular finite precision arithmetics and comparing the computed answer with the exact answer. But it is possible to estimate these effects roughly by considering the rounding errors one at a time. This means we look at the individual computational steps which make up the process. Suppose there are n such steps. Denote by x_i the output from the ith such step, and take $x_0 = x$. Such an x_i then serves as input to one or more of the later steps and, in this way, influences the final answer $x_n = f(x)$. Denote by f_i the function which describes the dependence of the final answer on the intermediate result x_i. In particular, f_0 is just f. Then the total process is unstable to the extent that one or more of these functions f_i is ill-conditioned. More precisely, the process is unstable to the extent that one or more of the f_i's has a much larger condition than $f = f_0$ has. For it is the condition of f_i which gauges the relative effect of the inevitable rounding error incurred at the ith step on the final answer.

To give a simple example, consider the function

$$f(x) = \sqrt{x + 1} - \sqrt{x}$$

for "large" x, say for $x \approx 10^4$. Its condition there is

$$\left| \frac{f'(x)x}{f(x)} \right| = \frac{1}{2} \frac{|1/\sqrt{x + 1} - 1/\sqrt{x}\,|x}{\sqrt{x + 1} - \sqrt{x}} = \frac{1}{2} \frac{x}{\sqrt{x + 1}\,\sqrt{x}} \approx \frac{1}{2}$$

which is quite good. But, if we calculate $f(12345)$ in six-decimal arithmetic,

we find

$$f(12345) = \sqrt{12346} - \sqrt{12345}$$
$$= 111.113 - 111.108 = 0.005$$

while, actually, $f(12345) = 0.00450003262627751 \cdots$

So our calculated answer is in error by 10 percent. We analyze the computational process. It consists of the following four computational steps:

$$\begin{aligned} x_0 &:= 12345 \\ x_1 &:= x_0 + 1 \\ x_2 &:= \sqrt{x_1} \\ x_3 &:= \sqrt{x_0} \\ x_4 &:= x_2 - x_3 \end{aligned} \tag{1.22}$$

Now consider, for example, the function f_3, i.e., the function which describes how the final answer x_4 depends on x_3. We have

$$f_3(t) = x_2 - t$$

hence its condition is, approximately,

$$\left|\frac{f_3'(t)t}{f_3(t)}\right| = \left|\frac{t}{x_2 - t}\right|$$

This number is usually near 1, i.e., f_3 is usually well-conditioned *except* when t is near x_2. In this latter case, f_3 can be quite badly conditioned. For example, in our particular case, $t \approx 111.11$ while $x_2 - t \approx 0.005$, so the condition is $\sim 22{,}222$, or more than 40,000 times as big as the condition of f itself.

We conclude that the process described in (1.22) is an unstable way to evaluate f. Of course, if you have read the beginning of this section carefully, then you already know a stable way to evaluate this function, namely by the equivalent formula

$$f(x) = \frac{1}{\sqrt{x+1} + \sqrt{x}}$$

In six-decimal arithmetic, this gives

$$f(12345) = \frac{1}{\sqrt{12346} + \sqrt{12345}} = \frac{1}{222.221} = 0.00450002$$

which is in error by only 0.0003 percent. The computational process is

$$\begin{aligned} x_0 &:= 12345 \\ x_1 &:= x_0 + 1 \\ x_2 &:= \sqrt{x_1} \\ x_3 &:= \sqrt{x_0} \\ x_4 &:= x_2 + x_3 \\ x_5 &:= 1/x_4 \end{aligned} \tag{1.23}$$

Here, for example, $f_3(t) = 1/(x_2 + t)$, and the condition of this function is, approximately,

$$\left|\frac{f_3'(t)t}{f_3(t)}\right| = \left|\frac{t}{x_2 + t}\right| \approx \frac{1}{2}$$

for $t \approx x_2$, which is the case here. Thus, the condition of f_3 is quite good; it is as good as that of f itself.

We will meet other examples of large instability, particularly in the discussion of the numerical solution of differential equations.

EXERCISES

1.4-1 Find the root of smallest magnitude of the equation

$$x^2 + 0.4002 \cdot 10^0 x + 0.8 \cdot 10^{-4} = 0$$

using formulas (1.18) and (1.20). Work in floating-point arithmetic using a four- (decimal-) place mantissa.

1.4-2 Estimate the error in evaluating $f(x) = (\cos x)\exp(10x^2)$ around $x = 2$ if the absolute error in x is 10^{-6}.

1.4-3 Find a way to calculate

(*a*) $f(x) = \dfrac{x - \sin x}{\tan x}$

(*b*) $f(x) = (\alpha + x)^n - \alpha^n$,

(*c*) $f(x) = \sin(\alpha + x) - \sin \alpha$

(*d*) $f(x) = x - \sqrt{x^2 - \alpha}$

correctly to the number of digits used when x is near zero for (*a*)–(*c*), very much larger than α for (*d*).

1.4-4 Assuming a computer with a four-decimal-place mantissa, add the following numbers first in ascending order (from smallest to largest) and then in descending order. In doing so round off the partial sums. Compare your results with the correct sum $x = 0.107101023 \cdot 10^5$.

$0.1580 \cdot 10^0$	$0.6266 \cdot 10^2$	$0.8999 \cdot 10^4$
$0.2653 \cdot 10^0$	$0.7555 \cdot 10^2$	
$0.2581 \cdot 10^1$	$0.7889 \cdot 10^3$	
$0.4288 \cdot 10^1$	$0.7767 \cdot 10^3$	

1.4-5 A dramatically unstable way to calculate $f(x) = e^x$ for negative x is provided by its Taylor series (1.36). Calculate e^{-12} by evaluating the Taylor series (1.36) at $x = -12$ and

we find

$$f(12345) = \sqrt{12346} - \sqrt{12345}$$
$$= 111.113 - 111.108 = 0.005$$

while, actually, $f(12345) = 0.00450003262627751 \cdots$

So our calculated answer is in error by 10 percent. We analyze the computational process. It consists of the following four computational steps:

$$\begin{aligned} x_0 &:= 12345 \\ x_1 &:= x_0 + 1 \\ x_2 &:= \sqrt{x_1} \\ x_3 &:= \sqrt{x_0} \\ x_4 &:= x_2 - x_3 \end{aligned} \tag{1.22}$$

Now consider, for example, the function f_3, i.e., the function which describes how the final answer x_4 depends on x_3. We have

$$f_3(t) = x_2 - t$$

hence its condition is, approximately,

$$\left|\frac{f_3'(t)t}{f_3(t)}\right| = \left|\frac{t}{x_2 - t}\right|$$

This number is usually near 1, i.e., f_3 is usually well-conditioned *except* when t is near x_2. In this latter case, f_3 can be quite badly conditioned. For example, in our particular case, $t \approx 111.11$ while $x_2 - t \approx 0.005$, so the condition is $\sim 22{,}222$, or more than 40,000 times as big as the condition of f itself.

We conclude that the process described in (1.22) is an unstable way to evaluate f. Of course, if you have read the beginning of this section carefully, then you already know a stable way to evaluate this function, namely by the equivalent formula

$$f(x) = \frac{1}{\sqrt{x+1} + \sqrt{x}}$$

In six-decimal arithmetic, this gives

$$f(12345) = \frac{1}{\sqrt{12346} + \sqrt{12345}} = \frac{1}{222.221} = 0.00450002$$

which is in error by only 0.0003 percent. The computational process is

$$\begin{aligned} x_0 &:= 12345 \\ x_1 &:= x_0 + 1 \\ x_2 &:= \sqrt{x_1} \\ x_3 &:= \sqrt{x_0} \\ x_4 &:= x_2 + x_3 \\ x_5 &:= 1/x_4 \end{aligned} \tag{1.23}$$

Here, for example, $f_3(t) = 1/(x_2 + t)$, and the condition of this function is, approximately,

$$\left|\frac{f_3'(t)t}{f_3(t)}\right| = \left|\frac{t}{x_2 + t}\right| \approx \frac{1}{2}$$

for $t \approx x_2$, which is the case here. Thus, the condition of f_3 is quite good; it is as good as that of f itself.

We will meet other examples of large instability, particularly in the discussion of the numerical solution of differential equations.

EXERCISES

H **1.4-1** Find the root of smallest magnitude of the equation

$$x^2 + 0.4002 \cdot 10^0 x + 0.8 \cdot 10^{-4} = 0$$

using formulas (1.18) and (1.20). Work in floating-point arithmetic using a four- (decimal-) place mantissa.

1.4-2 Estimate the error in evaluating $f(x) = (\cos x)\exp(10x^2)$ around $x = 2$ if the absolute error in x is 10^{-6}.

H **1.4-3** Find a way to calculate

(*a*) $f(x) = \dfrac{x - \sin x}{\tan x}$
(*b*) $f(x) = (\alpha + x)^n - \alpha^n$,
(*c*) $f(x) = \sin(\alpha + x) - \sin \alpha$
(*d*) $f(x) = x - \sqrt{x^2 - \alpha}$

correctly to the number of digits used when x is near zero for (*a*)–(*c*), very much larger than α for (*d*).

1.4-4 Assuming a computer with a four-decimal-place mantissa, add the following numbers first in ascending order (from smallest to largest) and then in descending order. In doing so round off the partial sums. Compare your results with the correct sum $x = 0.107101023 \cdot 10^5$.

$0.1580 \cdot 10^0$	$0.6266 \cdot 10^2$	$0.8999 \cdot 10^4$
$0.2653 \cdot 10^0$	$0.7555 \cdot 10^2$	
$0.2581 \cdot 10^1$	$0.7889 \cdot 10^3$	
$0.4288 \cdot 10^1$	$0.7767 \cdot 10^3$	

1.4-5 A dramatically unstable way to calculate $f(x) = e^x$ for negative x is provided by its Taylor series (1.36). Calculate e^{-12} by evaluating the Taylor series (1.36) at $x = -12$ and

compare with the accurate value $e^{-12} = 0.00000\ 61442\ 12354\ \cdots$. [*Hint:* By (1.36), the difference between e^x and the partial sum $s_n = \sum_0^n x^j/j!$ is less than the next term $|x^{n+1}/(n+1)!|$ in absolute value, in case x is negative. So, it would be all right to sum the series until $s_n = s_{n+1}$.]

1.4-6 Explain the result of Exercise 1.4-5 by comparing the condition of $f(x) = e^x$ near $x = -12$ with the condition of some of the functions f_i involved in the computational process. Then find a stable way to calculate e^{-12} from the Taylor series (1.36). (*Hint:* $e^{-x} = 1/e^x$.)

1.5 COMPUTATIONAL METHODS FOR ERROR ESTIMATION

This chapter is intended to make the student aware of the possible sources of error and to point out some techniques which can be used to avoid these errors. In appraising computer results, such errors must be taken into account. Realistic estimates of the total error are difficult to make in a practical problem, and an adequate mathematical theory is still lacking. An appealing idea is to make use of the computer itself to provide us with such estimates. Various methods of this type have been proposed. We shall discuss briefly five of them. The simplest method makes use of **double precision**. Here one simply solves the same problem twice—once in single precision and once in double precision. From the difference in the results an estimate of the total round-off error can then be obtained (assuming that all other errors are less significant). It can then be assumed that the same accumulation of roundoff will occur in other problems solved with the same subroutine. This method is extremely costly in machine time since double-precision arithmetic increases computer time by a factor of 8 on some machines, and in addition, it is not always possible to isolate other errors.

A second method is **interval arithmetic**. Here each number is represented by two machine numbers, the maximum and the minimum values that it might have. Whenever an operation is performed, one computes its maximum and minimum values. Essentially, then, one will obtain two solutions at every step, the true solution necessarily being contained within the range determined by the maximum and minimum values. This method requires more than twice the amount of computer time and about twice the storage of a standard run. Moreover, the usual assumption that the true solution lies about midway within the range is not, in general, valid. Thus the range might be so large that any estimate of the round-off error based upon this would be grossly exaggerated.

A third approach is **significant-digit** arithmetic. As pointed out earlier, whenever two nearly equal machine numbers are subtracted, there is a danger that some significant digits will be lost. In significant-digit arithmetic an attempt is made to keep track of digits so lost. In one version

only the significant digits in any number are retained, all others being discarded. At the end of a computation we will thus be assured that all digits retained are significant. The main objection to this method is that some information is lost whenever digits are discarded, and that the results obtained are likely to be much too conservative. Experimentation with this technique is still going on, although the experience to date is not too promising.

A fourth method which gives considerable promise of providing an adequate mathematical theory of round-off-error propagation is based on a **statistical approach**. It begins with the assumption that round-off errors are independent. This assumption is, of course, not valid, because if the same problem is run on the same machine several times, the answers will always be the same. We can, however, adopt a stochastic model of the propagation of round-off errors in which the local errors are treated as if they were random variables. Thus we can assume that the local round-off errors are either uniformly or normally distributed between their extreme values. Using statistical methods, we can then obtain the standard deviation, the variance of distribution, and estimates of the accumulated round-off error. The statistical approach is considered in some detail by Hamming [1] and Henrici [2]. The method does involve substantial analysis and additional computer time, but in the experiments conducted to date it has obtained error estimates which are in remarkable agreement with experimentally available evidence.

A fifth method is **backward error analysis**, as introduced in Sec. 1.3. As we saw, it reduces the analysis of rounding error effects to a study of perturbations in exact arithmetic and, ultimately, to a question of condition. We will make good use of this method in Chap. 4.

1.6 SOME COMMENTS ON CONVERGENCE OF SEQUENCES

Calculus, and more generally analysis, is based on the notion of convergence. Basic concepts such as derivative, integral, and continuity are defined in terms of convergent sequences, and elementary functions such as $\ln x$ or $\sin x$ are defined by convergent series. At the same time, numerical answers to engineering and scientific problems are never needed *exactly*. Rather, an approximation to the answer is required which is accurate "to a certain number of decimal places," or accurate to within a given tolerance ε.

It is therefore not surprising that many numerical methods for finding the answer α of a given problem merely produce (the first few terms of) a sequence $\alpha_1, \alpha_2, \alpha_3, \ldots$ which is shown to converge to the desired answer.

To recall the definition:

A sequence $\alpha_1, \alpha_2, \ldots$ of (real or complex) numbers converges to α if and only if, for all $\varepsilon > 0$, there exists an integer $n_0(\varepsilon)$ such that for all $n \geq n_0$, $|\alpha - \alpha_n| < \varepsilon$.

Hence, if we have a numerical method which produces a sequence $\alpha_1, \alpha_2, \ldots$ converging to the desired answer α, then we can calculate α to any desired accuracy merely by calculating α_n for "large enough" n.

From a computational point of view, this definition is unsatisfactory for the following reasons: (1) It is often not possible (without knowing the answer α) to know when n is "large enough." In other words, it is difficult to get hold of the function $n_0(\varepsilon)$ mentioned in the definition of convergence. (2) Even when some knowledge about $n_0(\varepsilon)$ is available, it may turn out that the required n is too large to make the calculation of α_n feasible.

Example The number $\pi/4$ is the value of the infinite series

$$\sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = 1 - \sum_{j=1}^{\infty} \frac{2}{16j^2 - 1}$$

Hence, with

$$\alpha_n = 1 - \sum_{j=1}^{n} \frac{2}{16j^2 - 1} \qquad n = 1, 2, \ldots$$

the sequence $\alpha_1, \alpha_2, \ldots$ is monotone-decreasing to its limit $\pi/4$. Moreover,

$$0 \leq \alpha_n - \pi/4 \leq \frac{1}{4n+3} \qquad n = 1, 2, \ldots$$

To calculate $\pi/4$ correct to within 10^{-6} using this sequence, we would need $10^6 \leq 4n + 3$, or roughly, $n = 250{,}000$. On a computer using eight-decimal-digit floating-point arithmetic, round-off in the calculation of $\alpha_{250{,}000}$ is probably much larger than 10^{-6}. Hence $\pi/4$ could not be computed to within 10^{-6} using this sequence (except, perhaps, by adding the terms from smallest to largest).

To deal with these problems, some notation is useful. Specifically, we would like to measure how fast sequences converge. As with all measuring, this is done by comparison, with certain standard sequences, such as

$$\left.\begin{array}{l} 1/n \\ 1/n^r \\ r^n \\ 1/(\ln n) \end{array}\right\} \qquad n = 1, 2, 3, \ldots$$

The comparison is made as follows: one says that α_n **is of order** β_n (or α_n is big-oh of β_n), and writes

$$\alpha_n = \mathcal{O}(\beta_n) \tag{1.24}$$

in case

$$|\alpha_n| \leq K|\beta_n| \tag{1.25}$$

for some constant K and all sufficiently large n. Thus

$$\left.\begin{array}{l} 1/n \\ 10{,}000/n \\ 10/n - 40/n^2 + e^{-n} \\ 1/n^2 \end{array}\right\} = \mathcal{O}(1/n)$$

Further, if it is possible to choose the constant K in (1.25) arbitrarily small as soon as n is large enough; that is, should it happen that

$$\lim_{n\to\infty} \alpha_n/\beta_n = 0$$

then one says that α_n **is of higher order than** β_n (or α_n is little-oh of β_n), and writes

$$\alpha_n = o(\beta_n) \tag{1.26}$$

Thus

$$\left.\begin{array}{l} 1/n^2 \\ 1/(n \ln n) \end{array}\right\} = o(1/n)$$

while $\sin(1/n) \neq o(1/n)$.

The order notation appears customarily only on the right-hand side of an equation and serves the purpose of describing the essential feature of an error term without bothering about multiplying constants or other detail. For instance, we can state concisely the unsatisfactory state of affairs in the earlier example by saying that

$$1 - \sum_{j=2}^{n} 1/(j^2 - 1) = \pi/4 + \mathcal{O}(1/n)$$

but also

$$1 - \sum_{j=2}^{n} 1/(j^2 - 1) \neq \pi/4 + o(1/n)$$

i.e., the series converges to $\pi/4$ as fast as $1/n$ (goes to zero) but no faster. A convergence order or rate of $1/n$ is much too slow to be useful in calculations.

Example If $\alpha_n = \alpha + o(1)$, then, by definition,

$$\lim_{n\to\infty} \frac{\alpha_n - \alpha}{1} = 0$$

Hence $\alpha_n = \alpha + o(1)$ is just a fancy way of saying that the sequence $\alpha_1, \alpha_2, \ldots$ converges to α.

Example If $|r| < 1$, then the geometric series $\sum_{i=0}^{\infty} r^i$ sums to $1/(1-r)$. With $s_n = \sum_{i=0}^{n} r^i$, we have $s_n = (1 - r^{n+1})/(1-r) = 1/(1-r) - r^{n+1}/(1-r)$. Thus

$$s_n = \frac{1}{1-r} + \mathcal{O}(r^n)$$

Further, if $\bar{r} > |r|$, then

$$s_n = \frac{1}{1-r} + o(\bar{r}^n)$$

Hence, whenever $\alpha_n = \alpha + \mathcal{O}(r^n)$ for some $|r| < 1$, we say that the convergence is (at least) geometric, for it is then (at least) of the same order as the convergence of the geometric series.

Although it is better to know that $\alpha_n = \alpha + \mathcal{O}(\beta_n)$ than to know nothing, knowledge about the order of convergence becomes quite useful only when we know more precisely that

$$\alpha_n = \alpha + \beta_n + o(\beta_n)$$

This says that for "large enough" n, $\alpha_n \approx \alpha + \beta_n$. To put it differently,

$$\begin{aligned}\alpha_n &= \alpha + \beta_n + o(\beta_n)\\ &= \alpha + \beta_n + \beta_n o(1)\\ &= \alpha + \beta_n(1 + \varepsilon_n)\end{aligned}$$

where $\varepsilon_1, \varepsilon_2, \ldots$ is a sequence converging to zero. Although we cannot *prove* that a certain n is "large enough," we can *test the hypothesis* that n is "large enough" by comparing $\alpha_{k+1} - \alpha_k$ with $\beta_{k+1} - \beta_k$. If

$$\frac{|\alpha_{k+1} - \alpha_k|}{|\beta_{k+1} - \beta_k|} \approx 1$$

for k near n, say for $k = n - 2, n - 1, n$, then we accept the hypothesis that n is "large enough" for

$$\alpha_n \approx \alpha + \beta_n$$

to be true, and therefore accept $|\beta_n|$ as a good estimate of the error $|\alpha - \alpha_n|$.

Example Let $p > 1$. Then the series $\Sigma_j 1/(p^j + 1)$ converges to its limit α like the geometric series $\Sigma_j 1/p^j$, i.e.,

$$\alpha_n = \sum_{j=1}^{n} 1/(p^j + 1) = \alpha + \mathcal{O}(1/p^{n+1})$$

To get a more precise statement, consider

$$\beta_n = \sum_{j=n+1}^{\infty} 1/p^j = (1/p^{n+1})/(1 - 1/p) = 1/[p^n(p - 1)]$$

Then

$$\begin{aligned}\alpha_n &= \alpha - \sum_{j=n+1}^{\infty} 1/(p^j + 1) = \alpha - \beta_n + \sum_{j=n+1}^{\infty} [1/p^j - 1/(p^j + 1)]\\ &= \alpha - \beta_n + o(\beta_n)\end{aligned}$$

since

$$\begin{aligned}0 \le \sum_{j=n+1}^{\infty} [1/p^j - 1/(p^j + 1)] &= \sum_{j=n+1}^{\infty} 1/(p^j(p^j + 1)) \le \sum_{j=n+1}^{\infty} (1/p^2)^j\\ &= \mathcal{O}((1/p^2)^{n+1})\end{aligned}$$

For the ratios, we find

$$\left|\frac{\alpha_{n+1} - \alpha_n}{\beta_{n+1} - \beta_n}\right| = p^{n+1}/(p^{n+1} + 1)$$

which is, e.g., within $1/10$ of 1 for $n = 3$ and $p = 2$. Thus, $\beta_3 = 1/8 = 0.125$ is then a good indication of the error in $\alpha_3 = 0.64444\ 44444$. In fact, $\alpha = 0.76449\ 9780$; the error in α_3 is therefore $0.12005 \cdots$.

This notation carries over to functions of a real variable. If

$$\lim_{h \to 0} T(h) = A$$

we say that the convergence is $\mathcal{O}(f(h))$ provided

$$\frac{|T(h) - A|}{|f(h)|} \le K$$

for some finite constant K and all small enough h. If this holds for all $K > 0$, that is, if

$$\lim_{h \to 0} \frac{T(h) - A}{f(h)} = 0$$

then we call the convergence $o(f(h))$.

Example For h "near" zero, we have

$$\begin{aligned}\frac{\sin h}{h} &= 1 - (\tfrac{1}{3!})h^2 + (\tfrac{1}{5!})h^4 - \cdots = 1 + \mathcal{O}(h^2)\\ &= 1 - \tfrac{1}{6}h^2 + o(h^2)\end{aligned}$$

Hence, for all $\gamma < 2$, $\quad \dfrac{\sin h}{h} = 1 + o(h^\gamma)$

Example If the function $f(x)$ has a zero of order γ at $x = \xi$, then

$$f(\xi + h) = \mathcal{O}(h^\gamma) \quad \text{but} \quad f(\xi + h) \ne o(h^\gamma)$$

Rules for calculating with the order symbols are collected in the following lemma.

Lemma 1.1 If $\alpha_n = \alpha + \mathcal{O}(f(n))$, $\lim_{n \to \infty} f(n) = 0$, and c is a constant, then

$$c\alpha_n = c\alpha + \mathcal{O}(f(n))$$

If also $\beta_n = \beta + \mathcal{O}(g(n))$, and $g(n) = \mathcal{O}(f(n))$, then

$$\alpha_n + \beta_n = \alpha + \beta + \mathcal{O}(f(n)) \quad \text{and} \quad \alpha_n\beta_n = \alpha\beta + \mathcal{O}(f(n)) \tag{1.27}$$

If, further, $\beta \ne 0$, then also

$$\frac{\alpha_n}{\beta_n} = \frac{\alpha}{\beta} + \mathcal{O}(f(n))$$

while if $\alpha = \beta = 0$, then

$$\alpha_n \beta_n = \mathcal{O}(f(n)g(n))$$

Finally, all statements remain true if $\mathcal{O}$ is replaced by o throughout.

The approximate calculation of a number α via a sequence $\alpha_1, \alpha_2, \ldots$ converging to α always involves an act of faith regardless of whether or not the order of convergence is known. Given that the sequence is known to converge to α, practicing numerical analysts ascertain that n is "large enough" by making sure that, for small values of i, α_{n-i} differs "little enough" from α_n. If they also know that the convergence is $\beta_n + o(\beta_n)$, they check whether or not the sequence behaves accordingly near n. If they also know that α satisfies certain equations or inequalities—α might be the sought-for solution of an equation—they check that α_n satisfies these equations or inequalities "well enough." In short, practicing numerical analysts make sure that n satisfies all conditions they can think of which are necessary for n to be "large enough." If all these conditions are satisfied, then, lacking sufficient conditions for n to be "large enough," they accept α_n on faith as a good enough approximation to α. In a way, numerical analysts use all means at their disposal to distinguish a "good enough" approximation from a bad one. They can do no more (and should do no less).

It follows that numerical results arrived at in this way should not be mistaken for final answers. Rather, they should be questioned freely if subsequent investigations throw any doubt upon their correctness.

The student should appreciate this as another example of the basic difference between numerical analysis and analysis. Analysis became a precise discipline when it left the restrictions of practical calculations to deal entirely with problems posed in terms of an abstract model of the number system, called the *real numbers*. This abstract model is designed to make a precise and useful definition of limit possible, which opens the way to the abstract or symbolic solution of an impressive array of practical problems, once these problems are translated into the terms of the model. This still leaves the task of translating the abstract or symbolic solutions back into practical solutions. Numerical analysis assumes this task, and with it the limitations of practical calculations from which analysis managed to escape so elegantly. Numerical answers are therefore usually tentative and, *at best*, known to be accurate only to within certain bounds.

Numerical analysis is therefore not merely concerned with the *construction* of numerical methods. Rather, a large portion of numerical analysis consists in the derivation of useful *error bounds*, or *error estimates*, for the numerical answers produced by a numerical algorithm. Throughout this book, the student will meet this preoccupation with error bounds so typical of numerical analysis.

EXERCISES

1.6-1 The number ln 2 may be calculated from the series

$$\ln 2 = 1 - \tfrac{1}{2} + \tfrac{1}{3} - \tfrac{1}{4} + \cdots$$

It is known from analysis that this series converges and that the magnitude of the error in any partial sum is less than the magnitude of the first neglected term. Estimate the number of terms that would be required to calculate ln 2 to 10 decimal places.

1.6-2 For h near zero it is possible to write

$$\frac{\tan h}{h} = 1 + \mathcal{O}(h^\gamma)$$

and

$$\frac{\tan h}{h} = 1 + o(h^\delta)$$

Find the values of γ and δ for which these equalities hold.

1.6-3 Try to calculate, on a computer, the limit of the sequence

$$\alpha_n = (\tan 8^{-n} - \sin 8^{-n}) \cdot 8^{3n} \qquad n = 0, 1, 2, \ldots$$

Theoretically, what is $\alpha = \lim_{n \to \infty} \alpha_n$ and what is the order of convergence of the sequence?

1.7 SOME MATHEMATICAL PRELIMINARIES

It is assumed that the student is familiar with the topics normally covered in the undergraduate analytic geometry and calculus sequence. These include elementary notions of real and complex number systems; continuity; the concept of limits, sequences, and series; differentiation and integration. For Chap. 4, some knowledge of determinants is assumed. For Chaps. 8 and 9, some familiarity with the solution of ordinary differential equations is also assumed, although these chapters may be omitted.

In particular, we shall make frequent use of the following theorems.

Theorem 1.1: Intermediate-value theorem for continuous functions Let $f(x)$ be a continuous function on the interval $[a, b]$. If $f(\underline{x}) \le \alpha \le f(\bar{x})$ for some number α and some $\underline{x}, \bar{x} \in [a, b]$, then

$$\alpha = f(\xi) \qquad \text{for some } \xi \in [a, b]$$

This theorem is often used in the following form:

Theorem 1.2 Let $f(x)$ be a continuous function on $[a, b]$, let $x_1, \ldots, x_n$ be points in $[a, b]$, and let $g_1, \ldots, g_n$ be real numbers all *of one sign*. Then

$$\sum_{i=1}^{n} f(x_i) g_i = f(\xi) \sum_{i=1}^{n} g_i \qquad \text{for some } \xi \in [a, b]$$

To indicate the proof, assume without loss of generality that $g_i \geq 0$, $i = 1, \ldots, n$. If $f(\underline{x}) = \min_i f(x_i)$ and $f(\bar{x}) = \max_i f(x_i)$, then

$$f(\underline{x}) \sum_{i=1}^{n} g_i = \sum_{i=1}^{n} f(\underline{x}) g_i \leq \sum_{i=1}^{n} f(x_i) g_i \leq \sum_{i=1}^{n} f(\bar{x}) g_i = f(\bar{x}) \sum_{i=1}^{n} g_i$$

Hence $\alpha = \sum_i f(x_i) g_i$ is a number between the two values $f(\underline{x}) \sum_i g_i$ and $f(\bar{x}) \sum_i g_i$ of the continuous function $f(x) \sum_i g_i$, and the conclusion follows from Theorem 1.1.

One proves analogously the corresponding statement for *infinite* sums or *integrals*:

Theorem 1.3: Mean-value theorem for integrals Let $g(x)$ be a nonnegative or nonpositive integrable function on $[a, b]$. If $f(x)$ is continuous on $[a, b]$, then

$$\int_a^b f(x) g(x)\, dx = f(\xi) \int_a^b g(x)\, dx \qquad \text{for some } \xi \in [a, b] \quad (1.28)$$

Warning The assumption that $g(x)$ is of one sign is essential in Theorem 1.3, as the simple example $f(x) = g(x) = x$, $[a, b] = [-1, 1]$ shows.

Theorem 1.4 Let $f(x)$ be a continuous function on the closed and bounded interval $[a, b]$. Then $f(x)$ "assumes its maximum and minimum values on $[a, b]$"; i.e., there exist points $\underline{x}$ and $\bar{x}$ in $[a, b]$ such that

$$\text{for all } x \in [a, b]: \qquad f(\underline{x}) \leq f(x) \leq f(\bar{x})$$

Theorem 1.5: Rolle's theorem Let $f(x)$ be continuous on the (closed and finite) interval $[a, b]$ and differentiable on (a, b). If $f(a) = f(b) = 0$, then

$$f'(\xi) = 0 \qquad \text{for some } \xi \in (a, b)$$

The proof makes essential use of Theorem 1.4. For by Theorem 1.4, there are points $\underline{x}$, $\bar{x}$ in $[a, b]$ such that, for all $x \in [a, b]$, $f(\underline{x}) \leq f(x) \leq f(\bar{x})$. If now neither $\underline{x}$ nor $\bar{x}$ is in (a, b), then $f(x) \equiv 0$, and every $\xi \in (a, b)$ will do. Otherwise, either $\underline{x}$ or $\bar{x}$ is in (a, b), say, $\bar{x} \in (a, b)$. But then $f'(\bar{x}) = 0$, since

$$0 \leq \lim_{h \to 0-} \frac{f(\bar{x}) - f(\bar{x} + h)}{-h} = f'(\bar{x}) = \lim_{h \to 0+} \frac{f(\bar{x} + h) - f(\bar{x})}{h} \leq 0$$

$f(\bar{x})$ being the *biggest value* achieved by $f(x)$ on $[a, b]$.

An immediate consequence of Rolle's theorem is the following theorem.

Theorem 1.6: Mean-value theorem for derivatives If $f(x)$ is continuous on the (closed and finite) interval $[a, b]$ and differentiable on (a, b),

then

$$\frac{f(b) - f(a)}{b - a} = f'(\xi) \qquad \text{for some } \xi \in (a, b) \quad (1.29)$$

One gets Theorem 1.6 from Theorem 1.5 by considering in Theorem 1.5 the function

$$F(x) = f(x) - f(a) - \frac{f(b) - f(a)}{b - a}(x - a)$$

instead of $f(x)$. Clearly, $F(x)$ vanishes both at a and at b.

It follows directly from Theorem 1.6 that if $f(x)$ is continuous on $[a, b]$ and differentiable on (a, b), and c is some point in $[a, b]$, then for all $x \in [a, b]$

$$f(x) = f(c) + (x - c) f'(c + \theta(x - c)) \qquad \text{for some } \theta \in (0, 1) \quad (1.30)$$

The fundamental theorem of calculus provides the more precise statement: If $f(x)$ is continuously differentiable, then for all $x \in [a, b]$

$$f(x) = f(c) + \int_c^x f'(s)\, ds \quad (1.31)$$

from which (1.30) follows by the mean-value theorem for integrals (1.28), since $f'(x)$ is continuous. More generally, one has the following theorem.

Theorem 1.7: Taylor's formula with (integral) remainder If $f(x)$ has $n + 1$ continuous derivatives on $[a, b]$ and c is some point in $[a, b]$, then for all $x \in [a, b]$

$$f(x) = f(c) + f'(c)(x - c) + \frac{f''(c)(x - c)^2}{2!} + \cdots + \frac{f^{(n)}(c)(x - c)^n}{n!} + R_{n+1}(x) \quad (1.32)$$

where

$$R_{n+1}(x) = \frac{1}{n!} \int_c^x (x - s)^n f^{(n+1)}(s)\, ds \quad (1.33)$$

One gets (1.32) from (1.31) by considering the function

$$F(x) = f(x) + f'(x)(c - x) + \frac{f''(x)(c - x)^2}{2!} + \cdots + \frac{f^{(n)}(x)(c - x)^n}{n!}$$

instead of $f(x)$. For, $F'(x) = f^{(n+1)}(x)(c - x)^n / n!$; hence by (1.31),

$$F(x) = F(c) + \frac{1}{n!} \int_c^x (c - s)^n f^{(n+1)}(s)\, ds$$

But since $F(c) = f(c)$, this gives

$$f(c) = F(x) + \frac{1}{n!} \int_x^c (c - s)^n f^{(n+1)}(s)\, ds$$

which is (1.32), after the substitution of x for c and of c for x.

Actually, $f^{(n+1)}(x)$ need not be continuous for (1.32) to hold. However, if in (1.32), $f^{(n+1)}(x)$ is continuous, one gets, using Theorem 1.3, the more familiar but less useful form for the remainder:

$$R_{n+1}(x) = \frac{f^{(n+1)}(\xi)(x-c)^{n+1}}{(n+1)!} \qquad \text{where } \xi = c + \theta(x-c) \quad (1.34)$$

By setting $h = x - c$, (1.32) and (1.34) take the form

$$f(c+h) = f(c) + hf'(c) + \frac{h^2}{2!}f''(c) + \cdots + \frac{h^n}{n!}f^{(n)}(c) + \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(c+\theta h) \qquad \text{for some } \theta \in (0,1) \quad (1.35)$$

Example The function $f(x) = e^x$ has the Taylor expansion

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \frac{x^{n+1}e^{\xi}}{(n+1)!}$$

for some ξ between 0 and x (1.36)

about $c = 0$. The expansion of $f(x) = \ln x = \log_e x$ about $c = 1$ is

$$\ln x = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots - \frac{(-1)^n(x-1)^n}{n} + \frac{(-1)^n(x-1)^{n+1}\xi^{-(n+1)}}{n+1}$$

where $0 < x \leq 2$, and ξ is between 1 and x.

A similar formula holds for functions of several variables. One obtains this formula from Theorem 1.7 with the aid of

Theorem 1.8: Chain rule If the function $f(x, y, \ldots, z)$ has continuous first partial derivatives with respect to each of its variables, and $x = x(t), y = y(t), \ldots, z = z(t)$ are continuously differentiable functions of t, then $g(t) = f(x(t), y(t), \ldots, z(t))$ is also continuously differentiable, and

$$g'(t) = \frac{\partial f}{\partial x}x'(t) + \frac{\partial f}{\partial y}y'(t) + \cdots + \frac{\partial f}{\partial z}z'(t)$$

From this theorem, one obtains an expression for $f(x, y, \ldots, z)$ in terms of the value and the partial derivatives at $(a, b, \ldots, c)$ by introducing the function

$$g(t) = f(a + t(x-a), b + t(y-b), \ldots, c + t(z-c))$$

and then evaluating its Taylor series expansion around $t = 0$ at $t = 1$. For example, this gives

Theorem 1.9 If $f(x, y)$ has continuous first and second partial derivatives in a neighborhood D of the point (a, b) in the (x, y) plane, then

$$f(x,y) = f(a,b) + f_x(a,b)(x-a) + f_y(a,b)(y-b) + R_2(x,y) \quad (1.37)$$

for all (x, y) in D, where

$$R_2(x,y) = \frac{f_{xx}(\xi,\eta)(x-a)^2}{2} + f_{xy}(\xi,\eta)(x-a)(y-b) + \frac{f_{yy}(\xi,\eta)(y-b)^2}{2}$$

for some $(\xi, \eta) \in D$ depending on (x, y), and the subscripts on f denote partial differentiation.

For example, the expansion of $e^{x \sin y}$ about $(a, b) = (0, 0)$ is

$$e^{x \sin y} = 1 + 0 \cdot x + 0 \cdot y + R_2(x,y) \quad (1.38)$$

Finally, in the discussion of eigenvalues of matrices and elsewhere, we need the following theorem.

Theorem 1.10: Fundamental theorem of algebra If $p(x)$ is a polynomial of degree $n \geq 1$, that is,

$$p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

with $a_0, \ldots, a_n$ real or complex numbers and $a_n \neq 0$, then $p(x)$ has at least one zero; i.e., there exists a complex number ξ such that $p(\xi) = 0$.

This rather deep theorem should not be confused with the straightforward statement, "A polynomial of degree n has at most n zeros, counting multiplicity," which we prove in Chap. 2 and use, for example, in the discussion of polynomial interpolation.

EXERCISES

1.7-1 In the mean-value theorem for integrals, Theorem 1.3, let $f(x) = e^x$, $g(x) = x$, $[a, b] = [0, 1]$. Find the point ξ specified by the theorem and verify that this point lies in the interval $(0, 1)$.

1.7-2 In the mean-value theorem for derivatives, Theorem 1.6, let $f(x) = x^2$. Find the point ξ specified by the theorem and verify that this point lies in the interval (a, b).

1.7-3 In the expansion (1.36) for e^x, find n so that the resulting power sum will yield an approximation correct to five significant digits for all x on $[0, 1]$.

1.7-4 Use Taylor's formula (1.32) to find a power series expansion about $c = 0$ for $\sin(\pi x/2)$. Find an expression for the remainder, and from this estimate the number of terms that would be needed to guarantee six-significant-digit accuracy for $\sin(\pi x/2)$ for all x on the interval $[-1, 1]$.

1.7-5 Find the remainder $R_2(x, y)$ in the example (1.38) and determine its maximum value in the region D defined by $[0 \le x \le \pi/2, 0 \le y \le \pi/2]$.

1.7-6 Prove that the remainder term in (1.35) can also be written

$$\frac{h^{n+1}}{(n+1)!} f^{(n+1)}(c) + o(h^{n+1})$$

[if $f^{(n+1)}(x)$ is continuous at $x = c$].

1.7-7 Illustrate the statement in Exercise 1.7-6 by calculating, for $f(x) = e^x$, $c = 0$,

$$R_3(h) = e^h - \left(1 + h + \frac{h^2}{2}\right) \quad \text{and} \quad \frac{h^3}{3!} f^{(3)}(0) = \frac{h^3}{3!}$$

for various values of h, for example, for $h = 2^{-k}$, $k = 1, 2, 3, \ldots$, and comparing $R_3(h)$ with $(h^3/3!)f^{(3)}(0)$.

1.7-8 Prove Theorem 1.9 from Theorems 1.7 and 1.8.

1.7-9 Prove Euler's formula

$$e^{i\theta} = \cos\theta + i \sin\theta$$

(with $i = \sqrt{-1}$), by comparing the power series for e^x, evaluated at $x = i\theta$, with the sum of the power series for $\cos\theta$ and i times the one for $\sin\theta$.

CHAPTER
TWO

INTERPOLATION BY POLYNOMIALS

Polynomials are used as the basic means of approximation in nearly all areas of numerical analysis. They are used in the solution of equations and in the approximation of functions, of integrals and derivatives, of solutions of integral and differential equations, etc. Polynomials owe this popularity to their simple structure, which makes it easy to construct effective approximations and then make use of them.

For this reason, the representation and evaluation of polynomials is a basic topic in numerical analysis. We discuss this topic in the present chapter in the context of polynomial interpolation, the simplest and certainly the most widely used technique for obtaining polynomial approximations. More advanced methods for getting good approximations by polynomials and other approximating functions are given in Chap. 6. But it will be shown there that even best polynomial approximation does not give appreciably better results than an appropriate scheme of polynomial interpolation.

Divided differences serve as the basis of our treatment of the interpolating polynomial. This makes it possible to deal with osculatory (or Hermite) interpolation as a special limiting case of polynomial interpolation at distinct points.

2.1 POLYNOMIAL FORMS

In this section, we point out that the customary way to describe a polynomial may not always be the best way in calculations, and we

An Intro to Numerical Computation
by S. Yakowitz & F. Szidarovszky

CHAPTER 1

Computer Number Representation and Roundoff

1.1 FUNDAMENTAL CAPABILITIES OF A COMPUTER

To appreciate the concerns of "numerical computations," one must understand what a computer can and cannot do with numbers. Within limits to be discussed in this chapter, the computer has two fundamental capabilities:

> **Capability 1.** The computer can store a finite set of numbers.
> **Capability 2.** The computer can perform arithmetic (addition, subtraction, multiplication, and division), and it can find the order of any two stored numbers x and y. That is, it can decide whether x is greater than, equal to, or less than y.

Every computational solution to a numerical problem must ultimately be built up of operations in which stored numbers are compared or operated on arithmetically and the resultant stored. Most problems of interest cannot be solved by a finite sequence of such operations. We must be content with some program that yields an approximation to the solution and accept that there will typically be approximation error. Such error resulting from replacing a desired mathematical operation by a realizable computation will be referred to as *truncation error*. Techniques for bounding this error or assuring a specified accuracy should accompany a computational method.

For reasons to be disclosed in this chapter, neither capability 1 nor capability 2 can be achieved exactly. Computer words are equivalent to strings of 0's and 1's of uniform length. There are consequently only a fixed finite number of distinct computer words available for approximation of numbers. Thus regardless of what scheme is used to map the infinite set of real numbers into computer numbers, error will usually occur. Such error is referred to as *roundoff error*. In several sections of this chapter, origins, bounds, illustrations, and remedies for roundoff error will be presented. Later chapters offering numerical methods for various

ies of computational problems will give us insight into the cause and mag-
ile of truncation error.
xamples 1.1 and 1.2 demonstrate roundoff and truncation error.

AMPLE 1.1

possible, in fact customary, for a computer to store a number different from
presented as input, and to commit this error without warning. We perform
simple experiment of reading the number $x = 0.1234567890123$ into a DEC
omputer, and then printing out the number actually stored. The program for
g this is shown in Table 1.1. The first of the numbers listed in Table 1.2 is
number stored in the input file, and the second is the representation received
ie output file. Note that the format field length is adequate and is not the cause
ne discrepancy. The difference between these numbers, which is about 2×10^{-9} is roundoff error.

BLE 1.1 Program to Illustrate Roundoff Error

```
      PROGRAM STORE

C     ***************************************************
C     THIS PROGRAM WILL READ A NUMBER FROM ONE FILE AND
C     WRITE IT TO ANOTHER
C     INPUT: X
C     OUTPUT: X
C     ***************************************************

      READ(11,1) X
      WRITE(10,1) X
    1 FORMAT(2X,F15.13)
      STOP
      END
```

BLE 1.2 Input and put

```
it:     0.1234567890123
put:    0.1234567910433
```

■

AMPLE 1.2

all from calculus that the derivative $f'(x)$ of a function $f(x)$ at x_0 is defined to
he limit of

$$D(h) = \frac{f(x_0 + h) - f(x_0)}{h}$$

i converges to 0. This formula for $D(h)$, which we have illustrated in Figure
, is representative of finite difference methods used in Chapter 3 to approximate
ivatives numerically. The true derivative is the slope of the tangent line, in
ure 1.1, and the approximation is the slope of the line connecting $f(x_0)$ and
$_0 + h)$.
f the magnitude of h is "too large," then $D(h)$ is inaccurate because h is not
ficiently close to the limit. This error is the truncation error associated with

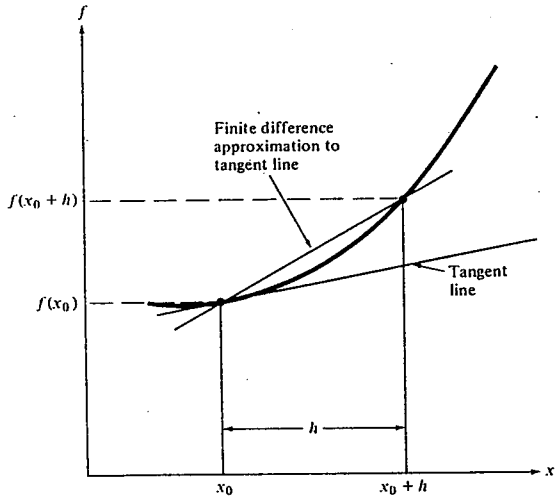


FIGURE 1.1 Finite Difference Approximation

using the "realizable" arithmetic formula $D(h)$ for approximating an unrealizable limiting operation, the derivative. Such error would occur even if $f(x)$ and $D(h)$ could be evaluated exactly. For reasons discussed in Section 1.4.2, as h becomes small, inaccuracies due to roundoff error dominate. The truncation and roundoff error in computing $f(x_0 + h) - f(x_0)$ is large relative to the actual value of this difference. Roundoff error would not be present if $D(h)$ could somehow be computed perfectly.

The program and printout displayed in Tables 1.3 and 1.4 give clear evidence

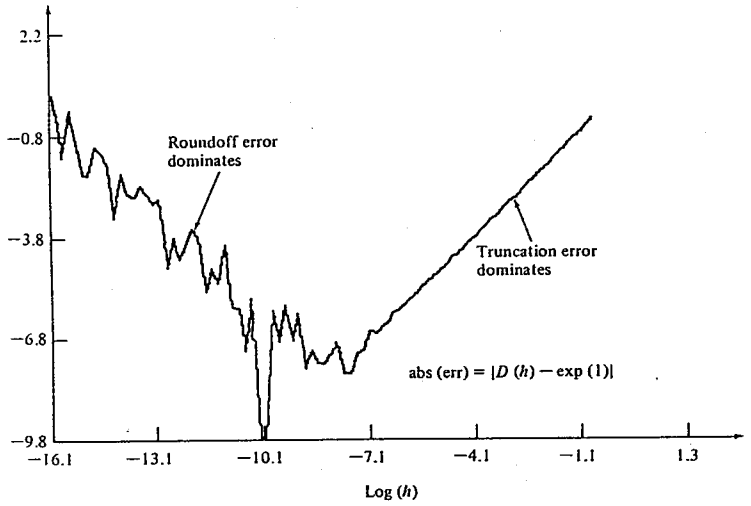
TABLE 1.3 Program for Derivative Approximation

```
C     PROGRAM NUMAPP
C
C     ***************************************************
C     THIS PROGRAM DEMONSTRATES ERRORS IN NUMERICAL
C     APPROXIMATION OF THE DERIVATIVE OF THE
C     EXPONENTIAL FUNCTION
C     OUTPUT: H-THE MAGNITUDE OF THE CHANGE IN INDEPENDENT
C               VARIABLE USED IN THE FINITE DIFFERENCE
C               APPROXIMATION
C             ERR-THE DIFFERENCE BETWEEN THE TRUE VALUE
C                 OF THE DERIVATIVE AS COMPUTED BY THE
C                 LIBRARY FUNCTION EXP AND THE APPROXIMATION
C     ***************************************************
C
      X=1.
      TRUE=EXP(1.)
      H=0.5
      B=EXP(X)
      DO WHILE(H.GT.1.E-8)
         APROX=(EXP(X+H)-B)/H
         ERR=TRUE-APROX
         WRITE(10,*)H,ERR
         H=H/2.
      END DO
      STOP
      END
```

BLE 1.4 Printout Showing Errors in merical Approximation of a Derivative

h	exp(1) - D(h)	
000000	-0.8085327	
500000	-0.3699627	
250000	-0.1771994	
500000E-02	-8.6745262E-02	
250000E-02	-4.2918205E-02	Truncation
625000E-02	-2.1357536E-02	Error
125000E-03	-1.0661125E-02	Dominates
062500E-03	-5.3510666E-03	
531250E-03	-2.6655197E-03	
656250E-04	-1.4448166E-03	
828125E-04	-9.5653534E-04	
414063E-04	-4.6825409E-04	
207031E-04	-4.6825409E-04	
035156E-05	-4.6825409E-04	
517578E-05	-4.6825409E-04	
258789E-05	-4.6825409E-04	
293945E-06	-4.6825409E-04	
146973E-06	-3.1718254E-02	Roundoff
073486E-06	-3.1718254E-02	Error
367432E-07	-3.1718254E-02	Dominates
583716E-07	-0.2817183	
841858E-07	-0.2817183	
920929E-07	-1.281718	
604645E-08	-5.281718	
802322E-08	2.718282	
901161E-08	2.718282	

there is a limit to how well the derivative of exp (x_0) at $x_0 = 1$ can be roximated by the finite difference formula $D(h)$. The approximation error is)hed against step size h in Figure 1.2. (The scales are logarithmic.)



GURE 1.2 Error Versus Step Size

In view of the foregoing remarks, computers might seem to be unexpectedly limited instruments, but one should note that compared to human beings, computers can do those operations in capability 2 very quickly. Therein lies the beauty and power of digital computation. But exploitation of this power ultimately requires that someone approximate more complicated operations [such as evaluating sin (x)] by a sequence of arithmetical operations and comparisons.

1.2 CONCEPTS OF COMPUTATION ERROR

Toward surveying the ground to be covered in this book, let us sharpen some notions introduced informally in the preceding section. We will view a *program* as a finite sequence of instructions involving only comparisons and arithmetic operations on stored numbers, and conditional branches from one place to another in the sequence of instructions. For a concrete example, think of FORTRAN. We disallow library functions such as sin (x) and exp (x) until Chapter 2, where their rationale is revealed. Also, input and output facilities are of little concern to us.

With this notion of "program" in mind, we define:

Roundoff error: The error introduced in approximating a given number by a computer number.
Truncation error: The error introduced by approximating some desired mathematical operation by computations directed by a program. It is presumed that computations are done without roundoff error.

The term *truncation error* stems from recognition that many numerical methods are constructed by first finding a Taylor's series representation $\Sigma_{j=0}^{\infty} \alpha_j x^j$ of the mathematical operation, and then computing a truncation [i.e., an initial (polynomial) segment $\Sigma_{j=0}^{n} \alpha_j x^j$] of this series. We will find that polynomial approximations and power series expansions lie behind a large proportion of the computational methods to be encountered. Analysis of truncation error is strongly dependent on the application area and computational methodology under consideration. Meaningful discussion must therefore await the special topics of later chapters.

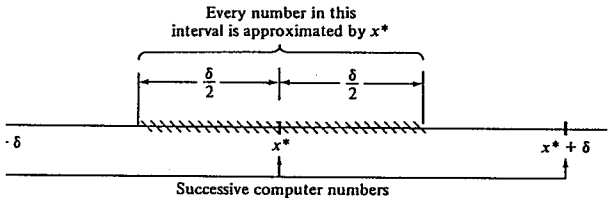
Roundoff error has its origins in computer operations regardless of problem area. In this chapter we describe these origins and examine some frequently encountered settings in which its effects can be distressing if not confronted intelligently.

Roundoff error tends to become a bothersome and perhaps a limiting factor when at some point in a computation, small changes of an input parameter to a calculation result in relatively large deviations of the calculated output.

Linear equations have the form

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i, \qquad 1 \le i \le m,$$

and arise quite naturally from engineering considerations. In many cases linear equations have solutions that depend sharply on the coefficient values a_{ij}. For such equations, roundoff error is a prime factor.



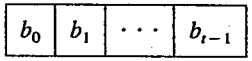
URE 1.3 Computer Number Approximation

egardless of problem area, roundoff error limits the accuracy to which a ion can be computed. Assume that in the neighborhood of a solution x, the between successive computer numbers is δ. If the computer approximates the closest computer number, clearly we cannot count on being able to lo- more closely than $\delta/2$. For if near the computer number x^*, the gap between ssive computer numbers is δ, any number x between $x^* - \delta/2$ and $x^* +$ s stored as x^*. We have illustrated this in Figure 1.3. Any number x in the hatched region must be approximated by x^*. Some computers use schemes than the nearest-machine-number rule, and even larger error can result. We ss approximation strategies later in this chapter.

oward understanding the source and magnitude of roundoff error, we need to a clearer notion of how computers store numbers. This topic is considered

ER REPRESENTATION OF NUMBERS

some exceptions, a computer number is stored in one, two, or four computer s. For purposes of this discussion, one may imagine a *computer word* to be ary string (i.e., a string of 0's and 1's) with t coordinates, where t is dependent achine design. That is, a computer word may be visualized as the array



LE 1.5 Word Length of Some Common Computers

Computer	*Word Length, t*
ol Data Corporation	
C 6000, 7000, and CYBER series	60
Corporation	
M 360/370 series and 303X, 308X series	32
M PC, AT, and XT	16
al Equipment Corporation	
C 10	36
P 11 series	16
X 11 series	32
e computers	32

where the b_j's are either 0 or 1. These terms b_j are referred to as binary coefficients, or *bits*, in the sense that decimal coefficients are known as "digits." We have tabulated the word lengths of some popular computer families in Table 1.5.

Physically, the bits of a computer word are stored by means of two-state electronic circuits that are embedded in integrated circuit chips or alternatively by the direction of flux in magnetic elements. Other storage devices appear to be on the horizon.

Everything—machine instructions, alphanumeric strings, memory pointers, and so on—not just computer numbers, must be stored as machine words. The problem of computer number representation is the problem of finding a sensible coding scheme for mapping real numbers into machine words.

1.3.1 Number Systems

Before proceeding to the details of machine numbers, we remind the reader of a few facts about number representation. Our conventional number system is decimal. The symbol 546.3 in the decimal number system designates the sum

$$\underline{5} \times 10^2 + \underline{4} \times 10^1 + \underline{6} \times 10^0 + \underline{3} \times 10^{-1}.$$

Thus 546.3 can be regarded as a member of a code for designating nonnegative numbers. The technique of the decimal representation can be generalized to any integer *base* $N > 1$. Assuming for each j that a_j is in the set $\{0, 1, \ldots, N-1\}$, define the string $(a_k a_{k-1} \cdots a_0)_N$ to be the code word for the integer

$$M = a_k N^k + a_{k-1} N^{k-1} + \cdots + a_0 N^0. \tag{1.1}$$

For reasons mentioned at the outset of this section, the *binary* ($N = 2$ and $a_j = 0$ or 1) system is the natural base for machine number representation.

EXAMPLE 1.3

Here we decode the string $(10110)_2$. In this specific case, (1.1) takes the form

$$M = \underline{1} \times 2^4 + \underline{0} \times 2^3 + \underline{1} \times 2^2 + \underline{1} \times 2^1 + \underline{0} \times 2^0 = 2^4 + 2^2 + 2 = (22)_{10}.$$

In brief, $(10110)_2 = (22)_{10}$. ■

Extending these ideas, and confining attention to the binary system, we find that any positive number x can be represented as

$$x = a_k 2^k + \cdots + a_0 2^0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + \cdots. \tag{1.2}$$

We may unambiguously write $(a_k \cdots a_0 . a_{-1} a_{-2} \cdots)_2$ for x, with the understanding that (1.2) gives the conversion. The radix point is positioned between a_0 and a_{-1}. It partitions the number into its integral and fractional parts.

:AMPLE 1.4

evaluating the right side of the equation below according to (1.2), the reader ll see that $(1.8)_{10}$ has the nonterminating binary representation

$$(1.8)_{10} = (1.110011001100\ldots)_2,$$

which the pattern 1100 repeats ad infinitum. ■

The binary representation just discussed is fundamental to all three principal nputer representation systems, which are

1. Integer representation.
2. Floating-point representation.
3. Multiple-precision representation.

ese systems are described below in this order.

teger Representation

ure 1.4 shows by vertical lines the locations of computer integers on a segment the real number line. Note that the integers form a uniform grid: The distance tween neighboring integers is always 1, regardless of the magnitudes of these mbers. The implication is that the roundoff error encountered in approximating iven positive number x by an integer can be just as large for small values of x for large values.

The most obvious encryption of integers into computer words is the *signed gnitude* method. Here the first (b_0) bit indicates the sign of the number, and remaining bits b_j, $1 \le j \le t-1$, store the coefficients a_i, where, as in (1.1),

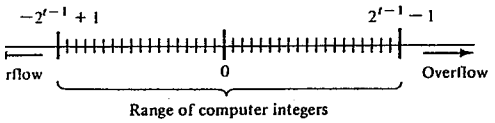
$$|M| = a_k 2^k + a_{k-1}2^{k-1} + \cdots + a_0. \tag{1.3}$$

re, if $2^{t-1} > M \ge 0$, the bits a_i in the representation above are stored directly h the understanding that $a_j = 0$, $t - 1 \ge j > k$:

a_{t-1}	$\cdots$	a_3	a_2	a_1	a_0

comparing the format above with the bit designation at the beginning of Section , we see that

$$b_j = a_{t-j-1}, \qquad 1 \le j \le t-1.$$



GURE 1.4 Distribution of Computer Integers

For reasons of speed and convenience in hardware design, negative integers are often represented by the *two's-complement* technique. Under this regime, for M negative, the bits of the positive integer representation of $2^t - |M|$ are stored. It turns out that the coefficients of $2^t - |M|$ can be obtained in the following way. First, replace all the 0's in the binary representation (1.3) of $|M|$ by 1's and all the 1's by 0, and then add 1 to the result.

EXAMPLE 1.5

We give the DEC 10 representation of 155 and -155. Recall from Table 1.5 that the word length of this computer is $t = 36$. One may check that

$$155 = (10011011)_2.$$

If the integer number is represented in the binary system, the binary coefficients may be stored directly, with a_0 placed in the rightmost box, b_{35}; a_1 stored in the next-to-last box, b_{34}; and so on. Consequently, the DEC 10 integer representation of 155 is

all 0's

j:	0	1		26	27	28	29	30	31	32	33	34	35
b_j:	0	0	$\cdots$	0	0	1	0	0	1	1	0	1	1

The two's complement representation for -155 is the reverse ordering of the coefficients of $2^{36} - 155$, which is

all 1's

j:	0	1		26	27	28	29	30	31	32	33	34	35
b_j:	1	1	$\cdots$	1	1	0	1	1	0	0	1	0	1

This representation can be obtained, as stated earlier, by taking the complements of the bits in the representation for $+155$ and adding 1, in binary, to the resulting string. ■

EXAMPLE 1.6

If some arithmetic operation results in an integer too large to store, some computers "remember" only the least significant t bits of the binary representation. The program and output in Tables 1.6 and 1.7, from a PDP 11, illustrates this point. In this case, $t = 16$, so the magnitude of the largest integer representation is bounded by $2^{15} - 1 = 32767$. Note what happens when the number accumulated by the variable KSUM exceeds this bound.

ıBLE 1.6 Program for Integer Overflow

```
      PROGRAM OVRFLO
C     ***************************************************
C     THIS PROGRAM ILLUSTRATES THE CONSEQUENCES OF
C     INTEGER OVERFLOW ON THE PDP 11
C     OUTPUT: KSUM=SUM OF 20 INTEGERS
C     ***************************************************
      KSUM=0
      NUM=5000
      DO 1 I=1,20
         KSUM=KSUM+NUM
         WRITE(10,*) KSUM
    1 CONTINUE
      STOP
      END
```

ıBLE 1.7 Printout for Example 1.6

Sum
5000
10000
15000
20000
25000
30000
-30536
-25536
-20536
-15536
-10536
-5536
-536
4464
9464
14464
19464
24464
29464
-31072

oating-Point Representation

e *floating-point representation*, and its close cousin, the multiple-precision rep-ıentation, are the most important machine number systems for numerical com-tations.

The floating-point representation is conveniently described in terms of what is own as the *normal-form representation*, which we define next. If x is a positive mber, then for some unique integer E and real number f such that $\frac{1}{2} \le f < 1$, : may write

$$x = f \cdot 2^E. \tag{1.4}$$

ır if x is written in the form (1.2), then

$$x = a_k 2^k + a_{k-1}2^{k-1} + \cdots + a_0 + a_{-1}2^{-1} + a_{-2}2^{-2} + \cdots$$
$$= (a_k 2^{-1} + a_{k-1}2^{-2} + \cdots) \cdot 2^{k+1}.$$

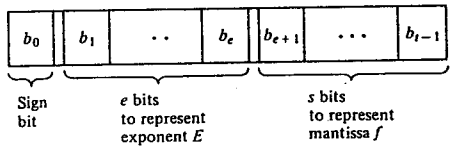


FIGURE 1.5 Segmentation of Computer Word for Floating Point Representation

With k so determined, we see that

$$f = a_k 2^{-1} + a_{k-1}2^{-2} + \cdots \quad \text{and} \quad E = k + 1.$$

For instance,

$$x = (1001.01)_2 = 2^3 + 2^0 + 2^{-2} = (2^{-1} + 2^{-4} + 2^{-6}) \cdot 2^4.$$

In this example, $f = 2^{-1} + 2^{-4} + 2^{-6}$ and $E = 4$. The integer E as described above is called the *exponent* of the representation, and the term f, $\frac{1}{2} \le f < 1$, is the *mantissa*. Assume that x is represented in normal form (which is a purely mathematical construct):

$$x = f \cdot 2^E, \qquad \tfrac{1}{2} \le f < 1, \qquad E \text{ integer}.$$

The number x is stored in a computer word as a floating-point number as follows:

1 bit stores the sign of x.
s bits store the mantissa f.
e bits store the exponent E.

The storage lengths s and e are usually fixed by machine design. If only one computer word, with the length t, is used, we would expect that

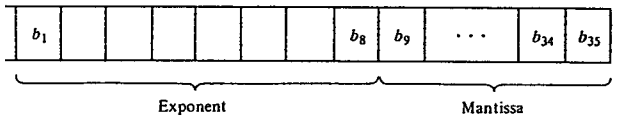
$$t = 1 + s + e.$$

If need be, the computer will round or chop the mantissa to the most significant s bits. (By observing that the most significant bit of f must be 1, some computers (e.g., VAXs) have effectively $s + 1$ bits of storage). The bit allocation of the floating-point number is illustrated in Figure 1.5.

EXAMPLE 1.7

We now describe the floating-point-number representation employed by the DEC 10 computer. In this representation, the first bit (b_0) of the binary string of length $t = 36$ gives the sign of the number (here 0 means "positive" and 1 means "negative"). The exponent is represented by bits 1 to 8 and the mantissa is given by the binary coordinates 9 to 35, as shown in Figure 1.6. Let b_j denote the bit stored in location j, $0 \le j \le 35$, in Figure 1.6. For positive numbers the mantissa is determined according to the rule

$$f = (0.b_9 b_{10} \cdots b_{35})_2 = b_9 \cdot 2^{-1} + b_{10} \cdot 2^{-2} + \cdots + b_{35} \cdot 2^{-27}.$$

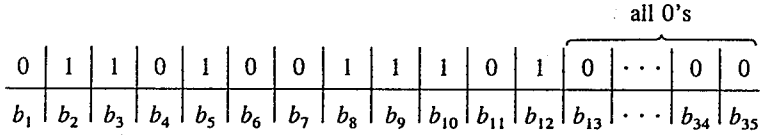


URE 1.6 DEC 10 Floating-Point Number Representation

conversion between the exponent E and the stored bits b_j $(1 \le j \le 8)$ is given

$$E = (b_1 b_2 \cdots b_8)_2 - 128 = -(1 - b_1) \cdot 128 + (b_2 b_3, \ldots, b_8)_2.$$

rom these considerations, one may conclude that a number may be represented floating-point number on a DEC 10 computer only if the magnitude of its ponent E lies between $(-128)_{10}$ and $(127)_{10}$. If, additionally, the mantissa f in normal-form representation (1.4) is an integral multiple of 2^{-27}, then x is tly equal to its DEC 10 floating-point representation. Otherwise, the mantissa btained by rounding f to 27 binary places.
Consider the specific DEC 10 floating-point number



calculate that

$$E = (01101001)_2 - 128 = 2^6 + 2^5 + 2^3 + 2^0 - 128$$
$$= 64 + 32 + 8 + 1 - 128 = -23$$

$$f = (2^{-1} + 2^{-2} + 2^{-4}) = (0.8125)_{10},$$

$$x = (0.8125)_{10} \times 2^{-23} \approx (9.7 \times 10^{-8})_{10}.$$

■

gure 1.7 illustrates the distribution of floating-point numbers on a segment e real number line. As in Figure 1.5, e and s are the numbers of bits used xponent and mantissa representations. In contrast to the integer representation, ing-point numbers are more dense near the origin and more sparse at the mes. It is this feature that gives the floating-point representation the capacity eserve significant digits, as discussed later in this chapter. If the number x e approximated has small magnitude, the real representation error will be spondingly small. In fact, the gap between a real number x and its floating-t representation x^* is essentially proportional to the magnitude of x itself. In ext section we will see how to find the constant of proportionality.
Figure 1.7 we have also illustrated a gap between zero and the smallest

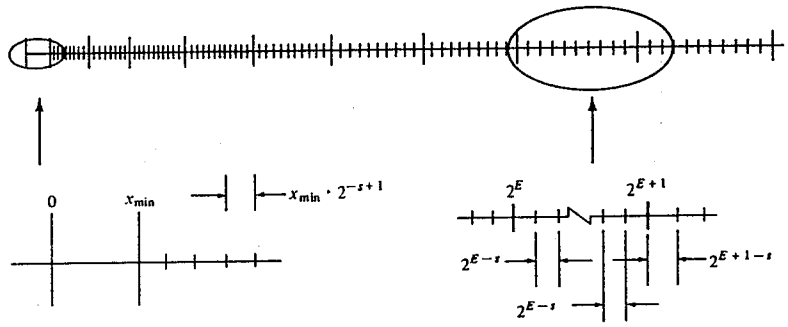


FIGURE 1.7 Distribution of Floating-Point Numbers

positive floating-point number, $x_{\min}$. We give a detailed derivation of $x_{\min}$. If e bits store the exponent E, then $E_{\min}$, the minimum value of E, is

$$E_{\min} = -2^{e-1}.$$

Also, $f \ge \frac{1}{2}$. So

$$x_{\min} = 2^{-1} \cdot 2^{E_{\min}} = 2^{-1} \cdot 2^{-2^{e-1}}.$$

The next larger floating-point number x is obtained by adding 1 to the least significant bit of f, so the new mantissa equals $2^{-1} + 2^{-s}$. Thus

$$x = (2^{-1} + 2^{-s}) \cdot 2^{E_{\min}} = x_{\min}(1 + 2^{-s+1}).$$

In the *multiple-precision representation,* two or more computer words are used to encode a given number x. The rationale of multiple-precision representation is exactly that of floating-point representation; the only difference is that the number of available storage bits is increased.

In Table 1.8 we have tabulated the number of bits assigned to the mantissas

TABLE 1.8 Floating-Point Characteristics of Some Computers*

	Single Precision		*Multiple Precision*	
Computer	*s*	e	*s*	e
Control Data Corporation				
CDC 6000, 7000, and CYBER series	48	11	96	11
IBM Corporation				
IBM 360/370 series and 303X and 308X series	24	7	56	7
IBM PC†, AT, and XT	24	7	56	7
Digital Equipment Corporation				
DEC 10	27	8	63	8
PDP 11 series†	23	8	55	8
VAX 11 series	23	8	55	8
Prime computers	23	8	47	15

*s, number of mantissa bits; e, number of exponent bits.
†Uses two computer words for a single-precision floating-point number.

and the exponents of floating-point and double-precision numbers in various computers.

JNDOFF ERROR

Roundoff Error Bounds and Machine Epsilon

Recall the normal-form representation $x = f \cdot 2^E$, with $\frac{1}{2} \le f < 1$. By adding 1 to the least significant place of the floating-point mantissa, we conclude that if for a given computer, s is the number of floating-point mantissa coefficients, then the difference between x^* and the next larger floating-point number, say y^*, is $2^{-s} \cdot 2^E$. As in Figure 1.3, let δ denote the separation of computer numbers. Then $\delta = 2^{E-s}$. The roundoff error in approximating a number x between x^* and y^* is bounded by 2^{E-s-1}, assuming that the computer finds the nearest floating-point number to x. This procedure is called *rounding*. Define *machine epsilon*, or simply ε, to be 2^{-s}. Note that since $f \ge \frac{1}{2}$, $|x| \ge 2^{E-1}$. We combine this with our earlier (rounding) bound to write

$$|x - x^*| \le \frac{\delta}{2} = 2^{-s+E-1} = 2^{-s} \cdot 2^{E-1} \le 2^{-s}|x| = \varepsilon|x|.$$

In summary,

$$\boxed{|\text{roundoff error } (x)| = |x - x^*| \le \varepsilon|x|.} \tag{1.5}$$

Intuitively, $1 + \varepsilon$ is the smallest number greater than 1 that the computer in question will distinguish from 1.

It is good practice to design programs that are "portable" in the sense that their performance does not depend on which computer is being used. We saw in

TABLE 1.9 Subroutine EPS for Finding Machine Epsilon

```
      SUBROUTINE EPS(E)
C
C     ************************************************************
C     *  FUNCTION: THIS SUBROUTINE COMPUTES THE MACHINE EPSILON  *
C     *  USAGE:                                                  *
C     *        CALL SEQUENCE: CALL EPS(E)                        *
C     *  PARAMETERS:                                             *
C     *        INPUT:                                            *
C     *              NONE                                        *
C     *        OUTPUT:                                           *
C     *              E=MACHINE EPSILON                           *
C     ************************************************************
C
      E=1.0
      DO WHILE(E+1.0.GT.1.0)
         E=E/2.0
      END DO
      E=2.0*E
      RETURN
      END
```


Table 1.8 that the number s of mantissa bits, and hence machine epsilon, can vary widely. Therefore, it is often advisable to include in a program a code that automatically estimates machine epsilon. In Table 1.9, subroutine EPS for computing machine epsilon is offered. Certain subroutines require the user to supply a "stopping rule" threshold. The choice of such a value should be based on an estimate of machine epsilon.

EXAMPLE 1.8

Machine epsilon of the DEC 10 computer was obtained by means of subroutine EPS. To three significant decimal places, the output was 7.45×10^{-9}. This number was actually 2^{-27}, which accords with the entry $s = 27$ in Table 1.8 for this machine.

With regard to the experiment described in Example 1.1, examination of the output Table 1.2 reveals that the observed roundoff error in the number actually stored for input $x = 0.1234567890123$ was 1.7×10^{-10}. By the considerations in this section, this error should be less than $\varepsilon \cdot |x| \approx 9.2 \times 10^{-10}$. Thus machine epsilon did lead to a reasonably close upper bound for the observed roundoff error. ■

1.4.2 Roundoff Error in Action

Here we present two examples illustrating common situations in which a computed solution is needlessly inaccurate as a result of roundoff error. These examples show further that mathematical and computational reasoning can be beneficial. For in each instance, once we see the origin of the problem, we are able to devise a more cunning computational strategy and thereby achieve an accurate answer. The ultimate source of difficulty in these examples is clearly roundoff error. Were computers able to store and operate on real numbers, rather than floating-point approximations, these difficulties would not arise.

EXAMPLE 1.9

We here examine a common "trapdoor" known as *subtractive cancellation*. It arises when two nearly equal floating-point numbers are subtracted from one another.

For convenience of notation, assume that we have a machine with decimal words of mantissa length $s = 4$. Suppose that we are to estimate the difference of square roots of two given integers.

$$y = \sqrt{1985} - \sqrt{1984}.$$

and we are given the rounded approximations

$$\sqrt{1985} = 44.55$$
$$\sqrt{1984} = 44.54.$$

Then blind subtraction would have us estimate y by

$$\hat{y} = 44.55 - 44.54 = 0.01.$$

'he correct answer to the accuracy shown is 0.011224. Could we have done better vith the given approximations? Yes! The basic trouble is that the definition of y ias us subtract two very nearly equal numbers, and the only information about heir difference resides in their unequal digits, which in this case is only the last igit. There are several paths that bypass this subtractive cancellation problem. tecall from algebra that

$$\sqrt{A} - \sqrt{B} = \frac{A - B}{\sqrt{A} + \sqrt{B}}. \tag{1.6}$$

[ere we presume that A and B are exact, so no cancellation occurs in the numerator f (1.6), and the denominator retains nearly four significant digits of accuracy. pecifically, by use of the right side of (1.6), we compute the new estimate

$$\tilde{y} = \frac{1}{44.55 + 44.54} = 0.011225.$$

ere the error resulting from using the rounded values is only about 10^{-6}. ■

XAMPLE 1.10

'om Taylor's series developments in calculus books (and as we learn in Chapter , it is known that

$$e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}. \tag{1.7}$$

ıe error of this approximation does not exceed

$$\frac{|x|^{n+1}}{(n+1)!} \max\{1, e^x\}.$$

:call that for a positive integer j,

$$j! = j \times (j-1) \times (j-2) \times \cdots \times 1.$$

sensible computational procedure would seem to be to sum terms $x^j/j!$ until e magnitude of the addends is less than machine epsilon. The program listed in ıble 1.10 follows this strategy and on a VAX, computes an approximation, ıich we denote by $S(-8)$, of e^{-8} to be 0.319328×10^{-3}. The correct value, the accuracy shown, is $0.3354626. \times 10^{-3}$.

The error of the estimate $S(-8)$ is 1.6×10^{-5}, and the answer is correct only the first significant decimal. The fundamental trouble is that whereas the mag-tude of the final sum is relatively small, as shown in Table 1.11, several of the rms $(-8)^j/j!$ are large, and these large terms, while eventually canceling, de-rmine the number of significant places. Henrici (1982) has termed this phenom-ıon *smearing*. Smearing can be anticipated whenever magnitudes of individual rms in a summation are considerably larger than the sum itself. Since the error a stored number x is in the neighborhood of $\varepsilon |x|$, in the notation of (1.5), the

TABLE 1.10 Program for Approximation of Exponential Function

```
C     PROGRAM TAYEXP
C
C     ************************************************************
C     COMPUTE THE TAYLOR SERIES APPROXIMATION OF EXP(X), (X=-8.)
C     OUTPUT: S=THE TAYLOR SERIES APPROXIMATION
C             TRUE=THE VALUE OF EXP(X) AS COMPUTED BY THE
C                  LIBRARY FUNCTION EXP
C             ERR=THE DIFFERENCE BETWEEN THE TRUE VALUE
C                 AND THE TAYLOR SERIES APPROXIMATION
C     ************************************************************
C
      S=0.
      A=1.
      X=-8.
      J=0
C
C     *** THIS LOOP WILL COMPUTE THE TAYLOR SERIES APPROX.    ***
C     *** UNTIL ADDEND IS NEAR MACHINE EPS                    ***
C
      DO WHILE(ABS(A).GT.1.E-7)
         WRITE(10,*)J,A
         S=S+A
         A=A*(X/(J+1))
         J=J+1
      END DO
      TRUE=EXP(X)
      ERR=TRUE-S
      WRITE(10,*)S,TRUE,ERR
      STOP
      END
```

TABLE 1.11 Addends in Series for exp (−8)

j	$\frac{(-8)^j}{j!}$
0	1.000000
1	-8.000000
2	32.00000
3	-85.33334
4	170.6667
5	-273.0667
6	364.0889
7	-416.1017
8	416.1017
9	-369.8681
10	295.8945
11	-215.1960
12	143.4640
13	-88.28555
14	50.44889
15	-26.90608
16	13.45304
17	-6.330842
18	2.813707
19	-1.184719
20	0.4738875
21	-0.1805286

error in computing a sum is typically at least as large as ε times the magnitude of the largest addend. If the sum itself is much smaller than the largest addend, smearing will almost surely occur, as in this example. The moral: Beware of summing a series with mixed signs.

For the particular case of evaluating $\exp(-8)$, smearing can be sidestepped by observing that for $x = -1$, for instance, the largest addends have the same order of magnitude as the sum. Thus the procedure in Table 1.10 should be accurate for finding an approximation $S(-1)$ of e^{-1}. In fact, with $x = -1.0$ the program in Table 1.10 computes the estimate 0.36787945, which is correct in all but the last decimal place. Then, since $e^{-8} = (e^{-1})^8$, one may offer $S(-1)^8$ as an estimate. The error in following this tactic is only about 2×10^{-9}, whereas direct use of the algorithm resulted in an error of 1.6×10^{-5}.

■

LUTE AND RELATIVE ERROR BOUNDS

The error analysis of computations is characterized according to several different criteria. Let x denote an exact value and let variable x^* denote an approximation of x. One may think of x^* as the result of a computation. Then the absolute difference $|x - x^*|$ gives the *absolute error*. Any nonnegative number $\delta(x^*)$ satisfying the inequality $|x - x^*| \leq \delta(x^*)$ is called an *absolute bound* or *upper bound* for the error of x^* as an approximation of x. Note that all values larger than $\delta(x^*)$ also serve as upper bounds.

An absolute error bound $\delta(x^*)$ for a floating-point approximation x^* that rounds a nonzero input x to the closest value is, in view of developments in Section 1.4, given by $\delta = \delta(x^*) = \varepsilon|x^*|$.

The *relative error* of approximating x by x^* is defined to be $|x - x^*|/|x|$. We summarize these definitions as follows:

$$\boxed{\begin{aligned}\text{absolute error} &= |\text{true value} - \text{approximation}| \\ \text{relative error} &= \frac{\text{absolute error}}{|\text{true value}|}.\end{aligned}} \tag{1.8}$$

We will use the notation $\delta(x^*)$ to denote an absolute bound for the error in approximating x by x^*, and Rel (x^*), a relative error bound.

A feature of integer representation is that the absolute roundoff error bound is always $\frac{1}{2}$ (assuming that rounding to the nearest integer is done). The relative error depends on what the number x to be approximated happens to be. By contrast, in floating-point representation, the absolute error depends on x, but in view of (1.5), Rel $(x^*) = \varepsilon$, ε being machine epsilon. In summary, integer representation assures that the absolute error bound is constant in x, and floating-point representation preserves the relative error.

In accordance with usage in our description of floating-point numbers, we say that $x = f \cdot 10^E$ is a normal-form representation of x if $0.1 \leq f < 1$ and E is an integer. Let x^* be an approximation of x. We say that x^* has *k significant digits* if

$$|x - x^*| \leq \tfrac{1}{2} \cdot 10^{-k+E}$$

[There is some divergence on the definition of "significant"; we follow Rice (1983), for example, in ours.]

EXAMPLE 1.11

If $x^* = 12.765$ and $x = 12.8111$, then

$$x = 0.128111 \times 10^2$$

and $|x^* - x| < \frac{1}{2} \times 10^{-1}$. Since $E = 2$ in the normal-form representation of x^*, one concludes that $k = 3$, and therefore x^* has three significant digits.

■

1.6 ERROR PROPAGATION

1.6.1 Propagation of Error in a Single Computation

We now derive upper bounds for the errors that result from computing functional values using incorrect data. Assume that x^* is an approximation of x. If the functional value $f(x)$ is desired, and only the approximating value x^* of x is known, we approximate $f(x)$ by $f(x^*)$. Assume that $\delta(x^*)$ is an absolute error bound for $|x - x^*|$ and that the function $f(x)$ is differentiable. Then the mean-value theorem of calculus (Appendix B, item 19) implies that

$$f(x) - f(x^*) = f'(\zeta)(x - x^*), \tag{1.9}$$

where ζ is some number in the interval with endpoints x and x^* and $f'(\zeta)$ is the derivative of $f(x)$ at ζ. Since the value of ζ is unknown, equation (1.9) cannot be used directly. To obtain practical formulas, we have to make further assumptions about function f.

Assume first that for arbitrary t between x^* and x, and for some fixed number D,

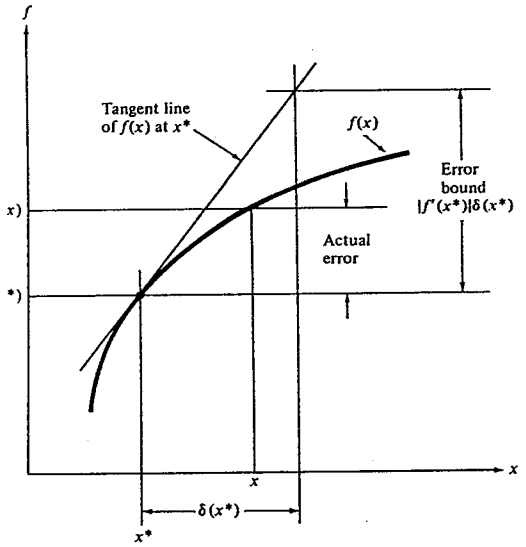
$$|f'(t)| \leq D.$$

Then (1.9) implies that

$$|f(x) - f(x^*)| = |f'(\zeta)| \cdot |x - x^*| \leq D\ \delta(x^*);$$

consequently, the quantity $D\ \delta(x^*)$ can be accepted as an error bound for $f(x^*)$. That is, for the interval I containing all points between x^* and x,

$$\delta(f(x^*)) = \max_{t \in I} |f'(t)|\ \delta(x^*). \tag{1.10}$$



IGURE 1.8 Error Bound for Operation of Inexact ata

he application of this error bound has the disadvantage that an upper bound for ıe derivative f' of $f(x)$ must be determined.

Assume next that $\delta(x^*)$ is small, $f'(x^*) \neq 0$ and $f'(t)$ is nearly constant near *. Then (1.10) implies that $D \approx |f'(x^*)|$. Consequently,

$$\boxed{\delta(f(x^*)) \approx |f'(x^*)|\, \delta(x^*).} \tag{1.11}$$

igure 1.8 illustrates the error-bound construct (1.11).

XAMPLE 1.12

et $f(x) = \sqrt{x}$ and assume that x^* is an approximation of x with an error bounded y $\delta(x^*)$. Then (1.11) implies that

$$\delta(\sqrt{x^*}) = \frac{1}{2\sqrt{x^*}}\,\delta(x^*).$$

, for example, $x^* = 4.00$ and $\delta(x^*) = 0.005$, then

$$\delta(\sqrt{4.00}) = \frac{1}{2\sqrt{4.00}} \times 0.005 = 0.00125.$$

n fact, if $x = x^* - \delta(x^*) = 3.995$, then

$$\sqrt{x} - \sqrt{x^*} = -0.0012504.$$

n this case, the approximation (1.11) does give an accurate error estimate. ■

Error bounds for functions of two or more variables can also be obtained by the approach described above. Consider a real-valued function $f(x, y)$ of two variables x and y. Assume that the "true" values x and y are unknowns but their approximations, x^* and y^* are given. Then the desired value $f(x, y)$ is approximated by $f(x^*, y^*)$. Assume again that f is differentiable, the error bounds $\delta(x^*)$ and $\delta(y^*)$ are small, and the first partial derivatives are not both zero and change slowly. Then

$$f(x, y) - f(x^*, y^*) = f(x, y) - f(x^*, y) + f(x^*, y) - f(x^*, y^*).$$

Consequently,

$$\begin{aligned} |f(x, y) - f(x^*, y^*)| &\leq |f(x, y) - f(x^*, y)| + |f(x^*, y) - f(x^*, y^*)| \\ &\approx \left|\frac{\partial f}{\partial x}(x^*, y)\right| \cdot |x - x^*| + \left|\frac{\partial f}{\partial y}(x^*, y^*)\right| \cdot |y - y^*| \\ &\approx \left|\frac{\partial f}{\partial x}(x^*, y^*)\right| \cdot |x - x^*| + \left|\frac{\partial f}{\partial y}(x^*, y^*)\right| \cdot |y - y^*| \\ &\leq \left|\frac{\partial f}{\partial x}(x^*, y^*)\right| \cdot \delta(x^*) + \left|\frac{\partial f}{\partial y}(x^*, y^*)\right| \cdot \delta(y^*). \end{aligned}$$

Hence analogously to (1.11),

$$\boxed{\delta(f(x^*, y^*)) \approx \left|\frac{\partial f}{\partial x}(x^*, y^*)\right| \delta(x^*) + \left|\frac{\partial f}{\partial y}(x^*, y^*)\right| \delta(y^*).} \tag{1.12}$$

Generalization of this approach to more variables should be evident. If x_i^* is an estimate of x_i, $1 \leq i \leq n$, and $\delta(x_i^*)$ denotes a bound for $|x_i - x_i^*|$, $1 \leq i \leq n$, then with $x^* = (x_1^*, \ldots, x_n^*)$, we have

$$\begin{aligned} \delta(f(x_1^*, \ldots, x_n^*)) \approx &\left|\frac{\partial f(x^*)}{\partial x_1}\right| \delta(x_1^*) + \left|\frac{\partial f(x^*)}{\partial x_2}\right| \delta(x_2^*) + \cdots \\ &+ \left|\frac{\partial f(x^*)}{\partial x_n}\right| \delta(x_n^*). \end{aligned} \tag{1.13}$$

ENGINEERING EXAMPLE

We given an upper bound for the error of the computation

$$t = 2\pi\sqrt{\frac{l}{g}}$$

if π, l, and g are not known precisely. The relation above gives the time period of a "linearized" pendulum (see Figure 1.9) with g the acceleration of gravity and l the pendulum length, which we take as being nominally 3 m, with an error bound $\delta(l)$ of 1 cm. As usual, π is the ratio of circle circumference to diameter.

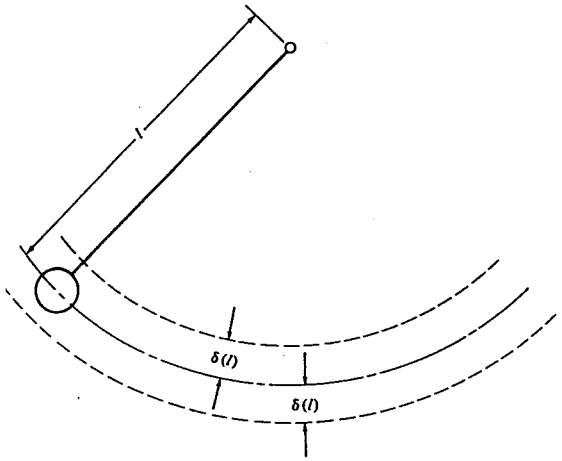


FIGURE 1.9 Uncertainty in Pendulum Length

In evaluating π by $4.0 \times \tan^{-1}$, one can approximate π with an error bound of $\delta(\pi) = \pi \cdot$ (machine epsilon). This term will give a negligible contribution to the error. Gravitational acceleration at 45° latitude is nominally 9.80 m/s^2 and varies from location to location. Unless site measurements are made, $\delta(g) \approx 10^{-2}$ m/s^2. With these error bounds in hand, we are ready to proceed with our computation. The nominal period will be

$$t = 2\pi \frac{\sqrt{3}}{\sqrt{9.80}} \approx 3.476.$$

By the use of the three-variable version of (1.13), we conclude that

$$\begin{aligned}\delta(t) &\approx \left|\frac{\partial t}{\partial \pi}\right| \delta(\pi) + \left|\frac{\partial t}{\partial l}\right| \delta(l) + \left|\frac{\partial t}{\partial g}\right| \delta(g) \\ &= 2\sqrt{\frac{l}{g}}\,\delta(\pi) + 2\pi \frac{1}{2\sqrt{l/g}} \frac{1}{g}\,\delta(l) + 2\pi \frac{1}{2\sqrt{l/g}} \frac{l}{g^2}\,\delta(g).\end{aligned}$$

As stated, the $\delta(\pi)$ term is negligible compared to the other uncertainties, so

$$\begin{aligned}\delta(t) &\approx \frac{\pi}{\sqrt{lg}}\,\delta(l) + \frac{\pi\sqrt{l}}{\sqrt{g^3}}\,\delta(g) \\ &= 0.58 \times 0.01 + 0.18 \times 0.01 \\ &= 0.0076 \text{ s}.\end{aligned}$$

In summary,

$$t = 3.48 \pm 0.01 \text{ s}.$$

■

*1.6.2 Error Propagation in Arithmetic

Bounds for error in arithmetic can be derived from the general rule (1.12), as the following example illustrates.

EXAMPLE 1.13

Assume that $f(x, y) = x + y$. Then

$$\frac{\partial}{\partial x} f(x^*, y^*) = \frac{\partial}{\partial y} f(x^*, y^*) = 1,$$

so, by (1.12),

$$\delta(f(x^*, y^*)) = \delta(x^* + y^*) = 1 \times \delta(x^*) + 1 \times \delta(y^*) = \delta(x^*) + \delta(y^*).$$

In addition, then, the error bounds are added.

Assume next that $f(x, y) = x - y$. Then

$$\delta(f(x^*, y^*)) = \delta(x^* - y^*) = 1 \times \delta(x^*) + 1 \times \delta(y^*) = \delta(x^*) + \delta(y^*).$$

Hence in subtraction the error bounds also add.

If we define $f(x, y) = xy$, then

$$\delta(f(x^*, y^*)) = \delta(x^*y^*) \approx |y^*|\,\delta(x^*) + |x^*|\,\delta(y^*),$$

and if $f(x, y) = x/y$, then

$$\delta(f(x^*, y^*)) = \delta\left(\frac{x^*}{y^*}\right) \approx \frac{1}{|y^*|}\,\delta(x^*) + \frac{|x^*|}{|y^*|^2}\,\delta(y^*) = \frac{|y^*|\,\delta(x^*) + |x^*|\,\delta(y^*)}{|y^*|^2}.$$

■

EXAMPLE 1.14

Consider the function

$$f(x, y, z) = xy + z,$$

where

$$\begin{aligned} x \approx x^* &= 2.20, & \delta(x^*) &= 0.005 \\ y \approx y^* &= 1.15, & \delta(y^*) &= 0.005 \\ z \approx z^* &= 3.05, & \delta(z^*) &= 0.005. \end{aligned}$$

Then repeated application of the relation above gives that

$$\delta(x^*y^*) = |1.15| \times 0.005 + |2.20| \times 0.005 = 0.01675$$

and therefore

$$\delta(x^*y^* + z^*) = \delta(x^*y^*) + \delta(z^*) = 0.02175.$$

■

TABLE 1.12 Absolute Error Bounds for Arithmetic Operations on Inexact Data

$$\delta(x^* + y^*) = \delta(x^*) + \delta(y^*)$$
$$\delta(x^* - y^*) = \delta(x^*) + \delta(y^*)$$
$$\delta(x^*y^*) \approx |x^*|\,\delta(y^*) + |y^*|\,\delta(x^*)$$
$$\delta\left(\frac{x^*}{y^*}\right) \approx \frac{|x^*|\,\delta(y^*) + |y^*|\,\delta(x^*)}{|y^*|^2}$$

TABLE 1.13 Relative Error Bounds for Arithmetic Operations on Inexact Data

$$\text{Rel}\,(x^* + y^*) = \max\{(\text{Rel}\,(x^*), \text{Rel}\,(y^*)\}, \text{ for } x, y \text{ having same sign}$$
$$\text{Rel}\,(x^* - y^*) = \frac{\delta(x^*) + \delta(y^*)}{|x - y|}, \text{ any } x, y$$
$$\text{Rel}\,(x^* \cdot y^*) \approx \text{Rel}\,(x^*) + \text{Rel}\,(y^*), \text{ any } x, y$$
$$\text{Rel}\,(x^*/y^*) \approx \text{Rel}\,(x^*) + \text{Rel}\,(y^*), \text{ any } x, \text{ and any } y \neq 0$$

In Tables 1.12 and 1.13 we collect the absolute and relative error bounds induced by performing arithmetic on approximating data.

EXAMPLE 1.15

We derive the entry in Table 1.13 for Rel $(x^* + y^*)$, with x and y having the same sign:

$$\text{Rel}\,(x^* + y^*) = \frac{\delta(x^*) + \delta(y^*)}{|x + y|}.$$

Suppose, by exchanging labels if necessary, that

$$\frac{\delta(x^*)}{|x|} = \text{Rel}\,(x^*) \geq \text{Rel}\,(y^*) = \frac{\delta(y^*)}{|y|}.$$

Then $\delta(x^*)|y| \geq \delta(y^*)|x|$ and

$$\delta(x^*)|x + y| = \delta(x^*)(|x| + |y|) \geq |x|(\delta(x^*) + \delta(y^*)),$$

which gives the relation

$$\frac{\delta(x^*)}{|x|} \geq \frac{\delta(x^*) + \delta(y^*)}{|x + y|}$$

or

$$\text{Rel}\,(x^*) \geq \text{Rel}\,(x^* + y^*).$$

■

*1.6.3 Propagation of Errors Through a Computation

In our discussion of bounds for propagated error, we assumed that all operations and computation of function values are done exactly. But in fact, at each stage of a computation, errors are introduced through truncation and rounding. Thus typically, each stage of a lengthy program operates on faulty data and is itself a source of new errors.

Figure 1.10 illustrates the distinction between an ideal sequence of computations and its computer-realizable counterpart. In this figure the exact input to the first stage is represented by x_1. After the first stage of computation, ideally the output should be $y_1 = F_1(x_1)$. This output serves as input to the next computational stage or module. In the realizable computation, the input x_1^* is an approximation of the desired input x_1. Whereas ideally, we wished the output of the first stage to be $F_1(x_1)$, in the actual computation, it will be $y_1^* = F_1(x_1^*)$. Added to this faulty output is an undesired error e_1, which can represent effects of inexact calculation of the operator F_1 as well as roundoff error accumulated in the first stage of computation. The sum of y_1^* and e_1 serves as input to the next computational stage or module, and this process continues, with error propagating and accumulating at each segment of the computation.

Armed with an understanding from the preceding section about how error propagates through an arithmetical operation, and assuming that the relative error introduced by roundoff after each such operation F_j is e_j, we are in a position to analyze the overall accumulated error in any program-directed computation. Such analysis is seldom undertaken for lengthy programs because, while conceptually straightforward, it involves excruciating detail due to all the many operations and branches in a typical computation. Moreover, such "worst-case" analysis tends to lead to pessimistic bounds if carried out over many stages. However, for relatively small computations that occur repeatedly within a larger program, error bounding is frequently useful.

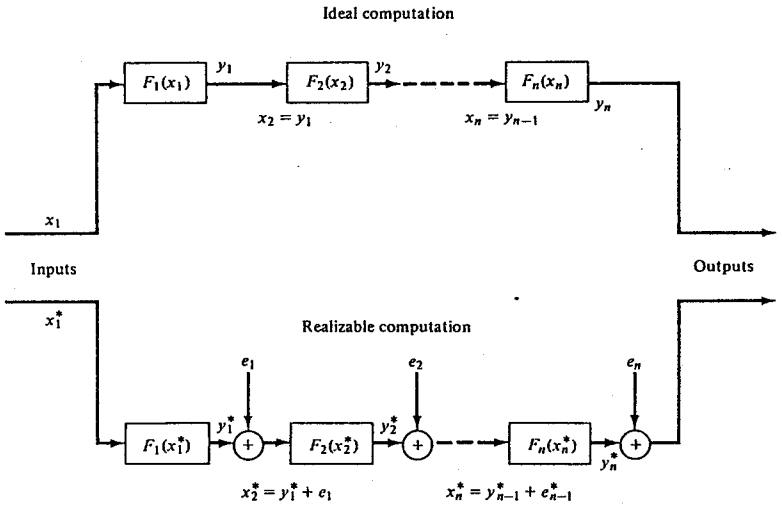


FIGURE 1.10 Comparison of Ideal and Realizable Computations

EXAMPLE 1.16

As illustration of the principle involved, let us bound the relative error in computing $xy + z$ using a machine with epsilon value ε. Further, presume the floating-point numbers x^*, y^*, and z^*, which approximate x, y, and z, are positive. Then, according to Table 1.13,

$$\text{Rel}\,(x^*y^*) \approx \text{Rel}\,(x^*) + \text{Rel}\,(y^*). \tag{1.14}$$

Now the number $(x^*y^*)^*$, stored as the computed x^*y^*, is subject to roundoff error with relative error bound ε. This error is added directly to the propagated error. We then have that

$$\text{Rel}\,((x^*y^*)^*) = \frac{\delta(x^*y^*) + \varepsilon|xy|}{|xy|} = \text{Rel}\,(x^*y^*) + \varepsilon$$
$$\approx \text{Rel}\,(x^*) + \text{Rel}\,(y^*) + \varepsilon.$$

Now again by Table 1.13,

$$\text{Rel}\,((x^*y^*)^* + z^*) = \max\{\text{Rel}\,((x^*y^*)^*), \text{Rel}\,(z^*)\},$$

and after approximating $(x^*y^*) + z^*$ by the floating-point number $((x^*y^*)^* + z^*)^*$ and again accounting for roundoff, we obtain the relative error bound of the output approximation $((x^*y^*)^* + z^*)^*$ for $xy + z$, as

$$\text{Rel}\,(((x^*y^*)^* + z^*)^*) = \max\{\text{Rel}\,(x^*) + \text{Rel}\,(y^*) + \varepsilon, \text{Rel}\,(z^*)\} + \varepsilon. \quad \blacksquare$$

EXAMPLE 1.17

Suppose for the sake of simplicity, that a binary computer allows only $s = 4$ binary coefficients to represent the mantissa of a number, and that in normal form representation, $x = (0.1101)_2 \times 2^3$ and $y = (0.1100)_2 \times 2^{-1}$. Then

$$x + y = (110.1)_2 + (0.011)_2 = (110.111)_2.$$

If the surplus mantissa coefficients are simply ignored (or "chopped"), this sum is stored as $(0.1101)_2 \times 2^3$. Note that y was not large enough to change the floating-point representation after addition. $\blacksquare$

Although the error resulting from a single arithmetic operation is usually insignificant in itself, over the course of a long computation, the cumulative effect of many such errors can make itself felt.

EXAMPLE 1.18

We perform the experiment of forming a sum by adding the number $A = 10^{-6}$ one million times. The program and outcome on a VAX are given in Tables 1.14 and 1.15. Note that the sum is correct only to two significant decimals, whereas from Table 1.8, or direct computation of machine epsilon, we are assured that the number A itself was stored with an accuracy of seven or eight significant decimals.

TABLE 1.14 Program for Accumulating a Sum

```
C     PROGRAM ADDALOT
C
C     ************************************************
C     THIS PROGRAM DEMONSTRATES THE CUMULATIVE EFFECT OF
C     ADDING NUMBERS OF DIFFERENT MAGNITUDES MANY TIMES
C     OUTPUT: SUM=SUM OF THE VARIABLE A 1000000 TIMES
C     ************************************************
C
      A=1.E-6
      SUM=0.
      DO 1 I=1,1000000
         SUM=SUM+A
    1 CONTINUE
      WRITE(10,2) SUM
    2 FORMAT(33X,'SUM=',F10.7)
      STOP
      END
```

TABLE 1.15 Output for Accumulating a Sum

```
SUM = 1.0090389
```

The major cause of this loss of precision is repeated loss of significance in adding a relatively small number to the much larger sum, as the number of terms increases. This is the phenomenon illustrated in Example 1.10. To some extent this loss of significance can be sidestepped by accumulating the sums in a great many smaller partial sums, and then adding these partial sums together. The program shown in Table 1.16 follows this idea, accumulating the sums of the small number A in a thousand different partial sums S(J), which are later added to one another to achieve an answer (Table 1.17) with five significant decimal places.

TABLE 1.16 Program for Accumulating a Sum

```
C     PROGRAM ADD
C
C     ************************************************
C     THIS PROGRAM ILLUSTRATES HOW MORE ACCURACY IS OBTAINED
C     WHEN CARE IS TAKEN TO ADD NUMBERS OF SIMILAR MAGNITUDE
C     OUTPUT: SUM=SUM OF THE VARIABLE A COMPUTED AS THE SUM
C                OF 1000 SUMS OF 1000 A'S
C     ************************************************
C
      DIMENSION S(1000)
      A=1.E-6
      SUM=0.
      DO 1 J=1,1000
         S(J)=0.
         DO 1 I=1,1000
            S(J)=S(J)+A
    1    CONTINUE
      DO 2 J=1,1000
         SUM=SUM+S(J)
    2 CONTINUE
      WRITE(10,3) SUM
    3 FORMAT(33X,'SUM=',F10.7)
      STOP
      END
```

TABLE 1.17 Output for Accumulating a Sum

SUM = 0.9999985

PPLEMENTARY NOTES AND DISCUSSIONS

During the period from 1940 to 1955, ideas and methodology for computer number representation underwent lively and innovative development, as indeed this was the period of the emergence of the electronic digital computer. Prior to this period, computers were mechanical and, with the exception of experimental devices, strictly decimal. The arithmetic mode was equivalent to integer representation. Metropolis, et al. (1980) give ample documentation of the evolution of floating-point representations. The advantage of this representation was clearly foreseen by the pioneers of modern computers, but in the first machines there was a tendency to force the programmer to encode the number representation. In early computer designs, essentially all user conveniences were sacrificed for the sake of computational speed. Since the advent of FORTRAN in the late 1950s, automatic floating-point representation seems to have become firmly established as the medium for numerical calculations. Many implementations of BASIC, for example, allow only such representation. The evolution will undoubtedly continue. For instance, some experimental computers have variable-length floating-point mantissas.

For details of number representation and the actual operations involved in computer arithmetic, the assembly language manual of the computer in question should be consulted. Knuth (1969, Chap. 4) provides many more details on this issue than we have given and illustrates the principles of computer number representation and arithmetic with a hypothetical assembly language.

We have described situations such as subtractive cancellation and smearing in which numerical error leads to large or intolerable "loss of significance." As illustrated in our examples, in many such situations these difficulties can be circumvented through more careful computational and analytical thinking. Stegun and Abramowitz (1956), Henrici (1982), and Rice (1983) provide further examples and insights along these lines. Our plan is to delay until later chapters (especially Chapter 4 on linear equations) a discussion of "ill-conditioned" problems in which slight roundoff or measurement error on input parameters can lead to devastating error in the computed output.

At the close of this chapter we sketched principles involved in finding the absolute bound of propagated error. For full-scale computations, such analysis, in addition to being excruciatingly tedious, tends to result in pessimistic answers. Early in the computer age, scientists occasionally concluded from such calculations that a given problem could not possibly be solved on present-day computers, only to find that somebody else had indeed found and verified the solution. At present, absolute bound analysis finds use in small-scale tasks such as bounding the effect of computer word roundoff in a library function routine. For large-scale computations, the tendency is to push ahead with the calculations but provide accuracy checks at key points along the way. For example, it may be possible to substitute a proposed solution back into an equation to be solved and to check

that the equation is indeed satisfied. Alternatively, a method may be checked on prototypical problems having known closed-form solutions.

At the outset of this chapter we described the goals of numerical analysis as including construction of computer-realizable approximations for unrealizable mathematical operations, and derivation of error bounds for these constructs. A further theme that will come to the fore in subsequent chapters is that methods should be efficient. If one method is essentially as accurate as another but requires only half the computation time, it is clearly the method of choice.

PROBLEMS

Section 1.3

1. Convert the following binary numbers to decimal.
   (a) $(111001)_2$.
   (b) $(110.1101)_2$.

2. Suppose, for our convenience, that we have a computer with decimal (instead of binary) words, and that the mantissa length s is only three digits, but in this problem the exponent length e is of no concern. Since $s = 3$, and assuming that the computer rounds a given number to the nearest computer number, it will store 416.8, for example, as 417., and store 0.04164 as 0.0416. Assuming that the computer does arithmetical operations perfectly and the only error is that incurred in storing the resultant as a computer number find the stored sum of the numbers in parts (a) to (d).
   (a) 30. + 4.12.
   (b) 300. + 1.312.
   (c) 922. + 106.73.
   (d) 12345. + (−12344.).
   (e) Suppose that each of the numbers in parts (a) to (d) must first be stored and then the stored representations added. What results form these operations?

3. Find the decimal value of the following floating-point representation by the DEC 10 computer.

0	1	2	3	4	5	6	7	8	9	10	11	12		33	34	35
0	1	0	0	1	1	1	1	1	1	1	0	0	· · ·	0	0	0

4. How many distinct machine words of length t are there? (HINT: Examine the situation for $t = 1, 2, \ldots$ and generalize. Establish your result rigorously by finite induction.) Note that this provides an upper bound to the number of distinct single-precision integers or floating-point numbers.

5. Write a subroutine which, when presented with positive integers N and K and integer array A = (A(1),. . .,A(K + 1)), returns the integer

$$M = A(1) + A(2)N + \cdots + A(K+1)N^K.$$

Check your subroutine by verifying that $(35411)_6 = (5119)_{10}$.

6. Write a subroutine that does the converse of Problem 5. That is, this subroutine, when presented with the input variables M and N, should successively compute integers A(1), A(2), . . ., A(K+1), satisfying the equation in Problem 5 as well as the condition that $0 \le A(j) < N$, $1 \le j \le K+1$. [HINT: Let "Rem" denote "remainder." Then A(1) = Rem(M/N). Let I = ((M−A(1))/N). Then A(2) = Rem(I/N). Figure out why these statements are so, and then continue the algorithm in a recursive loop.] Check your method by finding the representation of $(5119)_{10}$ in the base $N = 6$. Note that in FORTRAN, Rem (I/N) = MOD(I, N).

Section 1.4

7. Write a program that subtracts 50,000,000 from 50,000,001 in both integer and floating-point arithmetic. Explain the output.

8. Write a program and find a set of N numbers X(1), X(2), . . ., X(N) illustrating that the computed sum

$$S = X(1) + X(2) + \cdots + X(N)$$

can depend on the order in which the X(j)'s are added. Explain this effect.

9. Use Subroutine EPS (Table 1.9) to find machine epsilon on your computer. Check to see that if β is the smallest positive floating-point number such that

$$A + \beta \neq A,$$

then β is "not far" from $|A|\varepsilon$, ε being your computer machine epsilon. Take $A = 10$, $\frac{1}{30}$, and 104.

10. Write a program to approximate sin (x) by means of the expansion

$$\sin(x) = x - \frac{x^3}{3!} + \cdots + (-1)^j \frac{x^{2j+1}}{(2j+1)!} + \cdots.$$

Compute sin (10) and compare it with the library function estimate. Devise a "wise" procedure for approximating sin (10). Try $j = 1, 5, 10$, and 15.

11. The quadratic formula

$$x_1, x_2 = (-b \pm \sqrt{b^2 - 4ac})/(2a)$$

for the roots of $p(x) = ax^2 + bx + c$ is prone to subtractive cancellation error when ($|b| >> |a|$ and $|c|$). Illustrate this fact by showing that $p(x_j)$ is not very close to 0. If you have mathematical background, a cure might occur to you.

12. One must distinguish between machine epsilon EPS and $x_{\min}$, the smallest positive floating-point number, which is far smaller than EPS. Find a theoretical number value of this latter number in terms of the word length t and the mantissa length s of your computer. Experimentally approximate the latter number on your computer. A number having magnitude less than this quantity cannot reliably be distinguished from zero by the computer.

Section 1.6

13. Presume that numbers $x^* = 11.33$ and $y^* = 2.15$ are obtained by rounding x and y, respectively, to the number of significant digits shown. That is, $\delta(x^*) = \delta(y^*) = 0.005$. Give error bounds for the following quantities.
(a) $x + xy$.
(b) $\sqrt{xy}$.
(c) $\sin(2x - y) + \cos(2x - y)$.
(d) $\frac{x}{y} - \frac{y}{x}$.

14. With x and y as in Problem 13, estimate the relative errors for each of the following.
(a) $x + xy$.
(b) $\sqrt{xy}$.
(c) $\sin(2x - y) + \cos(2x - y)$.
(d) $\frac{x}{y} - \frac{y}{x}$.

15. In theory, after performing the recursion SUM = SUM + A, N times, we should have SUM = $N * A$. However, because of roundoff error, the branching statement "IF (SUM .EQ. $N * A$)" may fail to branch. Give a computational illustration of this phenomenon.

16. Write a program to compute the sum

$$x = 1 - \frac{1}{a} + \frac{1}{a^2} - \frac{1}{a^3} + \cdots + \frac{1}{a^{10,000}},$$

where $a = 1.001$. Then print the sum and recalculate the sum backward from the right and print the result. Compare the two values, stating which you believe to be more accurate, and why.

17. A real number x is approximated by 135.12 and we are told that the relative error bound is 0.01. What can be said about the true value of x?

18. The developments in this section have a bearing on the illustration of subtractive cancellation in Example 1.9. Show how to use the mean-value theorem to estimate $f(a) - f(b)$, for a and b given numbers. Use this technique to estimate $f(1985) - f(1984)$, where $f(x) = \text{SQRT}(x)$. Apply your formula to the numbers given in Example 1.9.

19. The area of a right triangle is completely determined by the length x of a side and the angle θ which that given side makes with the hypotenuse. Let $x^* = 5$ and the angle $\theta = 45°$. Suppose that $\delta(x^*) = 0.1$ and $\delta(\theta) = 5°$. Estimate bounds for the absolute and relative errors.

2.7 COMPLEX ROOTS AND MULLER'S METHOD

The methods discussed up to this point allow us to find an isolated zero of a function once an approximation to that zero is known. These methods are not very satisfactory when all the zeros of a function are required or when good initial approximations are not available. For polynomial functions there are methods which yield an approximation to all the zeros simultaneously, after which the iterative methods of this chapter can be applied to obtain more accurate solutions. Among such methods may be mentioned the quotient-difference algorithm [2] and the method of Graeffe [5].

A method of recent vintage, expounded by Muller [6], has been used on computers with remarkable success. This method may be used to find

any prescribed number of zeros, real or complex, of an arbitrary function. The method is iterative, converges almost quadratically in the vicinity of a root, does not require the evaluation of the derivative of the function, and obtains both real and complex roots even when these roots are not simple.

Moreover, the method is global in the sense that the user need not supply an initial approximation. In this section we describe briefly how the method is derived, omitting any discussion of convergence, and we discuss its use in finding both real and complex roots. We will especially emphasize the problem of finding complex zeros of polynomials with real coefficients since this problem is of great concern in many branches of engineering.

Muller's method is an extension of the secant method. To recall, in the secant method we determine, from the approximations x_i, x_{i-1} to a root of $f(x) = 0$, the next approximation x_{i+1} as the zero of the linear polynomial $p(x)$ which goes through the two points $\{x_i, f(x_i)\}$ and $\{x_{i-1}, f(x_{i-1})\}$. In Muller's method, the next approximation, x_{i+1}, is found as a zero of the *parabola* which goes through the three points $\{x_i, f(x_i)\}$, $\{x_{i-1}, f(x_{i-1})\}$, and $\{x_{i-2}, f(x_{i-2})\}$.

Let x_i, x_{i-1}, x_{i-2} be three distinct approximations to a root of $f(x) = 0$. We use the abbreviations

$$f_i = f(x_i) \qquad f_{i-1} = f(x_{i-1}) \qquad f_{i-2} = f(x_{i-2})$$

In Chap. 4, we show that, with

$$f[x_i, x_{i-1}] = \frac{f_i - f_{i-1}}{x_i - x_{i-1}}$$

$$f[x_i, x_{i-1}, x_{i-2}] = \frac{f[x_i, x_{i-1}] - f[x_{i-1}, x_{i-2}]}{x_i - x_{i-2}} \tag{2.45}$$

the function

$$p(x) = f_i + f[x_i, x_{i-1}](x - x_i) + f[x_i, x_{i-1}, x_{i-2}](x - x_i)(x - x_{i-1})$$

is the unique parabola which goes through the three points $\{x_i, f_i\}$, $\{x_{i-1}, f_{i-1}\}$, and $\{x_{i-2}, f_{i-2}\}$. In the more customary way of writing down polynomials, we get that

$$p(x) = a_0 + a_1 x + a_2 x^2$$

with

$$\begin{aligned} a_2 &= f[x_i, x_{i-1}, x_{i-2}] \\ a_1 &= f[x_i, x_{i-1}] - (x_i + x_{i-1}) a_2 \\ a_0 &= f_i - x_i(f[x_i, x_{i-1}] - x_{i-1} a_2) \end{aligned} \tag{2.46}$$

2.7 COMPLEX ROOTS AND MULLER'S METHOD

The methods discussed up to this point allow us to find an isolated zero of a function once an approximation to that zero is known. These methods are not very satisfactory when all the zeros of a function are required or when good initial approximations are not available. For polynomial functions there are methods which yield an approximation to all the zeros simultaneously, after which the iterative methods of this chapter can be applied to obtain more accurate solutions. Among such methods may be mentioned the quotient-difference algorithm [2] and the method of Graeffe [5].

A method of recent vintage, expounded by Muller [6], has been used on computers with remarkable success. This method may be used to find

any prescribed number of zeros, real or complex, of an arbitrary function. The method is iterative, converges almost quadratically in the vicinity of a root, does not require the evaluation of the derivative of the function, and obtains both real and complex roots even when these roots are not simple.

Moreover, the method is global in the sense that the user need not supply an initial approximation. In this section we describe briefly how the method is derived, omitting any discussion of convergence, and we discuss its use in finding both real and complex roots. We will especially emphasize the problem of finding complex zeros of polynomials with real coefficients since this problem is of great concern in many branches of engineering.

Muller's method is an extension of the secant method. To recall, in the secant method we determine, from the approximations x_i, x_{i-1} to a root of $f(x) = 0$, the next approximation x_{i+1} as the zero of the linear polynomial $p(x)$ which goes through the two points $\{x_i, f(x_i)\}$ and $\{x_{i-1}, f(x_{i-1})\}$. In Muller's method, the next approximation, x_{i+1}, is found as a zero of the *parabola* which goes through the three points $\{x_i, f(x_i)\}$, $\{x_{i-1}, f(x_{i-1})\}$, and $\{x_{i-2}, f(x_{i-2})\}$.

Let x_i, x_{i-1}, x_{i-2} be three distinct approximations to a root of $f(x) = 0$. We use the abbreviations

$$f_i = f(x_i) \qquad f_{i-1} = f(x_{i-1}) \qquad f_{i-2} = f(x_{i-2})$$

In Chap. 4, we show that, with

$$f[x_i, x_{i-1}] = \frac{f_i - f_{i-1}}{x_i - x_{i-1}}$$

$$f[x_i, x_{i-1}, x_{i-2}] = \frac{f[x_i, x_{i-1}] - f[x_{i-1}, x_{i-2}]}{x_i - x_{i-2}} \tag{2.45}$$

the function

$$p(x) = f_i + f[x_i, x_{i-1}](x - x_i) + f[x_i, x_{i-1}, x_{i-2}](x - x_i)(x - x_{i-1})$$

is the unique parabola which goes through the three points $\{x_i, f_i\}$, $\{x_{i-1}, f_{i-1}\}$, and $\{x_{i-2}, f_{i-2}\}$. In the more customary way of writing down polynomials, we get that

$$p(x) = a_0 + a_1 x + a_2 x^2$$

with

$$\begin{aligned} a_2 &= f[x_i, x_{i-1}, x_{i-2}] \\ a_1 &= f[x_i, x_{i-1}] - (x_i + x_{i-1}) a_2 \\ a_0 &= f_i - x_i (f[x_i, x_{i-1}] - x_{i-1} a_2) \end{aligned} \tag{2.46}$$

2.7 COMPLEX ROOTS AND MULLER'S METHOD

The methods discussed up to this point allow us to find an isolated zero of a function once an approximation to that zero is known. These methods are not very satisfactory when all the zeros of a function are required or when good initial approximations are not available. For polynomial functions there are methods which yield an approximation to all the zeros simultaneously, after which the iterative methods of this chapter can be applied to obtain more accurate solutions. Among such methods may be mentioned the quotient-difference algorithm [2] and the method of Graeffe [5].

A method of recent vintage, expounded by Muller [6], has been used on computers with remarkable success. This method may be used to find

any prescribed number of zeros, real or complex, of an arbitrary function. The method is iterative, converges almost quadratically in the vicinity of a root, does not require the evaluation of the derivative of the function, and obtains both real and complex roots even when these roots are not simple.

Moreover, the method is global in the sense that the user need not supply an initial approximation. In this section we describe briefly how the method is derived, omitting any discussion of convergence, and we discuss its use in finding both real and complex roots. We will especially emphasize the problem of finding complex zeros of polynomials with real coefficients since this problem is of great concern in many branches of engineering.

Muller's method is an extension of the secant method. To recall, in the secant method we determine, from the approximations x_i, x_{i-1} to a root of $f(x) = 0$, the next approximation x_{i+1} as the zero of the linear polynomial $p(x)$ which goes through the two points $\{x_i, f(x_i)\}$ and $\{x_{i-1}, f(x_{i-1})\}$. In Muller's method, the next approximation, x_{i+1}, is found as a zero of the *parabola* which goes through the three points $\{x_i, f(x_i)\}$, $\{x_{i-1}, f(x_{i-1})\}$, and $\{x_{i-2}, f(x_{i-2})\}$.

Let x_i, x_{i-1}, x_{i-2} be three distinct approximations to a root of $f(x) = 0$. We use the abbreviations

$$f_i = f(x_i) \qquad f_{i-1} = f(x_{i-1}) \qquad f_{i-2} = f(x_{i-2})$$

In Chap. 4, we show that, with

$$f[x_i, x_{i-1}] = \frac{f_i - f_{i-1}}{x_i - x_{i-1}}$$

$$f[x_i, x_{i-1}, x_{i-2}] = \frac{f[x_i, x_{i-1}] - f[x_{i-1}, x_{i-2}]}{x_i - x_{i-2}} \tag{2.45}$$

the function

$$p(x) = f_i + f[x_i, x_{i-1}](x - x_i) + f[x_i, x_{i-1}, x_{i-2}](x - x_i)(x - x_{i-1})$$

is the unique parabola which goes through the three points $\{x_i, f_i\}$, $\{x_{i-1}, f_{i-1}\}$, and $\{x_{i-2}, f_{i-2}\}$. In the more customary way of writing down polynomials, we get that

$$p(x) = a_0 + a_1 x + a_2 x^2$$

with

$$\begin{aligned} a_2 &= f[x_i, x_{i-1}, x_{i-2}] \\ a_1 &= f[x_i, x_{i-1}] - (x_i + x_{i-1}) a_2 \\ a_0 &= f_i - x_i(f[x_i, x_{i-1}] - x_{i-1} a_2) \end{aligned} \tag{2.46}$$

2.7 COMPLEX ROOTS AND MULLER'S METHOD

The methods discussed up to this point allow us to find an isolated zero of a function once an approximation to that zero is known. These methods are not very satisfactory when all the zeros of a function are required or when good initial approximations are not available. For polynomial functions there are methods which yield an approximation to all the zeros simultaneously, after which the iterative methods of this chapter can be applied to obtain more accurate solutions. Among such methods may be mentioned the quotient-difference algorithm [2] and the method of Graeffe [5].

A method of recent vintage, expounded by Muller [6], has been used on computers with remarkable success. This method may be used to find

any prescribed number of zeros, real or complex, of an arbitrary function. The method is iterative, converges almost quadratically in the vicinity of a root, does not require the evaluation of the derivative of the function, and obtains both real and complex roots even when these roots are not simple.

Moreover, the method is global in the sense that the user need not supply an initial approximation. In this section we describe briefly how the method is derived, omitting any discussion of convergence, and we discuss its use in finding both real and complex roots. We will especially emphasize the problem of finding complex zeros of polynomials with real coefficients since this problem is of great concern in many branches of engineering.

Muller's method is an extension of the secant method. To recall, in the secant method we determine, from the approximations x_i, x_{i-1} to a root of $f(x) = 0$, the next approximation x_{i+1} as the zero of the linear polynomial $p(x)$ which goes through the two points $\{x_i, f(x_i)\}$ and $\{x_{i-1}, f(x_{i-1})\}$. In Muller's method, the next approximation, x_{i+1}, is found as a zero of the *parabola* which goes through the three points $\{x_i, f(x_i)\}$, $\{x_{i-1}, f(x_{i-1})\}$, and $\{x_{i-2}, f(x_{i-2})\}$.

Let x_i, x_{i-1}, x_{i-2} be three distinct approximations to a root of $f(x) = 0$. We use the abbreviations

$$f_i = f(x_i) \qquad f_{i-1} = f(x_{i-1}) \qquad f_{i-2} = f(x_{i-2})$$

In Chap. 4, we show that, with

$$f[x_i, x_{i-1}] = \frac{f_i - f_{i-1}}{x_i - x_{i-1}}$$

$$f[x_i, x_{i-1}, x_{i-2}] = \frac{f[x_i, x_{i-1}] - f[x_{i-1}, x_{i-2}]}{x_i - x_{i-2}} \tag{2.45}$$

the function

$$p(x) = f_i + f[x_i, x_{i-1}](x - x_i) + f[x_i, x_{i-1}, x_{i-2}](x - x_i)(x - x_{i-1})$$

is the unique parabola which goes through the three points $\{x_i, f_i\}$, $\{x_{i-1}, f_{i-1}\}$, and $\{x_{i-2}, f_{i-2}\}$. In the more customary way of writing down polynomials, we get that

$$p(x) = a_0 + a_1 x + a_2 x^2$$

with

$$\begin{aligned} a_2 &= f[x_i, x_{i-1}, x_{i-2}] \\ a_1 &= f[x_i, x_{i-1}] - (x_i + x_{i-1}) a_2 \\ a_0 &= f_i - x_i (f[x_i, x_{i-1}] - x_{i-1} a_2) \end{aligned} \tag{2.46}$$

*3.7 COMPLEX ROOTS AND MÜLLER'S METHOD

The methods discussed up to this point allow us to find an isolated zero of a function once an approximation to that zero is known. These methods are not very satisfactory when all the zeros of a function are required or when good initial approximations are not available. For polynomial functions there are methods which yield an approximation to all the zeros simultaneously, after which the iterative methods of this chapter can be applied to obtain more accurate solutions. Among such methods may be mentioned the quotient-difference algorithm [2] and the method of Graeffe [5].

A method of recent vintage, expounded by Müller [6], has been used on computers with remarkable success. This method may be used to find any prescribed number of zeros, real or complex, of an arbitrary function. The method is iterative, converges almost quadratically in the vicinity of a root, does not require the evaluation of the derivative of the function, and obtains both real and complex roots even when these roots are not simple.

Moreover, the method is global in the sense that the user need not supply an initial approximation. In this section we describe briefly how the method is derived, omitting any discussion of convergence, and we discuss its use in finding both real and complex roots. We will especially emphasize the problem of finding complex zeros of polynomials with real coefficients since this problem is of great concern in many branches of engineering.

Müller's method is an extension of the secant method. To recall, in the secant method we determine, from the approximations x_i, x_{i-1} to a root of $f(x) = 0$, the next approximation x_{i+1} as the zero of the linear polynomial $p(x)$ which goes through the two points $\{x_i, f(x_i)\}$ and $\{x_{i-1} f(x_{i-1})\}$. In Müller's method, the next approximation, x_{i+1}, is found as a zero of the *parabola* which goes through the three points $\{x_i, f(x_i)\}$, $\{x_{i-1}, f(x_{i-1})\}$, and $\{x_{i-2}, f(x_{i-2})\}$.

As shown in Chap. 2, the function

$$p(x) = f(x_i) + f[x_i, x_{i-1}](x - x_i) + f[x_i, x_{i-1}, x_{i-2}](x - x_i)(x - x_{i-1})$$

is the unique parabola which agrees with the function $f(x)$ at the three points x_i, x_{i-1}, x_{i-2}. Since

$$(x - x_i)(x - x_{i-1}) = (x - x_i)^2 + (x - x_i)(x_i - x_{i-1})$$

we can also write $p(x)$ in the form

$$p(x) = f(x_i) + (x - x_i)c_i + f[x_i, x_{i-1}, x_{i-2}](x - x_i)^2 \qquad (3.51)$$

with

$$c_i = f[x_i, x_{i-1}] + f[x_i, x_{i-1}, x_{i-2}](x_i - x_{i-1})$$

Thus any zero α of the parabola $p(x)$ satisfies

$$\alpha - x_i = \frac{-2f(x_i)}{c_i \pm \{c_i^2 - 4f(x_i)f[x_i, x_{i-1}, x_{i-2}]\}^{1/2}} \qquad (3.52)$$

according to one version of the standard quadratic formula [see (1.20)]. If we choose the sign in (3.52) so that the denominator will be as large in magnitude as possible, and if we then label the right-hand side of (3.52) as h_{i+1}, then the next approximation to a zero of $f(x)$ is taken to be

$$x_{i+1} = x_i + h_{i+1}$$

The process is then repeated using x_{i-1}, x_i, x_{i+1} as the three basic approximations. If the zeros obtained from (3.52) are real, the situation is pictured graphically in Fig. 3.8. Note, however, that even if the zero being sought is real, we may encounter complex approximations because the solutions given by (3.52) may be complex. However, in such cases the complex component will normally be so small in magnitude that it can be neglected. In fact, in the subroutine given below, any complex components encountered in seeking a real zero can be suppressed.

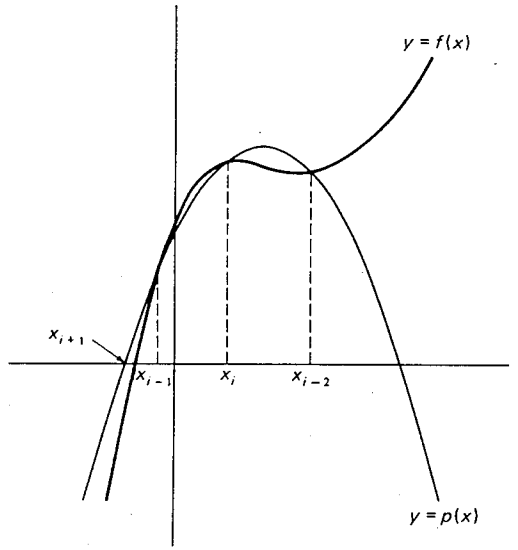


Figure 3.8

The sequence of steps required in Müller's method is formalized in Algorithm 3.10.

Algorithm 3.10: Müller's method

1. Let x_0, x_1, x_2 be three approximations to a zero ξ of $f(x)$. Compute $f(x_0)$, $f(x_1)$, $f(x_2)$.

2. Compute

$$h_2 := x_2 - x_1, h_1 := x_1 - x_0$$
$$f[x_2, x_1] = (f(x_2) - f(x_1))/h_2$$
$$f[x_1, x_0] = (f(x_1) - f(x_0))/h_1$$

3. Set $i = 2$
4. Compute

$$f[x_i, x_{i-1}, x_{i-2}] = (f[x_i, x_{i-1}] - f[x_{i-1}, x_{i-2}])/(h_i + h_{i-1})$$
$$c_i := f[x_i, x_{i-1}] + h_i f[x_i, x_{i-1}, x_{i-2}]$$

5. Compute

$$h_{i+1} := -2f(x_i)/\left(c_i \pm \sqrt{c_i^2 - 4f(x_i)f[x_i, x_{i-1}, x_{i-2}]}\right)$$

choosing the sign so that the denominator is largest in magnitude.

6. Set $x_{i+1} = x_i + h_{i+1}$
7. Compute

$$f(x_{i+1}) \quad \text{and} \quad f[x_{i+1}, x_i] = (f(x_{i+1}) - f(x_i))/h_{i+1}$$

8. Set $i = i + 1$ and repeat steps 4–7 until either of the following criteria is satisfied for prescribed ε_1, ε_2:
   (*a*) $|x_i - x_{i-1}| < \varepsilon_1 |x_i|$
   (*b*) $|f(x_i)| < \varepsilon_2$
   or until the maximum number of iterations is exceeded.

A complete subroutine based on this algorithm is given below. The calling parameters for the subroutine are explained in the comment cards. ZEROS(I) is a one-dimensional array containing initial estimates of the desired zeros. The subroutine automatically computes two additional approximations to ZEROS(I) as ZEROS(I) + .5 and ZEROS(I) − .5 and then proceeds with the Müller algorithm.

```
      SUBROUTINE MULLER ( FN, FNREAL, ZEROS, N, NPREV, MAXIT, EP1, EP2 )
C  DETERMINES UP TO  N  ZEROS OF THE FUNCTION SPECIFIED BY  FN , USING
C  QUADRATIC INTERPOLATION, I.E., MUELLER'S METHOD .
      EXTERNAL FN
      LOGICAL FNREAL
      INTEGER MAXIT,N,NPREV,   KOUNT
      REAL EP1,EP2,   EPS1,EPS2
      COMPLEX ZEROS(N),   C,DEN,DIVDF1,DIVDF2,DVDF1P,FZR,FZRDFL
     *       ,FZRPRV,H,ZERO,SQR
C******  I N P U T  ******
C  FN  NAME OF A SUBROUTINE, OF THE FORM  FN(Z, FZ)  WHICH, FOR GIVEN
C       Z , RETURNS  F(Z) . MUST APPEAR IN AN  E X T E R N A L  STATE-
C      MENT IN THE CALLING PROGRAM .
C  FNREAL  A LOGICAL VARIABLE. IF .TRUE., ALL APPROXIMATIONS ARE TAKEN
C      TO BE REAL, ALLOWING THIS ROUTINE TO BE USED EVEN IF  F(Z)  IS
C      ONLY DEFINED FOR REAL  Z .
C  ZEROS(1),...,ZEROS(NPREV)  CONTAINS PREVIOUSLY FOUND ZEROS (IF
```



```
C      NPREV .GT. 0).
C  ZEROS(NPREV+1),...,ZEROS(N)  CONTAINS FIRST GUESS FOR THE ZEROS TO BE
C      FOUND. (IF YOU KNOW NOTHING, 0  IS AS GOOD A GUESS AS ANY.)
C  MAXIT  MAXIMUM NUMBER OF FUNCTION EVALUATIONS ALLOWED PER ZERO.
C  EP1  ITERATION IS STOPPED IF ABS(H) .LT. EP1*ABS(ZR), WITH
C       H = LATEST CHANGE IN ZERO ESTIMATE  ZERO .
C  EP2  ALTHOUGH THE  EP1  CRITERION IS NOT MET, ITERATION IS STOPPED IF
C       ABS(F(ZERO)) .LT. EP2 .
C  N  TOTAL NUMBER OF ZEROS TO BE FOUND .
C  NPREV  NUMBER OF ZEROS FOUND PREVIOUSLY .
C******  O U T P U T  ******
C  ZEROS(NPREV+1), ..., ZEROS(N)  APPROXIMATIONS TO ZEROS .
C
C               INITIALIZATION
      EPS1 = MAX(EP1, 1.E-12)
      EPS2 = MAX(EP2, 1.E-20)
C
      DO 100 I=NPREV+1,N
         KOUNT = 0
C              COMPUTE FIRST THREE ESTIMATES FOR ZERO AS
C                 ZEROS(I)+5., ZEROS(I)-.5, ZEROS(I)
    1    ZERO = ZEROS(I)
         H = .5
         CALL DFLATE(FN, ZERO+.5, I, KOUNT, FZR, DVDF1P, ZEROS, *1)
         CALL DFLATE(FN, ZERO-.5, I, KOUNT, FZR, FZRPRV, ZEROS, *1)
         HPREV = -1.
         DVDF1P = (FZRPRV - DVDF1P)/HPREV
         CALL DFLATE(FN, ZERO, I, KOUNT, FZR, FZRDFL, ZEROS, *1)
C        DO WHILE  KOUNT.LE.MAXIT OR  H  IS RELATIVELY BIG
C            OR  FZR = F(ZERO)  IS NOT SMALL
C            OR  FZRDFL = FDEFLATED(ZERO)  IS NOT SMALL OR NOT MUCH
C             BIGGER THAN ITS PREVIOUS VALUE  FZRPRV .
   40       DIVDF1 = (FZRDFL - FZRPRV)/H
            DIVDF2 = (DIVDF1 - DVDF1P)/(H + HPREV)
            HPREV = H
            DVDF1P = DIVDF1
            C = DIVDF1 + H*DIVDF2
            SQR = C*C - 4.*FZRDFL*DIVDF2
            IF (FNREAL .AND. REAL(SQR) .LT. 0.)  SQR = 0.
            SQR = SQRT(SQR)
            IF (REAL(C)*REAL(SQR)+AIMAG(C)*AIMAG(SQR) .LT. 0.) THEN
               DEN = C - SQR
            ELSE
               DEN = C + SQR
            END IF
            IF (ABS(DEN) .LE. 0.)  DEN = 1.
            H = -2.*FZRDFL/DEN
            FZRPRV = FZRDFL
            ZERO = ZERO + H
            IF (KOUNT .GT. MAXIT)        GO TO 99
C
   70       CALL DFLATE(FN, ZERO, I, KOUNT, FZR, FZRDFL, ZEROS, *1)
C                    CHECK FOR CONVERGENCE
            IF (ABS(H) .LT. EPS1*ABS(ZERO)) GO TO 99
            IF (MAX(ABS(FZR),ABS(FZRDFL)) .LT. EPS2)  GO TO 99
C                    CHECK FOR DIVERGENCE
            IF (ABS(FZRDFL) .GE. 10.*ABS(FZRPRV))  THEN
               H = H/2.
               ZERO = ZERO - H
                                         GO TO 70
            ELSE
                                         GO TO 40
            END IF
   99    ZEROS(I) = ZERO
  100 CONTINUE
                                         RETURN
      END
      SUBROUTINE DFLATE ( FN, ZERO, I, KOUNT, FZERO, FZRDFL, ZEROS, * )
C  TO BE CALLED IN  M U L L E R
      INTEGER I,KOUNT,  J
      COMPLEX FZERO,FZRDFL,ZERO,ZEROS(I),  DEN
```

```
      KOUNT = KOUNT + 1
      CALL FN(ZERO, FZERO)
      FZRDFL = FZERO
      IF (I .LT. 2)                          RETURN
      DO 10 J=2,I
         DEN = ZERO - ZEROS(J-1)
         IF (ABS(DEN) .EQ. 0.) THEN
            ZEROS(I) = ZERO*1.001
                                             RETURN 1
         ELSE
            FZRDFL = FZRDFL/DEN
         END IF
   10 CONTINUE
                                             RETURN
      END
```

Müller's method, like the other algorithms described in this chapter, finds one zero at a time. To find more than one zero it uses a procedure known as **deflation**. If, for example, one zero ξ_1 has already been found, the routine calculates the next zero by working with the function

$$f_1(x) = \frac{f(x)}{x - \xi_1} \tag{3.53}$$

We already met this technique when solving polynomial equations by Newton's method, in which case the *deflated* or *reduced* function $f_1(x)$ was a by-product of the algorithm. In Müller's method, if r zeros $\xi_1, \ldots, \xi_r$ have already been found, the next zero is obtained by working with the deflated function

$$f_r(x) = \frac{f(x)}{(x - \xi_1)(x - \xi_2) \cdots (x - \xi_r)} \tag{3.54}$$

If no estimates are given, the routine always looks for zeros in order of increasing magnitude since this will usually minimize round-off-error growth. Also, all zeros found using deflated functions are tested for accuracy by substitution into the original function $f(x)$. In practice some accuracy may be lost when a zero is found using deflation. Approximate zeros found using deflation may be refined by using these approximate zeros as initial guesses in Newton's method applied to the original function. In applying the Müller subroutine, the user can specify the number of zeros desired. Some functions, for example, may have an infinite number of zeros, of which only the first few may be of interest.

Example 3.13 Bessel's function $J_0(x)$ is given by the infinite series

$$J_0(x) = 1 - \frac{x^2}{2^2 \cdot 1 \cdot 1} + \frac{x^4}{2^4 \cdot 2! \cdot 2!} - \frac{x^6}{2^6 \cdot 3! \cdot 3!} + \cdots$$

It is known that $J_0(x)$ has an infinite number of real zeros. Find the first three positive zeros, using Algorithm 3.10. The machine results given below were obtained on an IBM 7094 using a standard library subroutine for $J_0(x)$ based on the series given above. The values of $J_0(x)$ were computed to maximum accuracy.

The iterations were all started with the approximations $x_0 = -1$, $x_1 = 1$, $x_2 = 0$ and were continued until either of the following error criteria was satisfied:

(*a*) $\dfrac{|x_{i+1} - x_i|}{|x_{i+1}|} < 10^{-6}$

(*b*) $|J_0(x_i)| < 10^{-20}$

The converged values are correct to at least six significant figures. Note that the zeros are obtained in ascending order of magnitude.

COMPUTER RESULTS FOR EXAMPLE 3.13

Zero 1	Zero 2	Zero 3
−0.09999999E 01	−0.09999999E 01	−0.09999999E 01
0.09999999E 01	0.09999999E 01	0.09999999E 01
0.	0.	0.
0.20637107E 01	0.36557332E 01	0.47983123E 01
0.23167706E 01	0.44416171E 01	0.59396663E 01
0.23970029E 01	0.50863190E 01	0.70758440E 01
0.24047983E 01	0.55024961E 01	0.88981197E 01
0.24048255E 01	0.55202182E 01	0.92976399E 01
0.24048255E 01	0.55200780E 01	0.86854592E 01
	0.55200780E 01	0.86529856E 01
		0.86537299E 01
		0.86537278E 01

All the following examples were run on a CDC 6500 computer using Algorithm 3.10. The error criteria for these examples were $\varepsilon_1 = \varepsilon_2 = 10^{-8}$, and all used the same starting values (0.5, −0.5, 0.0) followed by deflation. Although the results are printed to 8 significant figures, one should recall that on a CDC 6500 the floating-point word length is 14 decimal digits. The output consists of the real and imaginary (if applicable) parts of the converged approximations to the roots, and the real and imaginary parts of the value of the function at those roots.

Example 3.14 Find all the zeros of the polynomial $p(x) = x^3 + x - 3$.

ROOT		F(ROOT)	
REAL PART	IMAGINARY PART	REAL PART	IMAGINARY PART
1.2134117E + 00	0.	−4.2632564E − 14	0.
−6.0670583E − 01	1.4506122E + 00	2.8421709E − 14	4.2632564E − 14
−6.0670583E − 01	−1.4506122E + 00	2.8421709E − 14	−2.6290081E − 13

Compare these results with those obtained in Example 3.10, where we computed the solutions on a hand calculator. Note that since $p(x)$ has real coefficients, the complex roots occur in complex-conjugate pairs. Note as well that no estimates of the complex roots had to be provided. While Newton's method can be used to find complex roots, it must be supplied with a good estimate of that root, an estimate that

1.8 Bairstow's Method for Quadratic Factors

The methods considered so far are difficult to use to find a complex root of a polynomial. It is true that Newton's and Muller's methods work satisfactorily, provided that we begin with initial estimates that are complex-valued; however, in a hand computation, performing the multiplications and divisions of complex numbers is awkward. There is no problem in a computer program if complex arithmetic capabilities exist, but the execution is slower.

For polynomials, the complex roots occur in conjugate pairs if the coefficients are all real-valued. For this case, if we extract the quadratic factors that are the products of the pairs of complex roots, we can avoid complex arithmetic because such quadratic factors have real coefficients. We first develop the algorithm for synthetic division by a trial quadratic, $x^2 - rx - s$, which, hopefully, is near to the desired factor of the polynomial:

$$\begin{aligned} P_n(x) &= a_0x^n + a_1x^{n-1} + \cdots + a_n \\ &= (x^2 - rx - s)Q_{n-2}(x) + \text{remainder} \\ &= (x^2 - rx - s)(b_0x^{n-2} + b_1x^{n-3} + \cdots + b_{n-3}x + b_{n-2}) \\ &\quad + b_{n-1}(x - r) + b_n. \end{aligned}$$

(The remainder is the linear term $b_{n-1}(x - r) + b_n$, written in this form to provide later simplicity. If $x^2 - rx - s$ is an exact divisor of $P_n(x)$, then b_{n-1} and b_n will both be zero.) The negative signs in the factor are also for later simplification.

On multiplying out and equating coefficients of like powers of x, we get

$$\begin{array}{lll} a_0 = b_0 & & b_0 = a_0 \\ a_1 = b_1 - rb_0 & & b_1 = a_1 + rb_0 \\ a_2 = b_2 - rb_1 - sb_0 & & b_2 = a_2 + rb_1 + sb_0 \\ a_3 = b_3 - rb_2 - sb_1 & \text{or} & b_3 = a_3 + rb_2 + sb_1 \\ \vdots & & \vdots \\ a_{n-1} = b_{n-1} - rb_{n-2} - sb_{n-3} & & b_{n-1} = a_{n-1} + rb_{n-2} + sb_{n-3} \\ a_n = b_n - rb_{n-1} - sb_{n-2} & & b_n = a_n + rb_{n-1} + sb_{n-2} \end{array} \tag{1.1}$$

We would like both b_{n-1} and b_n to be zero, for that would show $x^2 - rx - s$ to be a quadratic factor of the polynomial (because the remainder is zero). This will normally not be so; if we properly change the values of r and s, we can make the remainder zero, or at least make its coefficients smaller. Obviously b_{n-1} and b_n are both functions of the two parameters r and s. Expanding these as a Taylor series for a function of two variables* in terms of $(r^* - r)$ and $(s^* - s)$, where $(r^* - r)$ and

* Appendix A reviews this.

$(s^* - s)$ are presumed small so that terms of higher order than the first are negligible, we obtain

$$b_{n-1}(r^*,s^*) = b_{n-1}(r,s) + \frac{\partial b_{n-1}}{\partial r}(r^* - r) + \frac{\partial b_{n-1}}{\partial s}(s^* - s) + \cdots,$$

$$b_n(r^*,s^*) = b_n(r,s) + \frac{\partial b_n}{\partial r}(r^* - r) + \frac{\partial b_n}{\partial s}(s^* - s) + \cdots.$$

Let us take (r^*,s^*) as the point at which the remainder is zero, and

$$r^* - r = \Delta r, \qquad s^* - s = \Delta s.$$

(Δr and Δs are increments to add to the original r and s to get the new values r^* and s^* for which the remainder is zero.) Then

$$b_{n-1}(r^*,s^*) = 0 \simeq b_{n-1} + \frac{\partial b_{n-1}}{\partial r}\Delta r + \frac{\partial b_{n-1}}{\partial s}\Delta s,$$

$$b_n(r^*,s^*) = 0 \simeq b_n + \frac{\partial b_n}{\partial r}\Delta r + \frac{\partial b_n}{\partial s}\Delta s.$$

All the terms on the right are to be evaluated at (r,s). We wish to solve these two equations simultaneously for the unknown Δr and Δs, so we need to evaluate the partial derivatives.

Bairstow showed that the required partial derivatives can be obtained from the b's by a second synthetic division by the factor $x^2 - rs - s$ in just the same way that the b's are obtained from the a's. Define a set of c's by the following relations shown at the left, and compare these to the partial derivatives in the right columns:

$c_0 = b_0$	$\dfrac{\partial b_0}{\partial r} = \dfrac{\partial a_0}{\partial r} = 0$	$\dfrac{\partial b_0}{\partial s} = \dfrac{\partial a_0}{\partial s} = 0$
$c_1 = b_1 + rc_0$	$\dfrac{\partial b_1}{\partial r} = r\dfrac{\partial b_0}{\partial r} + b_0 = b_0 = c_0$	$\dfrac{\partial b_1}{\partial s} = \dfrac{\partial a_1}{\partial s} + r\dfrac{\partial b_0}{\partial s} = 0$
$c_2 = b_2 + rc_1 + sc_0$	$\dfrac{\partial b_2}{\partial r} = r\dfrac{\partial b_1}{\partial r} + b_1 = c_1$	$\dfrac{\partial b_2}{\partial s} = r\dfrac{\partial b_1}{\partial s} + s\dfrac{\partial b_0}{\partial s} + b_0 = b_0 = c_0$
$c_3 = b_3 + rc_2 + sc_1$	$\dfrac{\partial b_3}{\partial r} = r\dfrac{\partial b_2}{\partial r} + b_2 + s\dfrac{\partial b_1}{\partial r} = b_2 + rc_1 + sc_0 = c_2$	$\dfrac{\partial b_3}{\partial s} = r\dfrac{\partial b_2}{\partial s} + s\dfrac{\partial b}{\partial s} + b_1 = b_1 + rc_0 = c_1$
$\vdots$	$\vdots$	$\vdots$
$c_{n-1} = b_{n-1} + rc_{n-2} + sc_{n-3}$	$\dfrac{\partial b_{n-1}}{\partial r} = r\dfrac{\partial b_{n-2}}{\partial r} + b_{n-2} + s\dfrac{\partial b_{n-3}}{\partial r} = b_{n-2} + rc_{n-3} + sc_{n-4} = c_{n-2}$	$\dfrac{\partial b_{n-1}}{\partial s} = r\dfrac{\partial b_{n-2}}{\partial s} + s\dfrac{\partial b_{n-3}}{\partial s} + b_{n-3} = b_{n-3} + rc_{n-4} + sc_{n-5} = c_{n-3}$

Hence the partial derivatives that we need are equal to the properly corresponding c's. Our simultaneous equations become, where Δr and Δs are unknowns to be solved for,

$$\begin{aligned} -b_{n-1} &= c_{n-2}\Delta r + c_{n-3}\Delta s, \\ -b_n &= c_{n-1}\Delta r + c_{n-2}\Delta s. \end{aligned}$$

We express the solution as ratios of determinants:

$$\Delta r = \frac{\begin{vmatrix} -b_{n-1} & c_{n-3} \\ -b_n & c_{n-2} \end{vmatrix}}{\begin{vmatrix} c_{n-2} & c_{n-3} \\ c_{n-1} & c_{n-2} \end{vmatrix}},$$

$$\Delta s = \frac{\begin{vmatrix} c_{n-2} & -b_{n-1} \\ c_{n-1} & -b_n \end{vmatrix}}{\begin{vmatrix} c_{n-2} & c_{n-3} \\ c_{n-1} & c_{n-2} \end{vmatrix}}.$$

As an exercise, the student should write the algorithm for this method.

EXAMPLE 1.3 Find the quadratic factors of

$$x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0.$$

Use $x^2 + x + 1$ as starting factor ($r = -1, s = -1$). (Frequently $r = s = 0$ are used as starting values if no information as to an approximate factor is known.) Equations (1.1) lead to a double synthetic-division scheme as follows:

	a_0	a_1	a_2	a_3	a_4
	1	−1.1	2.3	0.5	3.3
$r = -1$		−1.0	2.1	−3.4	0.8
$s = -1$		—	−1.0	2.1	−3.4
	1	−2.1	3.4	(−0.8)	(0.7)
		−1.0	3.1	−5.5	
		—	−1.0	3.1	
	1	(−3.1)	(5.5)	(−3.2)	
		c_{n-3}	c_{n-2}	c_{n-1}	

b_{n-1} b_n

Note that the equations for b_1 and c_1 have no term involving s. The dashes in the preceding tableau represent these missing factors. Then

$$\Delta r = \frac{\begin{vmatrix} 0.8 & -3.1 \\ -0.7 & 5.5 \end{vmatrix}}{\begin{vmatrix} 5.5 & -3.1 \\ -3.2 & 5.5 \end{vmatrix}} = \frac{2.23}{20.33} = 0.11, \qquad r^* = -1 + 0.11 = -0.89,$$

$$\Delta s = \frac{\begin{vmatrix} 5.5 & 0.8 \\ -3.2 & -0.7 \end{vmatrix}}{20.33} = \frac{-1.29}{20.33} = -0.06, \qquad s^* = -1 - 0.06 = -1.06.$$

The second trial yields

	1	−1.1	2.3	0.5	3.3
−0.89		−0.89	1.77	−2.68	0.06
−1.06		—	−1.06	2.11	−3.17
	1	−1.99	3.01	−0.07	0.17 ← b's
		−0.89	2.56	−4.01	
		—	−1.06	3.05	
		−2.88	4.51	−1.03	← c's

$$\Delta r = \frac{\begin{vmatrix} 0.07 & -2.88 \\ -0.17 & 4.51 \end{vmatrix}}{\begin{vmatrix} 4.51 & -2.88 \\ -1.03 & 4.51 \end{vmatrix}} = \frac{-0.175}{17.374} = -0.010, \qquad r^* = -0.89 - 0.010 = -0.900,$$

$$\Delta s = \frac{\begin{vmatrix} 4.51 & 0.07 \\ -1.03 & -0.17 \end{vmatrix}}{17.374} = \frac{-0.694}{17.374} = -0.040, \qquad s^* = -1.06 - 0.040 = -1.100.$$

The exact factors are $(x^2 + 0.9x + 1.1)(x^2 - 2x + 3)$.

Observe that the other factor has its coefficients in the row of b's.

1.9 Other Methods for Polynomials*

Of the many other methods for finding the roots of polynomials, we discuss four in this section. Three methods—the QD algorithm, Graeffe's root-squaring method, and Lehmer's method—do not require that we start with a value near to a root. The fourth, Laguerre's method, does need a starting value, but it is remarkably efficient.

*The methods of this section are not widely used. You may therefore want to skip this section.

1.8 Bairstow's Method for Quadratic Factors

The methods considered so far are difficult to use to find a complex root of a polynomial. It is true that Newton's and Muller's methods work satisfactorily, provided that we begin with initial estimates that are complex-valued; however, in a hand computation, performing the multiplications and divisions of complex numbers is awkward. There is no problem in a computer program if complex arithmetic capabilities exist, but the execution is slower.

For polynomials, the complex roots occur in conjugate pairs if the coefficients are all real-valued. For this case, if we extract the quadratic factors that are the products of the pairs of complex roots, we can avoid complex arithmetic because such quadratic factors have real coefficients. We first develop the algorithm for synthetic division by a trial quadratic, $x^2 - rx - s$, which, hopefully, is near to the desired factor of the polynomial:

$$\begin{aligned} P_n(x) &= a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0 \\ &= (x^2 - rx - s)Q_{n-2}(x) + \text{remainder} \\ &= (x^2 - rx - s)(b_n x^{n-2} + b_{n-1}x^{n-3} + \cdots + b_3 x + b_2) + \text{remainder}, \\ \text{remainder} &= b_1(x - r) + b_0. \end{aligned}$$

If $x^2 - rx - s$ is an exact factor of $P_n(x)$, then b_1 and b_0 will both be zero. The negative signs in the factor are conventional in developing the algorithm.

On multiplying out and equating coefficients of like powers of x, we get

$$\left.\begin{aligned} a_n &= b_n \\ a_{n-1} &= b_{n-1} - rb_n \\ a_{n-2} &= b_{n-2} - rb_{n-1} - sb_n \\ a_{n-3} &= b_{n-3} - rb_{n-2} - sb_{n-1} \\ &\vdots \\ a_1 &= b_1 - rb_2 - sb_3 \\ a_0 &= b_0 - rb_1 - sb_2 \end{aligned}\right\} \quad \text{or} \quad \left\{\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + rb_n \\ b_{n-2} &= a_{n-2} + rb_{n-1} + sb_n \\ b_{n-3} &= a_{n-3} + rb_{n-2} + sb_{n-1} \\ &\vdots \\ b_1 &= a_1 + rb_2 + sb_3 \\ b_0 &= a_0 + rb_1 + sb_2 \end{aligned}\right. \tag{1.1}$$

We would like both b_1 and b_0 to be zero, for that would show $x^2 - rx - s$ to be a quadratic factor of the polynomial (because the remainder is zero). This will normally not be so; if we properly change the values of r and s, we can make the remainder zero, or at least make its coefficients smaller. Obviously b_1 and b_0 are both functions of the two parameters r and s. Expanding these as a Taylor series for a function of two variables* in terms of $(r^* - r)$ and $(s^* - s)$, where $(r^* - r)$ and $(s^* - s)$ are presumed small so that

*Appendix A reviews this.

terms of higher order than the first are negligible, we obtain

$$b_1(r^*, s^*) = b_1(r, s) + \frac{\partial b_1}{\partial r}(r^* - r) + \frac{\partial b_1}{\partial s}(s^* - s) + \cdots,$$

$$b_0(r^*, s^*) = b_0(r, s) + \frac{\partial b_0}{\partial r}(r^* - r) + \frac{\partial b_0}{\partial s}(s^* - s) + \cdots.$$

Let us take (r^*, s^*) as the point at which the remainder is zero, and

$$r^* - r = \Delta r, \qquad s^* - s = \Delta s.$$

(Δr and Δs are increments to add to the original r and s to get the new values r^* and s^* for which the remainder is zero.) Then

$$b_1(r^*, s^*) = 0 \simeq b_1 + \frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s,$$

$$b_0(r^*, s^*) = 0 \simeq b_0 + \frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s.$$

All the terms on the right are to be evaluated at (r, s). We wish to solve these two equations simultaneously for the unknown Δr and Δs, so we need to evaluate the partial derivatives.

Bairstow showed that the required partial derivatives can be obtained from the b's by a second synthetic division by the factor $x^2 - rx - s$ in just the same way that the b's are obtained from the a's. (Observe that this is analogous to getting the derivative of a polynomial by dividing the reduced polynomial by $(x - x_1)$.)

Define a set of c's by the following relations shown at the left, and compare these to the partial derivatives in the right-hand columns:

$c_n = b_n$	$\dfrac{\partial b_n}{\partial r} = \dfrac{\partial a_n}{\partial r} = 0$	$\dfrac{\partial b_n}{\partial s} = \dfrac{\partial a_n}{\partial s} = 0$
$c_{n-1} = b_{n-1} + rc_n$	$\dfrac{\partial b_{n-1}}{\partial r} = r\dfrac{\partial b_n}{\partial r} + b_n = b_n = c_n$	$\dfrac{\partial b_{n-1}}{\partial s} = \dfrac{\partial a_{n-1}}{\partial s} + r\dfrac{\partial b_n}{\partial s} = 0$
$c_{n-2} = b_{n-2} + rc_{n-1} + sc_n$	$\dfrac{\partial b_{n-2}}{\partial r} = r\dfrac{\partial b_{n-1}}{\partial r} + b_{n-1} = c_{n-1}$	$\dfrac{\partial b_{n-2}}{\partial s} = r\dfrac{\partial b_{n-1}}{\partial s} + s\dfrac{\partial b_n}{\partial s} = b_0$ $= b_n = c_n$
$c_{n-3} = b_{n-3} + rc_{n-2} + sc_{n-1}$	$\dfrac{\partial b_{n-3}}{\partial r} = r\dfrac{\partial b_{n-2}}{\partial r} + b_{n-2} + s\dfrac{\partial b_{n-1}}{\partial r}$ $= b_{n-2} + rc_{n-1} + sc_n = c_{n-2}$	$\dfrac{\partial b_{n-3}}{\partial s} = r\dfrac{\partial b_{n-2}}{\partial s} + s\dfrac{\partial b_{n-1}}{\partial s} + b_{n-1}$ $= b_{n-1} + rc_n = c_{n-1}$
$\vdots$	$\vdots$	$\vdots$
$c_1 = b_1 + rc_2 + sc_3$	$\dfrac{\partial b_1}{\partial r} = r\dfrac{\partial b_2}{\partial r} + b_2 + s\dfrac{\partial b_3}{\partial r}$ $= b_2 + rc_3 + sc_4$ $= c_2$	$\dfrac{\partial b_1}{\partial s} = r\dfrac{\partial b_2}{\partial s} + s\dfrac{\partial b_3}{\partial s} + b_3$ $= b_3 + rc_4 + sc_5$ $= c_3$

Hence the partial derivatives that we need are equal to the properly corresponding c's. Our simultaneous equations become, where Δr and Δs are unknowns to be solved for,

$$\begin{aligned} -b_1 &= c_2\Delta r + c_3\Delta s, \\ -b_0 &= c_1\Delta r + c_2\Delta s. \end{aligned}$$

We express the solution as ratios of determinants, using Cramer's rule:

$$\Delta r = \frac{\begin{vmatrix} -b_1 & c_3 \\ -b_0 & c_2 \end{vmatrix}}{\begin{vmatrix} c_2 & c_3 \\ c_1 & c_2 \end{vmatrix}},$$

$$\Delta s = \frac{\begin{vmatrix} c_2 & -b_1 \\ c_1 & -b_0 \end{vmatrix}}{\begin{vmatrix} c_2 & c_3 \\ c_1 & c_2 \end{vmatrix}}.$$

As an exercise, the student should write the algorithm for this method.

EXAMPLE 1.3 Find the quadratic factors of

$$x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0.$$

Use $x^2 + x + 1$ as starting factor ($r = -1$, $s = -1$). (Frequently $r = s = 0$ are used as starting values if no information as to an approximate factor is known.) Equations (1.1) lead to a double synthetic-division scheme as follows:

	a_4	a_3	a_2	a_1	a_0
	1	−1.1	2.3	0.5	3.3
$r = -1$		−1.0	2.1	−3.4	0.8
$s = -1$		—	−1.0	2.1	−3.4
	1	−2.1	3.4	(−0.8)	(0.7)
		−1.0	3.1	−5.5	
		—	−1.0	3.1	
	1	(−3.1)	(5.5)	(−3.2)	
		c_3	c_2	c_1	

(b_1 and b_0 mark the circled −0.8 and 0.7.)

Note that the equations for b_1 and c_1 have no term involving s. The dashes in the preceding tableau represent these missing factors. Then

$$\Delta r = \frac{\begin{vmatrix} 0.8 & -3.1 \\ -0.7 & 5.5 \end{vmatrix}}{\begin{vmatrix} 5.5 & -3.1 \\ -3.2 & 5.5 \end{vmatrix}} = \frac{2.23}{20.33} = 0.11, \qquad r^* = -1 + 0.11 = -0.89,$$

$$\Delta s = \frac{\begin{vmatrix} 5.5 & 0.8 \\ -3.2 & -0.7 \end{vmatrix}}{20.33} = \frac{-1.29}{20.33} = -0.06, \qquad s^* = -1 - 0.06 = -1.06.$$

The second trial yields

$$\begin{array}{r|rrrrr} & 1 & -1.1 & 2.3 & 0.5 & 3.3 \\ -0.89 & & -0.89 & 1.77 & -2.68 & 0.06 \\ -1.06 & & \text{—} & -1.06 & 2.11 & -3.17 \\ \hline & 1 & -1.99 & 3.01 & -0.07 & 0.17 \quad \leftarrow b\text{'s} \\ & & -0.89 & 2.56 & -4.01 & \\ & & \text{—} & -1.06 & 3.05 & \\ \hline & & -2.88 & 4.51 & -1.03 & \quad \leftarrow c\text{'s} \end{array}$$

$$\Delta r = \frac{\begin{vmatrix} 0.07 & -2.88 \\ -0.17 & 4.51 \end{vmatrix}}{\begin{vmatrix} 4.51 & -2.88 \\ -1.03 & 4.51 \end{vmatrix}} = \frac{-0.175}{17.374} = -0.010, \qquad r^* = -0.89 - 0.010 = -0.900,$$

$$\Delta s = \frac{\begin{vmatrix} 4.51 & 0.07 \\ -1.03 & -0.17 \end{vmatrix}}{17.374} = \frac{-0.694}{17.374} = -0.040, \qquad s^* = -1.06 - 0.040 = -1.100.$$

The exact factors are $(x^2 + 0.9x + 1.1)(x^2 - 2x + 3)$.

Observe that the other factor has its coefficients in the row of b's. ▲

If we use MATLAB to divide a polynomial by a quadratic, as we do in Bairstow's method, we do not get the same remainder that Bairstow gives:

```
N = [1 -1.1 2.3 .5 3.3]; D = [1 1 1];
[q, r] = deconv(N, D)
```

results in

```
q =
     1.0000 -2.1000 3.4000
r =
     0  0  0  -.8000 -0.1000
```

b) Using only three significant digits (chopped) in your arithmetic, find the solution to $Hx = b$. Explain why the answers are so poor.
c) Using only three significant digits, but rounding, again find the solution and compare it to that obtained in part (b).
d) Using five significant digits in your arithmetic, again find the solution and compare it to those found in parts (b) and (c).

Section 2.7

26. Find the determinant of the matrix

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

by row operations to make it (a) upper-triangular, (b) lower-triangular.

▸27. Find the determinant of the matrix

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

28. Invert the coefficient matrix in Exercise 5, and then use the inverse to generate the solution. Note that a symmetric matrix has a symmetric inverse.

29. If the constant vector in Exercise 5 is changed to one with components (13, 11, −22), what is now the solution? Observe that the inverse obtained in Exercise 28 gives the answer readily.

30. Attempt to find the inverse of the coefficient matrix in Exercise 8. Note that a singular matrix has no inverse.

31. a) Find the determinant of the Hilbert matrix in Exercise 25. A small value of the determinant (when the matrix has elements of the order of unity) indicates ill-conditioning.
▸b) Find the inverse of the Hilbert matrix in Exercise 25. The inverse of an ill-conditioned matrix has some very large elements in comparison to the elements of the original matrix.

32. Both Gaussian elimination and the Gauss–Jordan method can be adapted to invert a matrix. In Section 2.7, we say "it is more efficient to use the Gaussian elimination algorithm." Verify this for the specific case of a 3 × 3 matrix by counting arithmetic operations for each method.

Section 2.8

33. Find the Euclidean norm of these vectors and matrices. find ∞, 2, e norm

a) $x = [1.06, -2.15, 14.05, 0.0]$.
b) $y = [1, 0, 0, 0]$.
c) $z^T = [1, 1, 1]$.
d) $A = \begin{bmatrix} 14 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -2 \end{bmatrix}$.

e) $B = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

g) Find the norms of AB; of Az.
h) Does the triangle inequality of Eq

34. Find the Euclidean norm of the Hilbe

▸35. Find the Euclidean norm of the invers

Section 2.9*

36. Consider the system $Ax = b$, where

$$A = \begin{bmatrix} 3.01 & \\ 1.27 & \\ 0.987 & - \end{bmatrix}$$

▸a) Using double precision (or a calcul
b) Solve the system using three-digit (c this solution $\bar{x}$.
c) Compare x and $\bar{x}$ and compute $e =$
d) Is the system ill-conditioned? What

37. Repeat Exercise 36, except change the

▸38. Suppose, in Exercise 36, that uncertain the elements of A. Specifically, suppose of 0.987. What change does this cause

39. Compute the residuals for the imperfect in this computation.

40. What are the condition numbers of the c 1-norms.

▸41. Verify Eq. (2.14), using the results in E

42. Verify Eq. (2.14), using the results in E

43. Verify Eq. (2.15) for the results of Exer

44. Apply iterative improvement to the impe

▸45. Apply iterative improvement to the impe

Section 2.10

46. Solve Exercise 5 by the Jacobi method, b the rate of convergence when Gauss–Seid

47. Solve Exercise 5 by Gauss–Seidel iteratio

▸48. The pair of equations

$$x_1$$
$$3x_1$$

can be rearranged to give $x_1 = 3 - 2x_2$, rearrangement, beginning with a vector ve observe divergence. Now apply Gauss–Se

49. Solve Exercise 20(a) by Gauss–Seidel iter

EGM 5346

COMP. ENG. ANAL.

6/9/08

INTERPOLATION

LAGRANGE POLYNOMIALS

$$P_n(x) = \sum_{k=0}^{n} y_k l_k(x)$$

$$l_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)}$$

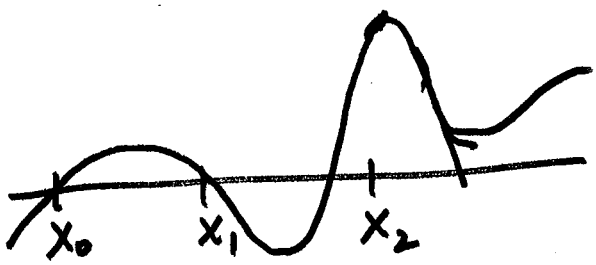
$$P_1(x) = y_0 l_0(x) + y_1 l_1(x) \qquad l_0(x) = \frac{(x - x_1)}{(x_0 - x_1)} \qquad l_1(x) = \frac{(x - x_0)}{(x_1 - x_0)}$$

$$P_2(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x)$$

$$l_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}$$

$$l_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

$$l_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$



$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$$

$$= (((((a_n x + a_{n-1}) x + a_{n-2}) x + \cdots) \quad) \quad) x + a_0)$$

Create $p(x)$: $(x_0, y_0), (x_1, y_1) \cdots (x_n, y_n)$

$$p(x) = \sum_{i=0}^{n} a_i (x-c)^i = a_0 + a_1(x-c) + a_2(x-c)^2 + \cdots$$
$$a_n(x-c)^n$$

Newton form.

$$p_n(x) = A_0 + A_1(x-x_0) + A_2(x-x_0)(x-x_1) + \cdots + A_n \prod_{i=0}^{n-1} (x-x_i)$$

$A_i = f[x_0, x_1, \ldots, x_i]$ i^{th} divided difference

$= (f[x_1, x_2, \ldots, x_i] - f[x_0, x_1, \ldots, x_{i-1}]) / (x_i - x_0)$

$A_0 = f[x_0] = f(x_0)$

$A_1 = f[x_0, x_1] = (f[x_1] - f[x_0]) / (x_1 - x_0) = (f(x_1) - f(x_0)) / (x_1 - x$

$A_2 = f[x_0, x_1, x_2] = (f[x_1, x_2] - f[x_0, x_1]) / (x_2 - x_0)$

Efficient Algorithm to calculate $p_n(x)$

Given the A_i's, x's & the value of x

set $b_n = A_n$

for $i = n-1, n-2, \ldots, 0$ do:

  $b_i = A_i + (x - x_i) b_{i+1}$

$p_n(x) = b_0$

DIVIDED DIFFERENCE TABLE

x_0	$f(x_0)=y_0$	$f[x_0,x_1]$	$f[x_0,x_1,x_2]$	$f[x_0,x_1,x_2,x_3]$	$\cdots$ $f[x_0,x_1,x_2,\ldots,x_n]$
x_1	$f(x_1)=y_1$	$f[x_1,x_2]$	$f[x_1,x_2,x_3]$		
x_2	$f(x_2)=y_2$	$f[x_2,x_3]$			
x_3	$f(x_3)=y_3$				
$\vdots$	$\vdots$	$\vdots$	$f[x_{n-2},x_{n-1},x_n]$		
x_n	$f(x_n)=y_n$	$f[x_{n-1},x_n]$			

Algorithm to generate DDT

Given $x_0, x_1, x_2, \ldots, x_n$ & $f(x_0), f(x_1), \ldots, f(x_n)$

for $k = 1, \ldots, n$ do:

for $i = 0, \ldots, n-k$ do:

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$$

$$P_n(x) = \sum_{i=0}^{n} f[x_i, \ldots, x_n] \prod_{j=i+1}^{n} (x - x_j)$$

$$P_3(x) = f[x_0,..,x_3](x-x_0)(x-x_1)(x-x_2) + f[x_0,x_1,x_2](x-x_0)(x-x_1) + f[x_0,x_1](x-x_0) + f(x_0) \Rightarrow \sum_{i=0}^{n} f[x_0,x_1,\ldots,x_i] \prod_{j=0}^{i-1} (x-x_j)$$

$$P_3(x) = f[x_0,x_1,\ldots,x_3](x-x_1)(x-x_2)(x-x_3) + f[x_1,x_2,x_3](x-x_2)(x-x_3) + f[x_2,x_3](x-x_3) + f(x_3) \Rightarrow \sum_{i=0}^{n} f[x_i,x_{i+1},\ldots x_n] \prod_{j=i+1}^{n} (x-x_j)$$

INTERPOLATION USING INCREASING NO. OF INTERPOLATION POINTS

GIVEN $x_0, x_1, \ldots, x_n$ & $f(x_0), f(x_1), \ldots, f(x_n)$ and apt. $\bar{x}$ for which I want to find $p(\bar{x})$

define $f[x_0] = f(x_0)$, [redacted] $\cdot\ P_0(\bar{x}) = f(x_0)$
and $\psi_0(\bar{x}) = 1$

for $k = 0, 1, 2, \ldots$ until satisfied

$$f[x_{k+1}] := f(x_{k+1})$$

for $i = k, k-1, \ldots, 0$ do:

$$f[x_i, x_{i+1}, \ldots, x_{k+1}] = \frac{f[x_{i+1}, \ldots, x_{k+1}] - f[x_i, x_{i+1}, \ldots, x_k]}{x_{k+1} - x_i}$$

$$\psi_{k+1}(\bar{x}) = \psi_k(\bar{x})(\bar{x} - x_k)$$

$$p_{k+1}(\bar{x}) = p_k(\bar{x}) + f[x_0, x_1, \ldots, x_{k+1}]\,\psi_{k+1}(\bar{x})$$

Error in the polynomial is strongly dependent on placement of points.

Evenly spaced set of points, the error is largest away from the center of the interval.

Error is more pronounced at the ends as n increases.

In general evaluate using the polynomial in the center of interval

WE ARE INTERESTED IN $f[x_i, x_{i+1}, \ldots, x_n]$

STORE $d_0 = f(x_0)$ $d_1 = f(x_1)$, $d_2 = f(x_2)$ .... $d_n = f(x_n)$

for $k = 1, \ldots, n$ do:

  for $j = 0, \ldots, n-k$ do:

    $d_i = (d_{i+1} - d_i)/(x_{i+k} - x_i)$

d_i's will contain $f[x_i, x_{i+1}, \ldots, x_n]$ $\quad i = 0, 1, 2, \ldots, n$

Given z

LET $v = f[x_0, x_1, \ldots, x_n]$

  for $i = 1, 2, \ldots, n$ do:

    $v = f[x_i, x_{i+1}, \ldots, x_n] + (z - x_i)v$

$v = p_n(z)$

BASED UPON UPPER DIAG.

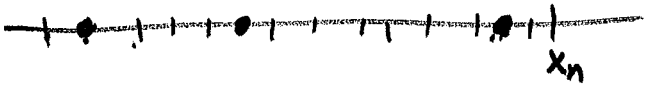
$$p_n(x) = \sum_{i=0}^{n} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1} (x - x_j) = \sum_{i=0}^{n} f[x_0, x_1, \ldots, x_i]\, \psi_i(x)$$

$$p_{n+1}(x) = \sum_{i=0}^{n+1} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1} (x - x_j) = \sum_{i=0}^{n+1} f[x_0, x_1, \ldots, x_i]\, \psi_i(x)$$

$$= \sum_{i=0}^{n} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1} (x - x_j) + f[x_0, x_1, \ldots, x_{n+1}] \left\{ \prod_{j=0}^{n-1} (x - x_j) \right\} (x - x_n)$$

$$p_{n+1}(x) = p_n(x) + \underline{f[x_0, x_1, \ldots, x_{n+1}]\, \psi_n(x) \cdot (x - x_n)}$$

$$|p_{n+1}(x) - p_n(x)| \leq \text{TOL}$$



Next, we prove (i). If $y_0 = y_1 = \cdots = y_n$, then (i) is just a restatement of (2.33). Otherwise, we may assume that

$$y_0 \le y_1 \le \cdots \le y_n$$

and then $y_0 < y_n$. But then we may find, for all r, $x_0^{(r)} < \cdots < x_n^{(r)}$ in $[a, b]$ so that $\lim_{r\to\infty} x_i^{(r)} = y_i$, $i = 0, \ldots, n$. By Theorem 2.2, we can find then $\xi^{(r)} \in [x_0^{(r)}, x_n^{(r)}]$ so that

$$f\left[x_0^{(r)}, \ldots, x_n^{(r)}\right] = f^{(n)}(\xi^{(r)})/n! \qquad \text{for } r = 1, 2, 3, \ldots$$

But then, by (ii) just proved for this case,

$$f\left[y_0, \ldots, y_n\right] = \lim_{r\to\infty} f\left[x_0^{(r)}, \ldots, x_n^{(r)}\right] = \lim_{r\to\infty} f^{(n)}(\xi^{(r)})/n!$$
$$= f^{(n)}(\xi)/n!$$

for some $\xi \in [\lim x_0^{(r)}, \lim x_n^{(r)}] = [y_0, y_n]$, by the continuity of $f^{(n)}(x)$, which proves (i).

Finally, to prove (ii) in the case that $y_0 = y_1 = \cdots = y_n$, we now use (i) to conclude the existence of $\xi^{(r)} \in [\min_i x_i^{(r)}, \max_i x_i^{(r)}]$ so that $f[x_0^{(r)}, \ldots, x_n^{(r)}] = f^{(n)}(\xi^{(r)})/n!$ for all r. But then, since $y_0 = \cdots = y_n$ and $\lim x_i^{(r)} = y_i$, all i, we have $\lim \xi^{(r)} = y_0$ and so, with (2.36) and the continuity of $f^{(n)}(x)$,

$$f\left[y_0, \ldots, y_n\right] = f^{(n)}(y_0)/n! = \lim f^{(n)}(\xi^{(r)})/n! = \lim f\left[x_0^{(r)}, \ldots, x_n^{(r)}\right]$$

This proves both (i) and (ii) for $n = k$ and for all choices of $y_0, \ldots, y_n$ in $[a, b]$.

We conclude this section with some interesting consequences of Theorem 2.5. It follows at once that the function

$$g_n(x) = f\left[x_0, \ldots, x_n, x\right]$$

which appears in the error term for polynomial interpolation is defined for all x and is a *continuous* function of x if $f(x)$ is sufficiently smooth. Thus it follows that

$$f(x) = \sum_{i=0}^{n} f\left[x_0, \ldots, x_i\right] \prod_{j=0}^{i-1} (x - x_j) + f\left[x_0, \ldots, x_n, x\right] \prod_{j=0}^{n} (x - x_j) \tag{2.37}$$

for all x, and not only for $x \neq x_0, \ldots, x_n$ [see (2.16)], and also for all $x_0, \ldots, x_n$, distinct or not, in case $f(x)$ has enough derivatives.

Further, if $f(x)$ is sufficiently often differentiable, then $g_n(x)$ is differentiable. For by the definition of derivatives,

$$g_n'(x) = \lim_{h\to 0} g_n[x, x + h]$$

if this limit exists. On the other hand,

by Theorem 2.5. Hence

$$\frac{d}{dx} f\left[x_0, \ldots, x_n, x\right] = f\left[x_0, \ldots, x_n, x, x\right] \tag{2.38}$$

Finally, it explains our definition of osculatory interpolation as *repeated* interpolation. For it shows that the interpolating polynomial at points $x_0, \ldots, x_n$ converges to the interpolating polynomial at points $y_0, \ldots, y_n$ as $x_i \to y_i$, all i. Thus, k-fold interpolation at a point is the limiting case as we let k distinct interpolation points coalesce. The student is familiar with this phenomenon in the case $n = 1$ of linear interpolation. In this case, the straight line $p_1(x) = f(x_0) + f[x_0, x_1](x - x_0)$ is a secant to (the graph of) $f(x)$ which goes over into the tangent $\hat{p}_1(x) = f(y) + (x - y)f'(y)$ as both x_0 and x_1 approach the point y, and $\hat{p}_1(x)$ agrees with $f(x)$ in value *and* slope at $x = y$.

Example 2.9 With $f(x) = \ln x$, calculate $f(1.5)$ by cubic interpolation, using $f(1) = 0$, $f(2) = 0.693147$, $f'(1) = 1$, $f'(2) = 0.5$.

In this case, the four interpolation points are $y_0 = y_1 = 1$, $y_2 = y_3 = 2$. We calculate

$$f[y_0, y_1] = f'(y_0) = 1 \qquad f[y_1, y_2] = 0.693147 \qquad f[y_2, y_3] = f'(y_2) = 0.5$$

$$f[y_0, y_1, y_2] = \frac{0.693147 - 1}{2 - 1} = -0.306853$$

$$f[y_1, y_2, y_3] = \frac{0.5 - 0.693147}{2 - 1} = -0.193147$$

$$f[y_0, y_1, y_2, y_3] = \frac{-0.193147 + 0.306853}{2 - 1} = 0.113706$$

The complete divided-difference table is written as follows:

y_i	$f[\ ]$	$f[,]$	$f[,,]$	$f[,,,]$
1	0.0			
		1.0		
1	0.0		−0.306853	
		0.693147		0.113706
2	0.693147		−0.193147	
		0.5		
2	0.693147			

With this

$$p_3(x) = 0. + (1.)(x - 1) + (-0.306853)(x - 1)^2 + (0.113706)(x - 1)^2(x - 2)$$

is the cubic polynomial which agrees with $\ln x$ in value *and* slope at the two points $x = 1$ and $x = 2$. The osculatory character of the approximation of $\ln x$ by $p_3(x)$ is evident from Fig. 2.7. Using Algorithm 2.1 to evaluate $p_3(x)$ at 1.5, we get

$$\ln 1.5 \approx p_3(1.5) = 0.409074$$

With $e_3(x) = f(x) - p_3(x)$ the error, we get from (2.37) and Theorem 2.5(i) the estimate

$$|e_3(1.5)| \le \frac{1}{4!} \max |f^{(4)}(\xi)|(1.5 - 1)^2(1.5 - 2)^2 = 0.015625$$

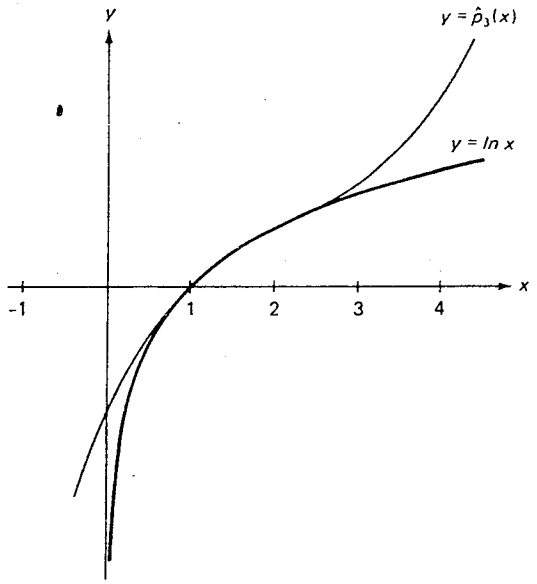


Figure 2.7 Osculatory interpolation.

Since ln 1.5 = 0.405465, the error is actually only 0.00361. This shows once again that the uncertainty about the location of ξ makes error estimates based on (2.18) rather conservative—to put it nicely.

We conclude this section with a FORTRAN program which calculates the coefficients for the Newton form of $p_n(x)$ and then evaluates $p_n(x)$ at a given set of equally spaced points.

```
C  CONSTRUCTION OF THE NEWTON FORM FOR THE POLYNOMIAL OF DEGREE
C  .LE. N , WHICH AGREES WITH  F(X)  AT  Y(I), I=1,...,NP1.
C  SOME OR ALL OF THE INTERPOLATION POINTS MAY COINCIDE, SUBJECT
C  ONLY TO THE FOLLOWING RESTRICTIONS.
C  (1) IF  Y(I) = Y(I+K),  THEN  Y(I) = Y(I+1) = ... = Y(I+K) .
C  (2) IF ALSO  Y(I-1) .NE. Y(I) , OR IF   I = 1 , THEN
C      F(I+J) = VALUE OF J-TH DERIVATIVE OF  F(X)  AT  X = Y(J),
C                                                    J=0,...,K.
C
      INTEGER I,J,K,N,NPOINT,NP1
      REAL DX,DY,F(30),FLAST,PNOFX,REALK,X,Y(30)
      READ 500,NP1,(Y(I),F(I),I=1,NP1)
  500 FORMAT(I2/(2F10.3))
C               CONSTRUCT DIVIDED DIFFERENCES
      N = NP1 - 1
      DO 10 K=1,N
         REALK = K
         FLAST = F(1)
         DO 9 I=1,NP1-K
            DY = Y(I+K) - Y(I)
            IF (DY .EQ. 0.) THEN
               F(I) = F(I+1)/REALK
            ELSE
               F(I) = (F(I+1) - FLAST)/DY
               FLAST = F(I+1)
            END IF
    9    CONTINUE
```



```
         F(NP1-K+1) = FLAST
   10 CONTINUE
C           CALCULATE PN(X) FOR VARIOUS VALUES OF X.
      READ 501,NPOINT,X,DX
  501 FORMAT(I3/2F10.3)
      DO 30 J=1,NPOINT
         PNOFX = F(1)
         DO 29 I=2,NP1
            PNOFX = F(I) + (X - Y(I))*PNOFX
   29    CONTINUE
         PRINT 629,J,X,PNOFX
  629    FORMAT(I10,2E20.7)
         X = X + DX
   30 CONTINUE
                                          STOP
      END
```

The calculation of divided differences corresponds to Algorithm 2.3 if all interpolation points are distinct. If some interpolation points coincide, the input must contain values of derivatives of the interpolant. Specifically, the input is assumed to consist of the array of interpolation points Y(I), I = 1, . . . , NP1 = n + 1, together with an array of numbers F(I), I = 1, . . . , NP1. For simplicity of programming, the sequence of interpolation points is assumed to satisfy the restriction that

$$\text{if } Y(I) = Y(I + K), \text{ then } Y(I) = Y(I + 1) = \cdots = Y(I + K)$$

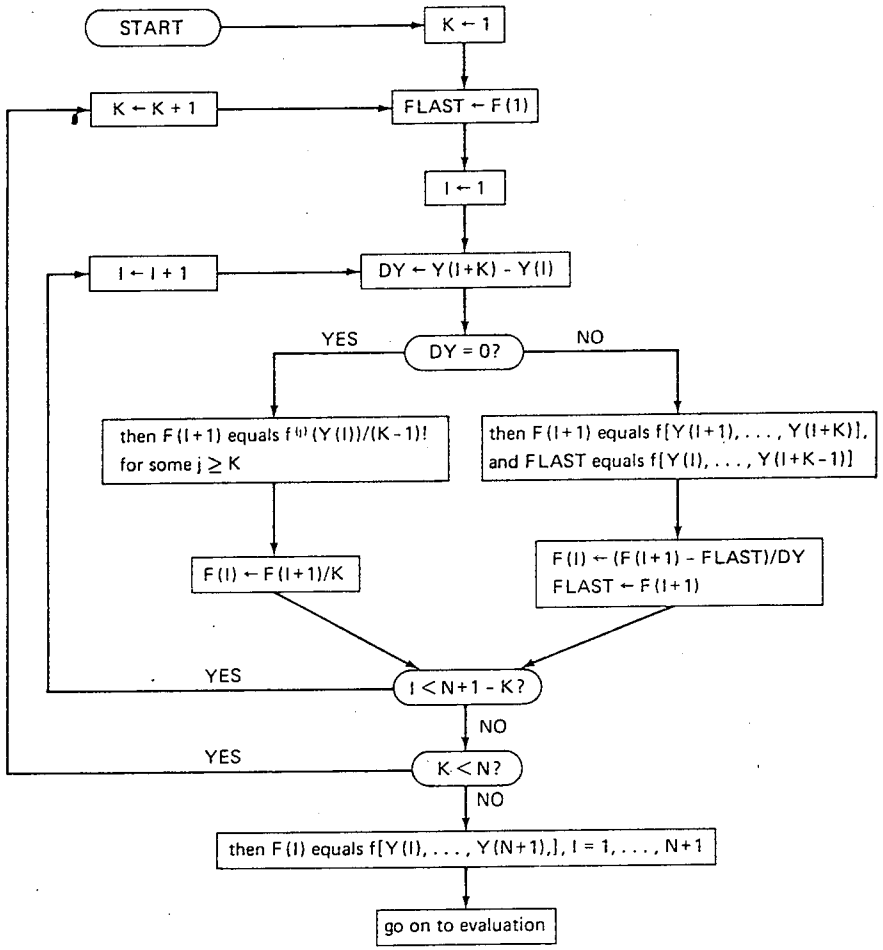
i.e., all repeated interpolation points appear together. With this restriction, it is further assumed that, for each I,

$$F(I) = f^{(j)}(Y(I)) \quad \text{if} \quad Y(I) = Y(I - j) \neq Y(I - j - 1)$$

Thus, with $f(x) = 1/x$, $n = 6$, the following input would be correct, in the sense that it would produce the polynomial of degree ≤ 6, which interpolates $f(x) = 1/x$ at the given Y(I), I = 1, . . . , 7.

I	1	2	3	4	5	6	7
Y(I)	2.	2.	2.	1.	4.	4.	5.
F(I)	0.5	−0.25	0.25	1.	0.25	−0.0625	0.2
	$\frac{1}{x}$	$f'(x) = -\frac{1}{x^2}$	$f''(x) = \frac{2}{x^3}$	$f(x)$	$f'(x) = -\frac{1}{x^2}$		

The student is encouraged to take an example like this and trace through the calculations in the FORTRAN program. The following flow chart describing the calculations of the divided differences might help in this endeavor.



EXERCISES

2.7-1 For $f(x) = e^x$ calculate $f(0.5)$, using quadratic interpolation, given that $f(0) = 1, f'(0) = 1, f(1) = 2.7183$. Compare with the correctly rounded result $f(0.5) = 1.6487$.

2.7-2 For $f(x) = \sinh x$ we are given that

$$f(0) = 0 \qquad f'(0) = 1 \qquad f(1) = 1.1752 \qquad f'(1) = 1.5431$$

Form a divided-difference table and calculate $f(0.5)$ using cubic interpolation. Compare the result with $\sinh 0.5 = 0.5211$.

2.7-3 A function $f(x)$ has a double zero at z_1 and a triple zero at z_2. Determine the form of the polynomial of degree ≤ 5 which interpolates $f(x)$ twice at z_1, three times at z_2, and once at some point z_3.

2.7-4 Find the coefficients a_0, a_1, a_2, a_3 for the cubic polynomial $p_3(x) = a_0 + a_1(x - y) + a_2(x - y)^2 + a_3(x - y)^3$, so that

$$p_3(y) = f_y \qquad p_3'(y) = f_y' \qquad p_3(z) = f_z \qquad p_3'(z) = f_z'$$

where $y, z, f_y, f_y', f_z, f_z'$ are given numbers (and $y \neq z$).

2.7-5 Get a simple expression for $p_3[(y + z)/2]$ in terms of the given numbers $y, z, f_y, f_y', f_z, f_z'$, where $p_3(x)$ is the polynomial determined in Exercise 2.7-4.

2.7-6 Let $f(x)$ and $g(x)$ be smooth functions. Prove that $f(x)$ agrees with $g(x)$ k-fold at the point $x = c$ if and only if $f(x) = g(x) + \mathcal{O}(|x - c|^k)$ for x near c.

2.7-7 Let $g(x) = f[x_0, \ldots, x_k, x]$. Prove that

$$g[y_0, \ldots, y_n] = f[x_0, \ldots, x_k, y_0, \ldots, y_n]$$

(use induction).

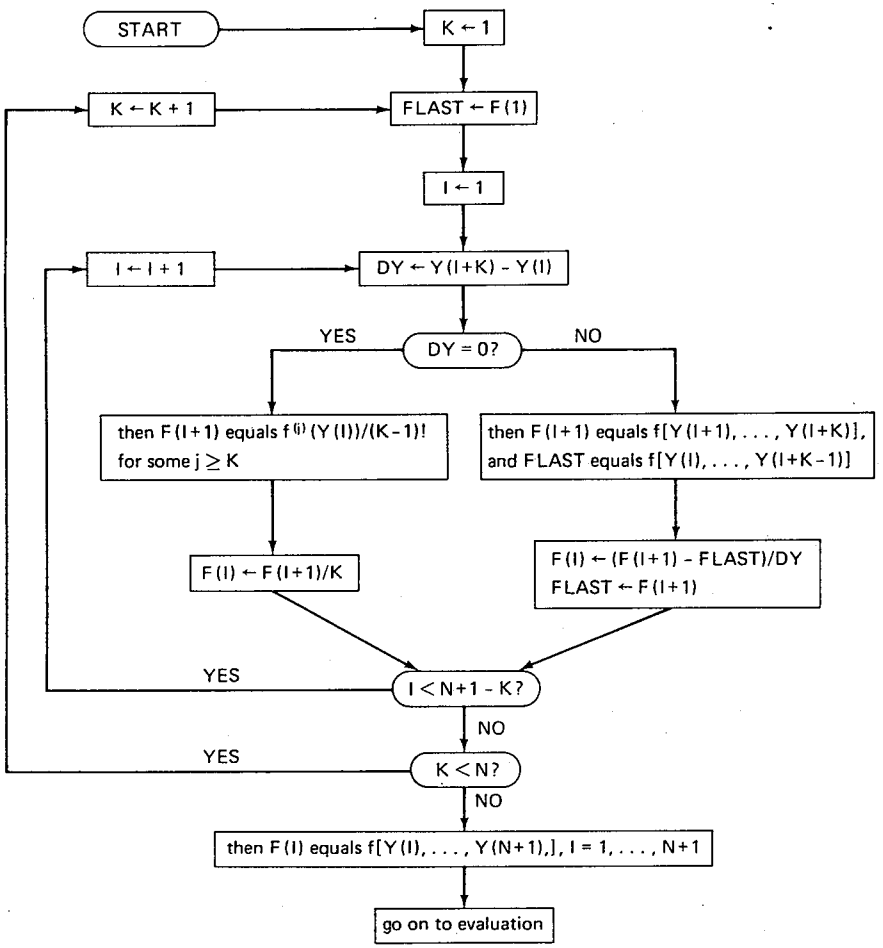
2.7-8 Use Exercise 2.7-7 to prove that if $g(x) = f[x_0, \ldots, x_k, x]$, then

$$g^{(n)}(x) = n! f[x_0, \ldots, x_k, \overbrace{x, \ldots, x}^{n+1 \text{ times}}]$$

2.7-9 Let $f(x) = g(x)h(x)$. Prove that

$$f[x_0, \ldots, x_k] = \sum_{i=0}^{k} g[x_0, \ldots, x_i]h[x_i, \ldots, x_k]$$

(use induction; else identify the right side as the leading coefficient of a polynomial of degree $\leq k$ which interpolates $g(x)h(x)$ at $x_0, \ldots, x_k$). What well known calculus formula do you obtain from this in case $x_0 = \cdots = x_k$?



EXERCISES

2.7-1 For $f(x) = e^x$ calculate $f(0.5)$, using quadratic interpolation, given that $f(0) = 1, f'(0) = 1, f(1) = 2.7183$. Compare with the correctly rounded result $f(0.5) = 1.6487$.

2.7-2 For $f(x) = \sinh x$ we are given that

$$f(0) = 0 \qquad f'(0) = 1 \qquad f(1) = 1.1752 \qquad f'(1) = 1.5431$$

Form a divided-difference table and calculate $f(0.5)$ using cubic interpolation. Compare the result with $\sinh 0.5 = 0.5211$.

2.7-3 A function $f(x)$ has a double zero at z_1 and a triple zero at z_2. Determine the form of the polynomial of degree ≤ 5 which interpolates $f(x)$ twice at z_1, three times at z_2, and once at some point z_3.

2.7-4 Find the coefficients a_0, a_1, a_2, a_3 for the cubic polynomial $p_3(x) = a_0 + a_1(x - y) + a_2(x - y)^2 + a_3(x - y)^3$, so that

$$p_3(y) = f_y \qquad p_3'(y) = f_y' \qquad p_3(z) = f_z \qquad p_3'(z) = f_z'$$

where $y, z, f_y, f_y', f_z, f_z'$ are given numbers (and $y \neq z$).

2.7-5 Get a simple expression for $p_3[(y + z)/2]$ in terms of the given numbers $y, z, f_y, f_y', f_z, f_z'$, where $p_3(x)$ is the polynomial determined in Exercise 2.7-4.

2.7-6 Let $f(x)$ and $g(x)$ be smooth functions. Prove that $f(x)$ agrees with $g(x)$ k-fold at the point $x = c$ if and only if $f(x) = g(x) + \mathcal{O}(|x - c|^k)$ for x near c.

2.7-7 Let $g(x) = f[x_0, \ldots, x_k, x]$. Prove that

$$g[y_0, \ldots, y_n] = f[x_0, \ldots, x_k, y_0, \ldots, y_n]$$

(use induction).

2.7-8 Use Exercise 2.7-7 to prove that if $g(x) = f[x_0, \ldots, x_k, x]$, then

$$g^{(n)}(x) = n! f[x_0, \ldots, x_k, \overbrace{x, \ldots, x}^{n+1 \text{ times}}]$$

2.7-9 Let $f(x) = g(x)h(x)$. Prove that

$$f[x_0, \ldots, x_k] = \sum_{i=0}^{k} g[x_0, \ldots, x_i]h[x_i, \ldots, x_k]$$

(use induction; else identify the right side as the leading coefficient of a polynomial of degree $\leq k$ which interpolates $g(x)h(x)$ at $x_0, \ldots, x_k$). What well known calculus formula do you obtain from this in case $x_0 = \cdots = x_k$?

An Intro to Numerical Computations

Yakowitz & Szidarovszky

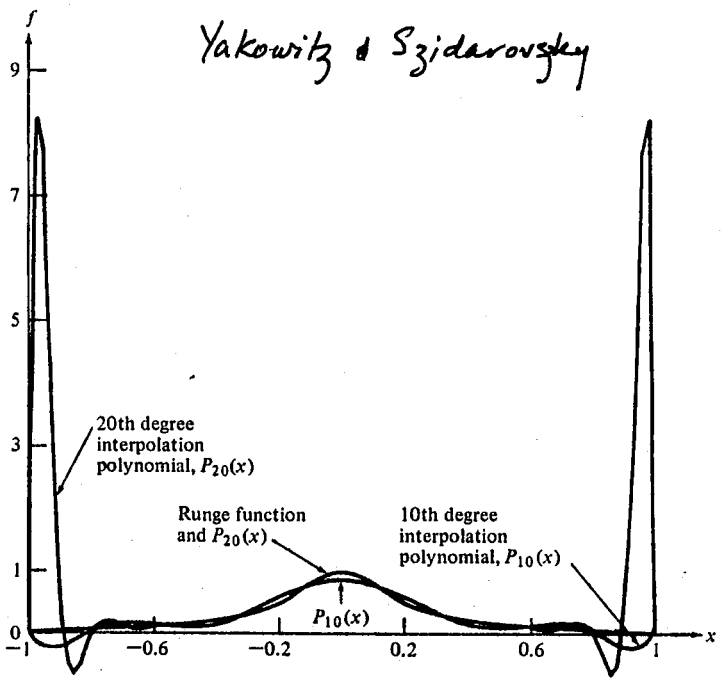


FIGURE 2.6 Interpolation Polynomial Approximation of the Runge Function

2.4.4 Refinements and Limitations of Polynomial Interpolation

One might hope that if a function $f(x)$ is continuous and "well behaved" on a given finite interval $[a, b]$, then as the number of interpolation points x_k increases and becomes dense on that interval, the associated polynomial sequence $\{p_{n-1}(x)\}$ converges to $f(x)$ at each point x in $[a, b]$. In Example 2.5, with the x_k's evenly spaced on $[-1, 1]$ and $f(x)$ not only continuous but differentiable, we see an approximation actually get worse.

EXAMPLE 2.5

Let $f(x) = 1/(1 + 25x^2)$, where $x \in [-1, 1]$. This function is called the *Runge function*. Figure 2.6 shows this function and its interpolation polynomials of degrees 10 and 20. It is known (Isaacson and Keller, 1966, pp. 275–278) that the sequence $\{p_{n-1}(x)\}$ of these interpolation polynomials with equally spaced interpolation points does not converge to $f(x)$ at any x in the intervals $[-1, -0.727]$ or $[0.727, 1]$. Note that the Runge function is fairly well represented by interpolation polynomials over the central 72.7% of the interval, but the approximation is very bad near the extremes. ■

The power of mathematical reasoning is most astonishing in cases in which it directs us to unintuitive conclusions. It would seem "reasonable" that equally spaced interpolation points x_k (i.e., $x_{k+1} - x_k$ the same for all k) ought to be best in the sense that they provide the most representative $f(x_k)$ samples. Thus it might

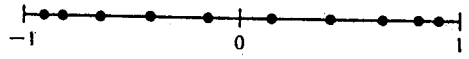


FIGURE 2.7 Distribution of Chebyshev Points

be expected that these ought to be the most logical points for constructing an interpolation polynomial. This line of reasoning is incorrect.

Let us recall from Section 2.4.3 that the truncation error for approximating an n-times differentiable function by an interpolation polynomial $p_{n-1}(x)$ with interpolation points $x_1, \ldots, x_n$ is

$$f(x) - p_{n-1}(x) = \frac{f^{(n)}(\zeta)w_n(x)}{n!}. \tag{2.16}$$

In (2.16), ζ is some number in the interval containing all interpolation points as well as x, and

$$w_n(x) = (x - x_1)(x - x_2) \cdots (x - x_n).$$

The term $f^{(n)}(\zeta)$ above is beyond our control. However, the term $w_n(x)$ offers possibilities if we are free to choose the interpolation points. A sensible ambition might then be to choose these points so that the maximum of $|w_n(x)|$ is as small as possible over the interval $[a, b]$ of interpolation. It turns out (Szidarovszky and Yakowitz, 1978, Sec. 2.2) that this ambition is realized by choosing the so-called Chebyshev points as the interpolation points. For the interval $[a, b]$, the *Chebyshev points* are defined to be

$$x_k = \frac{a + b}{2} + \frac{a - b}{2} \cos\left(\frac{2k - 1}{2n}\pi\right) \qquad (k = 1, \ldots, n). \tag{2.17}$$

In Figure 2.7 we have located the Chebyshev points for $n = 10$ on the interval $[-1, 1]$. Figure 2.8 compares $w_{10}(x)$ when x_i's are evenly spaced against $w_{10}(x)$ for x_i's the Chebyshev points.

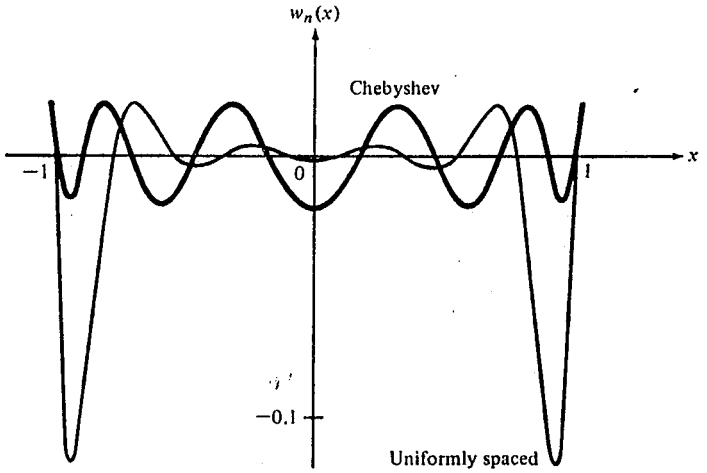


FIGURE 2.8 Comparison of $w_n(x)$ Functions

TABLE 2.9 Interpolation of Runge Function Using Chebyshev Points

```
C     PROGRAM RUNGE
C
C     ************************************************************
C     COMPUTES 30TH DEGREE INTERPOLATION POLYNOMIAL USING
C     CHEBYSHEV POINTS FOR THE RUNGE FUNCTION
C     CALLS:  LAGR
C     OUTPUT: T=THE VALUE OF THE INDEPENDENT VARIABLE
C             TRUE=THE FUNCTIONAL VALUE AT THE POINT T AS
C                  DETERMINED BY THE STATEMENT FUNCTION FF
C             P=THE VALUE OF THE INTERPOLATION POLYNOMIAL AT T
C             E=THE ERROR OF THE APPROXIMATION AT T
C     ************************************************************
C
      DIMENSION X(40),F(40)
C
C     *** RUNGE FUNCTION STATEMENT                              ***
C
      FF(X)=1.0/(1.0+(5.0*X)**2)
C
C     *** COMPUTE VALUES AT CHEBYSHEV POINTS                    ***
C
      N=31
      PI=4.0*ATAN(1.0)
      DO 10 I=1,N
        X(I)=COS((2.0*I-1)*PI/(2.0*N))
        F(I)=FF(X(I))
   10 CONTINUE
C
C     *** SUBROUTINE LAGR WILL COMPUTE VALUE OF THE             ***
C     *** INTERPOLATION POLYNOMIAL AT T AND E GIVES THE ERROR   ***
C
      DO 30 I=1,20
        T=-1.0+I*0.1
        CALL LAGR(N,X,F,T,P)
        TRUE=FF(T)
        E=TRUE-P
        WRITE(10,2)T,TRUE,P,E
    2   FORMAT(2X,F3.1,2X,E14.7,2X,E14.7,2X,E14.7)
   30 CONTINUE
      STOP
      END
```

EXAMPLE 2.6

By means of the simple calling program in Table 2.9 we have constructed the 31-point Chebyshev-interpolation polynomial for the Runge function (described in Example 2.5). In the output listing (Table 2.10), the interpolation error is tabulated at 20 evenly spaced values. We see that the Runge function is apparently under control.

TABLE 2.10 Output of Runge Function Interpolation

x	*Runge Function*	*Approximation*	*Error*
-.9	0.4705882E-01	0.4692032E-01	0.1385026E-03
-.8	0.5882353E-01	0.5837149E-01	0.4520416E-03
-.7	0.7547170E-01	0.7447892E-01	0.9927750E-03
-.6	0.9999998E-01	0.1011296E+00	-0.1129620E-02
-.5	0.1379310E+00	0.1372023E+00	0.7287264E-03

TABLE 2.10 (Continued)

x	*Runge Function*	*Approximation*	*Error*
-.4	0.2000000E+00	0.2003204E+00	-0.3204048E-03
-.3	0.3076923E+00	0.3076517E+00	0.4056096E-04
-.2	0.5000001E+00	0.4999129E+00	0.8720160E-04
-.1	0.8000001E+00	0.8000615E+00	-0.6145239E-04
0.0	0.1000000E+01	0.1000000E+01	-0.1192093E-06
0.1	0.8000000E+00	0.8000612E+00	-0.6127357E-04
0.2	0.4999999E+00	0.4999129E+00	0.8702278E-04
0.3	0.3076922E+00	0.3076518E+00	0.4041195E-04
0.4	0.2000000E+00	0.2003204E+00	-0.3203750E-03
0.5	0.1379310E+00	0.1372023E+00	0.7287711E-03
0.6	0.9999998E-01	0.1011297E+00	-0.1129672E-02
0.7	0.7547168E-01	0.7447891E-01	0.9927750E-03
0.8	0.5882351E-01	0.5837147E-01	0.4520416E-03
0.9	0.4705882E-01	0.4692034E-01	0.1384839E-03
1.0	0.3846154E-01	0.3764865E-01	0.8128881E-03

Have we mastered the approximation problem through interpolation using Chebyshev points? In some cases "yes" and in other cases "no." From de Boor (1978), for example, the following facts are known:

1. If (as in the case of the Runge function) $f(x)$ has a *continuous derivative* on $[a, b]$, then using Chebyshev points, for each x in $[a, b]$,
$$p_{n-1}(x) \to f(x), \qquad \text{as } n \to \infty.$$
2. No matter how interpolation points are chosen, there is some continuous function $f(x)$ for which $\{p_{n-1}(x)\}$ fails to converge, regardless of x.
3. Even when $f(x)$ has a continuous derivative, adequate approximation may require an impossibly large n.

Point 3 is illustrated by our printout in Table 2.10, where a 30th-degree polynomial is scarcely accurate to three significant decimals. In this regard we note that the degree must be over 1 million for $\sqrt{|x|}$ to be interpolated on $[-1, 1]$ to an accuracy of 10^{-3} (de Boor, 1978). Interpolation polynomials of degree 30 are on the borderline of the impractical, and degrees greater than 100 lead almost invariably to grievous roundoff effects.

*2.4.5 The Newton Representation

From developments in Section 2.4.2, we know that all interpolation polynomials having degree not exceeding $n - 1$, n being the number of points to be interpolated, are identical to the Lagrange representation (2.11). But other representations sometimes serve useful purposes. For example, the *Newton representation* of the interpolating polynomial has the feature that it is easier to update—that is, to include an additional point (x_{n+1}, f_{n+1}). Also, if coded wisely, it requires only about half as many arithmetic operations per call as the Lagrange form and is less subject to deterioration due to roundoff effects.

Then (2.34) implies that

$$\int_a^b [f''(x)]^2\,dx = \int_a^b [g''(x)]^2\,dx + \int_a^b [s''(x)]^2\,dx \geqq \int_a^b [s''(x)]^2\,dx,$$

which completes the proof.

Experimentally, we have concluded that the spline approach does not fully resolve some approximation issues. To be a candidate for a computer library function, for example, we would anticipate that an algorithm ought to provide an answer with an accuracy of at least 7 to 10 significant decimals after very few arithmetic computations. Sometimes interpolation polynomials fill this need very nicely. We found that by using Chebyshev points, we could interpolate sin (x) on [0, 1] to about 27 significant decimals with a 21st-order interpolation polynomial. The natural spline, using the same 21 data points, gave only about three places of accuracy, and even when we raised the number of data points n to 90, the interpolations were accurate to only about five significant places.

Our conclusion is that spline approximation is perfectly adequate if relative error on the order of 1% is acceptable. On the other hand, if library-function accuracy is required, alternative methods or piecewise polynomials more sophisticated than cubic splines will be needed, unless the intervals are small.

There are undeniably some loose ends to our attempted mastery of the function approximation problem. For example, it is known that there is no polynomial of degree less than 1 million, interpolating or otherwise, which approximates $\sqrt{|x|}$ to four significant places at all points on the interval $[-1, 1]$. In fact, experiments have convinced the authors that interpolation of this function on [0, 1] by either splines or polynomials cannot feasibly be done accurately. Fortunately, we will see that by methods in Chapter 5 direct computation of $\sqrt{|x|}$ can be done quickly, accurately, and simply. One realm in which piecewise polynomials and splines appear to reign supreme is in approximation of "natural" and "nonmathematical" shapes.

2.6 SUPPLEMENTARY NOTES AND DISCUSSIONS

We have considered three basic techniques—Taylor's polynomials, interpolation polynomials, and splines—for approximating functions and "smooth" data. The background developments for Taylor's and interpolation polynomials will serve us well in describing numerical methods for other problem areas. Of these three procedures, Taylor's polynomials have been the least widely used in actual numerical computation (but most widely used in mathematical analysis of numerical algorithms). Our first thought was that the lack of enthusiasm stemmed from the requirement of higher derivatives in (2.6). Higher derivatives of functions other than polynomials are typically at least a nuisance to obtain, and frequently a rich source of error because of calculus mistakes.

Using numerical differentiation ideas such as those offered in Chapter 3, one can in many cases obtain adequate approximations to Taylor's polynomials. Moreover, algebraic computer languages (e.g., MACSYMA and MU-MATH), which

automate differentiation, are becoming available. Another drawback then comes to the fore. In contrast to interpolation polynomials, Taylor's polynomial approximation tends to be much more accurate than needed in the vicinity of the expansion point x_0, and defective for large values of $|x - x_0|$. Interpolation error, while also tending to be larger near the endpoints of the interpolation interval, does spread the error out more evenly (particularly if Chebyshev interpolation points are used). In Example 2.4 (see Figure 2.5), this more even distribution of approximation error is clearly in evidence.

For "smooth" functions, interpolation polynomials are adequate for many needs. When the curve or data set has more erratic features, piecewise polynomial approximation and splines have some appealing advantages. Whereas our discussion centered on natural cubic splines, we notify the reader that in recent years, the tendency has been to keep all options open and choose the piecewise polynomial approach most suited to the application at hand. For instance, in interpolating solutions of differential equations, some investigators have found the higher-derivative requirement of the "spline" definition to be overly constricting. Even within the domain of cubic interpolation splines, "natural" splines such as we and most other numerical methods authors propose are known to have certain technical deficiencies. Nevertheless, they do guarantee convergence to any continuous limiting function, as the number of node points x_i become dense, a property that we have noted is not shared by interpolating polynomials.

The techniques given in this chapter are adequate for obtaining computer approximations of common functions, and for transcribing "natural" curves, such as demand/sales curves or transistor operating characteristic curves, into a computer-manageable form. Also, they are useful for "rough and ready" computer graphics displays.

We are hasty to point out, however, that program functions, such as in computer libraries or commercial packages, employ much more sophisticated principles than can be related at the level of this textbook. The subject of computer approximation of functions is a lively and sophisticated research area. Algorithms employing advanced theory have greater accuracy, for a given computational effort, than the prototypical techniques related here. On the other hand, the advanced methods

TABLE 2.16 Match of Interpolation Problem Characteristics and Methods

Function Characteristics	*Suggested Method**
Few Data Points (<20)	
Derivatives not large	I, S
Large derivatives	S
User can choose data locations x_i, function fairly smooth	IC
Many Data Points	
Data due to "natural" (i.e., nonmathematical) origin	S
Function smooth, regular	IL
Large errors or random origins for data	Go to Chapter 6

*I, interpolation polynomials; IC, interpolation polynomial using Chebyshev points; S, splines; IL, interpolation polynomial using only a moderate number of local (neighboring) points, [i.e., if you wish to approximate at x, construct $p_{n-1}(x)$ on basis of a data pairs (x_i, f_i) such that the x_i's are as close to x as possible.

EGM 5346 - COMP. ENG. ANAL.

6/13/05

Chebyshev Points

$$x_i = \frac{a+b}{2} + \frac{a-b}{2}\cos\frac{2i-1}{2n}\pi \qquad i = 1, \ldots, n$$

Error in the interpolating polynomial is like the next term in the newton form

$$p_n(x) = \sum_{i=0}^{N} f[x_i, \ldots, x_n] \prod_{j=i+1}^{n} (x - x_j)$$

$$P_3 = f[x_1, x_2, x_3](x-x_1)(x-x_2)(x-3) + \cdots$$

$$\text{error} \sim \prod_{i=1}^{4} (x - x_i)$$

next term is closely related to $(n+1)^{ST}$ derivative

$$\text{error} \sim \frac{f^{IV}(\xi)}{4!}(x-c)^4$$

We can say that 2 functions $f(x)$ & $g(x)$ agree at points $x_0, x_1, \ldots, x_k$

if $$f^{(j)}(x) = g^{(j)}(x) \quad \text{for } j = 0, 1, \ldots, m-1$$

for every x which occurs m times in the sequence $x_0, x_1, \ldots, x_k$

Next, we prove (i). If $y_0 = y_1 = \cdots = y_n$, then (i) is just a restatement of (2.33). Otherwise, we may assume that

$$y_0 \le y_1 \le \cdots \le y_n$$

and then $y_0 < y_n$. But then we may find, for all r, $x_0^{(r)} < \cdots < x_n^{(r)}$ in $[a, b]$ so that $\lim_{r\to\infty} x_i^{(r)} = y_i$, $i = 0, \ldots, n$. By Theorem 2.2, we can find then $\xi^{(r)} \in [x_0^{(r)}, x_n^{(r)}]$ so that

$$f\left[x_0^{(r)}, \ldots, x_n^{(r)}\right] = f^{(n)}(\xi^{(r)})/n! \qquad \text{for } r = 1, 2, 3, \ldots$$

But then, by (ii) just proved for this case,

$$f\left[y_0, \ldots, y_n\right] = \lim_{r\to\infty} f\left[x_0^{(r)}, \ldots, x_n^{(r)}\right] = \lim_{r\to\infty} f^{(n)}(\xi^{(r)})/n!$$
$$= f^{(n)}(\xi)/n!$$

for some $\xi \in [\lim x_0^{(r)}, \lim x_n^{(r)}] = [y_0, y_n]$, by the continuity of $f^{(n)}(x)$, which proves (i).

Finally, to prove (ii) in the case that $y_0 = y_1 = \cdots = y_n$, we now use (i) to conclude the existence of $\xi^{(r)} \in [\min_i x_i^{(r)}, \max_i x_i^{(r)}]$ so that $f[x_0^{(r)}, \ldots, x_n^{(r)}] = f^{(n)}(\xi^{(r)})/n!$ for all r. But then, since $y_0 = \cdots = y_n$ and $\lim x_i^{(r)} = y_i$, all i, we have $\lim \xi^{(r)} = y_0$ and so, with (2.36) and the continuity of $f^{(n)}(x)$,

$$f\left[y_0, \ldots, y_n\right] = f^{(n)}(y_0)/n! = \lim f^{(n)}(\xi^{(r)})/n! = \lim f\left[x_0^{(r)}, \ldots, x_n^{(r)}\right]$$

This proves both (i) and (ii) for $n = k$ and for all choices of $y_0, \ldots, y_n$ in $[a, b]$.

We conclude this section with some interesting consequences of Theorem 2.5. It follows at once that the function

$$g_n(x) = f\left[x_0, \ldots, x_n, x\right]$$

which appears in the error term for polynomial interpolation is defined for all x and is a *continuous* function of x if $f(x)$ is sufficiently smooth. Thus it follows that

$$f(x) = \sum_{i=0}^{n} f\left[x_0, \ldots, x_i\right] \prod_{j=0}^{i-1} (x - x_j) + f\left[x_0, \ldots, x_n, x\right] \prod_{j=0}^{n} (x - x_j) \tag{2.37}$$

for all x, and not only for $x \neq x_0, \ldots, x_n$ [see (2.16)], and also for all $x_0, \ldots, x_n$, distinct or not, in case $f(x)$ has enough derivatives.

Further, if $f(x)$ is sufficiently often differentiable, then $g_n(x)$ is differentiable. For by the definition of derivatives,

$$g_n'(x) = \lim_{h\to 0} g_n[x, x + h]$$

if this limit exists. On the other hand,

by Theorem 2.5. Hence

$$\frac{d}{dx} f\left[x_0, \ldots, x_n, x\right] = f\left[x_0, \ldots, x_n, x, x\right] \tag{2.38}$$

Finally, it explains our definition of osculatory interpolation as *repeated* interpolation. For it shows that the interpolating polynomial at points $x_0, \ldots, x_n$ converges to the interpolating polynomial at points $y_0, \ldots, y_n$ as $x_i \to y_i$, all i. Thus, k-fold interpolation at a point is the limiting case as we let k distinct interpolation points coalesce. The student is familiar with this phenomenon in the case $n = 1$ of linear interpolation. In this case, the straight line $p_1(x) = f(x_0) + f[x_0, x_1](x - x_0)$ is a secant to (the graph of) $f(x)$ which goes over into the tangent $\hat{p}_1(x) = f(y) + (x - y)f'(y)$ as both x_0 and x_1 approach the point y, and $\hat{p}_1(x)$ agrees with $f(x)$ in value *and* slope at $x = y$.

Example 2.9 With $f(x) = \ln x$, calculate $f(1.5)$ by cubic interpolation, using $f(1) = 0$, $f(2) = 0.693147$, $f'(1) = 1$, $f'(2) = 0.5$.

In this case, the four interpolation points are $y_0 = y_1 = 1$, $y_2 = y_3 = 2$. We calculate

$$f[y_0, y_1] = f'(y_0) = 1 \qquad f[y_1, y_2] = 0.693147 \qquad f[y_2, y_3] = f'(y_2) = 0.5$$

$$f[y_0, y_1, y_2] = \frac{0.693147 - 1}{2 - 1} = -0.306853$$

$$f[y_1, y_2, y_3] = \frac{0.5 - 0.693147}{2 - 1} = -0.193147$$

$$f[y_0, y_1, y_2, y_3] = \frac{-0.193147 + 0.306853}{2 - 1} = 0.113706$$

The complete divided-difference table is written as follows:

y_i	$f[\ ]$	$f[,]$	$f[,,]$	$f[,,,]$
1	0.0			
		1.0		
1	0.0		−0.306853	
		0.693147		0.113706
2	0.693147		−0.193147	
		0.5		
2	0.693147			

With this

$$p_3(x) = 0. + (1.)(x - 1) + (-0.306853)(x - 1)^2 + (0.113706)(x - 1)^2(x - 2)$$

is the cubic polynomial which agrees with $\ln x$ in value *and* slope at the two points $x = 1$ and $x = 2$. The osculatory character of the approximation of $\ln x$ by $p_3(x)$ is evident from Fig. 2.7. Using Algorithm 2.1 to evaluate $p_3(x)$ at 1.5, we get

$$\ln 1.5 \approx p_3(1.5) = 0.409074$$

With $e_3(x) = f(x) - p_3(x)$ the error, we get from (2.37) and Theorem 2.5(i) the estimate

$$|e_3(1.5)| < \frac{1}{4!} \max |f^{(4)}(\xi)|(1.5 - 1)^2(1.5 - 2)^2 = 0.015625$$

then the interpolating polynomial has osculatory interpolation when it has higher than first order contact at the interpolation point.

Given x_0, x_1, x_1, x_2, x_3 & y_0, y_1, y_1', y_2, y_3

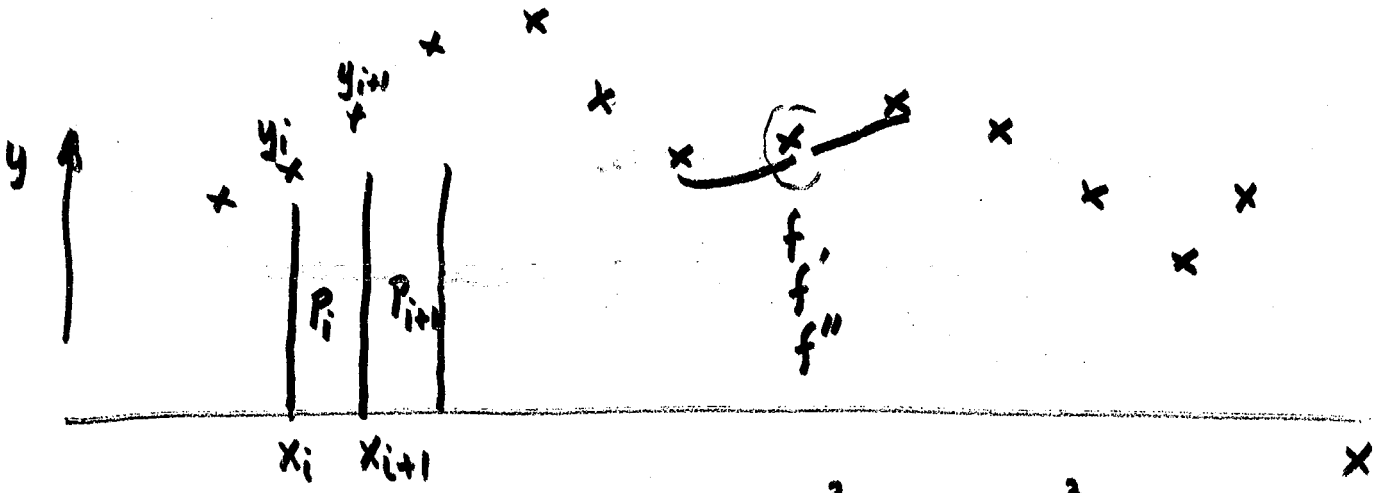
$x_0 \text{ — } y_0$

$x_1 = y_1$ > $\frac{y_1 - y_0}{x_1 - x_0}$

$x_1 \text{ — } y_1$ > y_1' > > >

$x_2 \quad y_2$ > > >

$x_3 \quad y_3$ >



$$P_i(x) = a + b(x - x_i) + c(x - x_i)^2 + d(x - x_i)^3$$

Hermite Polynomial

$P_i(x_i) = y_i = f(x_i)$ $\qquad$ $P_{i+1}(x_{i+1}) = f(x_{i+1})$

$P_i'(x_i) = y_i' = f'(x_i)$ $\qquad$ $P_{i+1}'(x_{i+1}) = f'(x_{i+1})$

$$P_i(x_{i+1}) = P_{i+1}(x_{i+1})$$

$$P_i'(x_{i+1}) = P_{i+1}'(x_{i+1})$$

x_i	$f(x_i)$			
		$f'(x_i)$		
x_i	$f(x_i)$		$f[x_i, x_i, x_{i+1}]$	
		$f[x_i, x_{i+1}]$		$f[x_i, x_i, x_{i+1}, x_{i+1}]$
x_{i+1}	$f(x_{i+1})$		$f[x_i, x_{i+1}, x_{i+1}]$	
		$f'(x_{i+1})$		
x_{i+1}	$f(x_{i+1})$			

$$P_3 = f(x_i) + f'(x_i)(x-x_i) + f[x_i, x_i, x_{i+1}](x-x_i)^2 + f[x_i, x_i, x_{i+1}, x_{i+1}] \cdot (x-x_i)^2(x-x_{i+1})$$

$$= a + b(x-x_i) + c(x-x_i)^2 + d(x-x_i)^3$$

$$(x-x_{i+1}) = (x-x_i) + (x_i - x_{i+1}) \qquad -\Delta x_i$$

$$c = f[x_i, x_i, x_{i+1}] - f[x_i, x_i, x_{i+1}, x_{i+1}]\Delta x_i$$

$$d = f[x_i, x_i, x_{i+1}, x_{i+1}] = \frac{f'(x_i) + f'(x_{i+1}) - 2f[x_i, x_{i+1}]}{\Delta x_i^2}$$

this satisfies $P_3(x_i) = f(x_i)$ $\quad P_3(x_{i+1}) = f(x_{i+1})$

$P_3'(x_i) = f'(x_i)$ $\quad P_3'(x_{i+1}) = f'(x_{i+1})$

Cube spline (more ambitious)

$P_i(x_{i+1}) = P_{i+1}(x_{i+1})$

$P_i'(x_{i+1}) = P_{i+1}'(x_{i+1})$

$P_i''(x_{i+1}) = P_{i+1}''(x_{i+1})$

$P_i''(x_{i+1}) = 2c + 6d(x_{i+1} - x_i)$

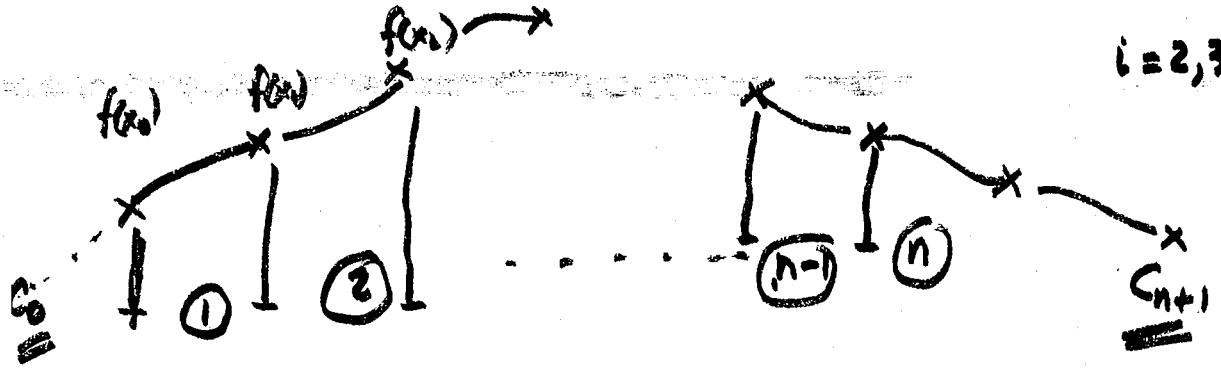
$$P_{i+1} = a + b(x - x_{i+1}) + c(x - x_{i+1})^2 + d(x - x_{i+1})^3$$

$$P''_{i+1} = 2c + 6d(x - x_{i+1}) = 2c \quad \text{when } x = x_{i+1}$$

$$2c_i + 6d_i \Delta x_i = 2c_{i+1}$$

$$\Rightarrow \Delta x_{i-1}\, c_{i-1} + 2(\Delta x_{i-1} + \Delta x_i)c_i + \Delta x_i\, c_{i+1} = 6\frac{f(x_{i+1}) - f(x_i)}{\Delta x_i} - 6\frac{f(x_i) - f(x_{i-1})}{\Delta x_{i-1}}$$

$i = 2, 3, \ldots, n$



$$\begin{bmatrix} 2(\Delta x_0 + \Delta x_1) & \Delta x_1 & & & \\ \Delta x_1 & 2(\Delta x_1 + \Delta x_2) & \Delta x_2 & & 0 \\ & \ddots & \ddots & \ddots & \\ 0 & & \ddots & \ddots & \\ & & & & \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 6\frac{f(x_2) - f(x_1)}{\Delta x_1} - 6\frac{f(x_1) - f(x_0)}{\Delta x_0} \\ \vdots \\ \vdots \end{bmatrix}$$

Case 1: Natural Spline c_0 & c_{n+1} are chosen are zero.

Case 2: $c_{n+1} = c_n$ & $c_0 = c_1$

Case 3: Choose c_0 & c_{n+1} to be obtained by linear interpolation

$$c_0 = \frac{c_2 - c_1}{(\Delta x_2 - \Delta x_1)} + c_1 \qquad c_{n+1} = \frac{c_n - c_{n-1}}{(\Delta x_{n+1} - \Delta x_n)} + c_n$$

if c_0 & c_{n+1} are given then use them in the equations

FOR CUBIC SPLINE

$$P_{3_i}(x) = a_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3$$

$$a_i = f(x_i)$$

$$b_i = \frac{f(x_{i+1}) - f(x_i)}{\Delta x_i} - \frac{\Delta x_i}{3}(2c_i + c_{i+1})$$

c_i's are solved for by solving the matrix equations

$$d_i = \frac{c_{i+1} - c_i}{3\Delta x_i}$$

▶2. Show that the point whose x-coordinate is the mean of all the x-values and whose y-coordinate is the mean of all the y-values satisfies the least-squares line. Often a change of variable is made to relocate the origin at this point, with a corresponding reduction in the magnitude of the numbers worked with, making them more readily handled by hand or on some desk calculators.

▶3. Find the least-squares line that fits to the following data, assuming the x-values are free of error:

x	y	x	y
1	2.04	4	7.18
2	4.12	5	9.20
3	5.64	6	12.04

(The data are tabulated from $y = 2x$, with perturbations from a table of random numbers.)

4. In the data for Exercise 3, consider that the y-values are free of error and that all the errors are in the x-values. By suitable modifications of the normal equations, now determine the least-squares line for $x = ay + b$. Note that this is not the same line as determined in Exercise 3.

5. In multivariate analysis, the least-squares technique is used to determine the hyperplane from which the sum of the squares of the deviations is a minimum. For z, a function of two independent variables, x and y, find the normal equations to determine the parameters a, b, and c in:

$$z = ax + by + c.$$

Use this relation to find the least-squares values for the constants if the data below are available.

x	0	1.2	2.1	3.4	4.0	4.2	5.6	5.8	6.9
y	0	0.5	6.0	0.5	5.1	3.2	1.3	7.4	10.2
z	1.2	3.4	−4.6	9.9	2.4	7.2	14.3	3.5	1.3

Section 10.2

6. Observe that the data in Table 10.8 seem to be fit by a curve $y = ae^{bx}$ by plotting on semi-log paper and noting that the points then fall near a straight line. The data are for the solubilities of n-butane in anhydrous hydrofluoric acid at high pressures, and were needed in the design of petroleum refineries.

TABLE 10.8

Temperature, °F	Solubility, weight percent
77	2.4
100	3.4
185	7.0
239	11.1
285	19.6

By plotting on rectilinear graph paper, observe that the relationship is nonlinear.

▶7. Determine the constants for $y = ae^{bx}$ for the data in Exercise 6 by the least-squares method by fitting to the relation $\ell n\, y = \ell n\, a + bx$.

8. It is suspected (from theoretical considerations) that the rate of flow from a fire hose is proportional to some power of the pressure at the nozzle. Determine whether the speculation seems to be true, and what the exponent is from the data in Table 10.9. (Assume that the pressure data are more accurate.)

TABLE 10.9

Flow, gallons per minute	Pressure, psi
94	10
118	16
147	25
180	40
230	60

9. Since the data of Exercise 8, when plotted on log-log paper (pressure as a function of flow), seem to have a slope of nearly 2, we should expect that fitting a quadratic to the data would be successful. Do this and compare the deviations with those from the power-function relation of Exercise 8.

▶10. The data given in Exercise 3, while perturbations from a linear relation, seem to plot better along a curve because of the accidental occurrence of three negative deviations in succession. Fit a quadratic to the data.

11. To compare the results of an exact polynomial fit according to Chapter 3 with the least-squares procedure, find y-values at $x = 1.5$, 2.5, 3.5, 4.5, and 5.5 for the data of Exercise 3, utilizing a fifth-degree interpolating polynomial. Sketch the interpolating polynomial and compare to the least-squares line of Exercise 3, and the least-squares quadratic of Exercise 10. The slope of the "true" function is 2.0. How do the maximum and minimum values of the slope as determined from the three approximations (use a graphical procedure) compare to the "true" value?

Intro to Numerical Computations by Yakowitz & Szidarovszky

*7.7
ADAPTIVE STEP-SIZE SELECTION AND ERROR CONTROL

Up to this point we have not discussed how the step size h of the preceding methods is to be chosen. Obviously, there is a trade-off to be made: If the step size is too small, then computer time is needlessly wasted and accumulation of arithmetic roundoff errors can become a hazard. A large step size invites large truncation error associated with higher-order terms neglected in the construction of the methods. For simplicity, our developments will be concerned only with Runge–Kutta rules.

Techniques for automatic step-size selection are based on estimating the local error at each step and then choosing the step size to keep this estimated error within some tolerance bound. Thus step-size selection hinges on estimation of the *local error*, which at the jth step is defined to be

$$\hat{y}(x_{j+1}) - y_{j+1}.$$

Here y_{j+1} is, of course, the computed approximation of $y(x_{j+1})$, and $\hat{y}(x_{j+1})$ we define to be the exact value at x_{j+1} of the differential equation solution that passes through the point (x_j, y_j). That is, $\hat{y}(x)$ solves the initial-value problem

$$\hat{y}' = f(x, \hat{y}), \qquad \hat{y}(x_j) = y_j.$$

In contrast to local errors, the *global error* at x_{j-1} is defined to be

$$y(x_{j-1}) - y_{j-1},$$

where $y(x)$ is the exact solution of the original initial-value problem (7.3). Figure 7.7 illustrates the relationships between $y(x)$, $\hat{y}(x)$, and local and global errors. Intuitively, the local error is the additional truncation error arising from inexact solution at a given step. The global error gives the accumulated total error propagating from the entire sequence of steps.

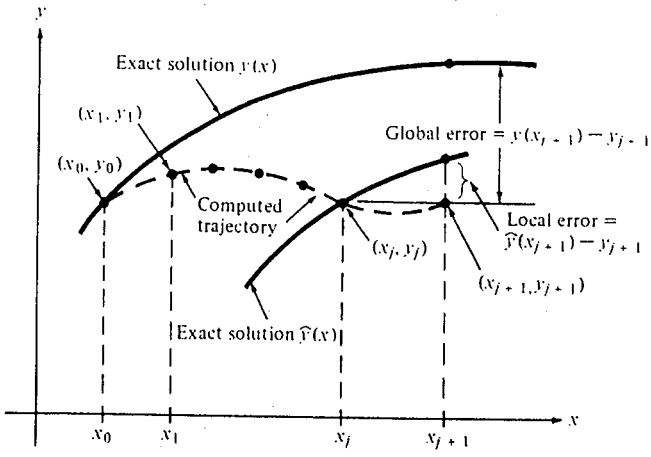


FIGURE 7–7 Relationship Between $y(x)$, $\hat{y}(x)$, Local and Global Errors

Assume that some Runge–Kutta procedure has been selected. We let y_0, y_1, y_2, . . . denote the computed solution values at the arguments x_0, x_1, x_2, ... The local error estimation techniques at each stage apply a higher-order technique to compute an additional approximation, say z_{j+1}, of $y(x_{j+1})$. Since a higher-order technique is used, if the solution is "well behaved" and the step size h is small enough that neglected terms really are negligible, then one may anticipate that the local error of the higher-order method is much less than that of the selected Runge–Kutta procedure. That is,

$$|\hat{y}(x_{j+1}) - z_{j+1}| << |\hat{y}(x_{j+1}) - y_{j+1}|. \tag{7.51}$$

If the approximation above indeed holds, then

$$z_{j+1} - y_{j+1} \simeq \hat{y}(x_{j+1}) - y_{j+1}. \tag{7.52}$$

and we take $z_{j+1} - y_{j+1}$ as the estimate of local error.

Of course, computation of z_{j+1} is typically more expensive than that of y_{j+1} itself, since z_{j+1} must be more accurate. Here, as in other walks of life, information must be paid for. A popular idea toward making this expense as small as possible has been offered by Fehlberg (1964). For a given order, say $p + 1$, the corresponding member of the Fehlberg family computes z_{j+1} with a minimum number of function calls, according to the limitations in Table 7.10, and then provides the pth-order estimate y_{j+1} without any additional function calls. A particularly popular Fehlberg rule is given in Table 7.21, which gives a fifth-order estimate z_{j+1} for a fourth-order rule y_{j+1}.

Subroutine RKF (Table 7.22) implements a single step of this Runge–Kutta–Fehlberg formula, outputting y_{j+1} and z_{j+1} as the parameters YOUT and ZOUT. In view of (7.52), the difference of these values provides a local error estimate. Subroutine ARUKU (Table 7.23) utilizes RKF to update the step size as the computation progresses. If the absolute value of ZOUT-YOUT is less than

TABLE 7.21 Runge–Kutta–Fehlberg Formula

$$k_1 = f(x_j, y_j)$$

$$k_2 = f\left(x_j + \frac{1}{4}h,\ y_j + \frac{1}{4}hk_1\right)$$

$$k_3 = f\left(x_j + \frac{3}{8}h,\ y_j + h\left(\frac{3}{32}k_1 + \frac{9}{32}k_2\right)\right)$$

$$k_4 = f\left(x_j + \frac{12}{13}h,\ y_j + h\left(\frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right)\right)$$

$$k_5 = f\left(x_j + h,\ y_j + h\left(\frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right)\right)$$

$$k_6 = f\left(x_j + \frac{1}{2}h,\ y_j + h\left(-\frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right)\right)$$

$$y_{j+1} = y_j + h\left(\frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5\right)$$

$$z_{j+1} = y_j + h\left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right)$$

TABLE 7.22 Subroutine RKF for the Runge–Kutta–Fehlberg Formula

```
      SUBROUTINE RKF(XI,YI,H,YOUT,ZOUT)
C
C  ************************************************************
C  *  FUNCTION: A CALL TO THIS SUBROUTINE COMPUTES ONE STEP OF*
C  *            THE SOLUTION AND A GUESS OF THE ERROR FOR A   *
C  *            DIFFERENTIAL EQUATION Y'=F(X,Y) WITH INITIAL  *
C  *            VALUES XI, YI. THIS SOLUTION IS OBTAINED      *
C  *            USING A 4-TH ORDER RUNGE-KUTTA FEHLBERG STEP  *
C  *            METHOD IMBEDDED IN A 5-TH ORDER STEP SOLUTION *
C  *  USAGE:                                                  *
C  *       CALL SEQUENCE: CALL RKF(XI,YI,H,YOUT,ZOUT)         *
C  *       EXTERNAL FUNCTIONS/SUBROUTINES: FUNCTION F(U,V)    *
C  *  PARAMETERS:                                             *
C  *       INPUT:                                             *
C  *          XI=INDEPENDENT VARIABLE INITIAL VALUE           *
C  *          YI=DEPENDENT VARIABLE INITIAL VALUE             *
C  *           H=INTERVAL STEP SIZE                           *
C  *       OUTPUT:                                            *
C  *         YOUT=4-TH ORDER SOLUTION ESTIMATE                *
C  *         ZOUT=5-TH ORDER SOLUTION ESTIMATE                *
C  *              (ZOUT-YOUT=LOCAL ERROR ESTIMATE)            *
C  ************************************************************
C
      REAL K1,K2,K3,K4,K5,K6
      K1=F(XI,YI)
      U=XI+0.25*H
      V=YI+0.25*H*K1
      K2=F(U,V)
      U=XI+(3./8.)*H
      V=YI+H*((3./32.)*K1+(9./32.)*K2)
      K3=F(U,V)
      U=XI+H*(12./13.)
      V=YI+(H/2197.)*(1932.*K1-7200.*K2+7296.*K3)
      K4=F(U,V)
      U=XI+H
      V=YI+H*((439./216.)*K1-8.*K2+(3680./513.)*K3-
     1(845./4104.)*K4)
      K5=F(U,V)
      U=XI+0.5*H
      V=-(8./27.)*K1+2.*K2-(3544./2565.)*K3+
     1(1859./4104.)*K4-(11./40.)*K5
      V=YI+H*V
      K6=F(U,V)
      YOUT=(25./216.)*K1+(1408./2565.)*K3+
     1(2197./4104.)*K4-K5/5.
      YOUT=YI+H*YOUT
      ZOUT=(16./135.)*K1+(6656./12825.)*K3+
     1(28561./56430.)*K4-(9./50.)*K5
      ZOUT=ZOUT+(2./55.)*K6
      ZOUT=YI+H*ZOUT
      RETURN
      END
```

the user-specified value TOL (for tolerance), the value ZOUT is accepted for y_{j+1}, and a larger step size (by a factor of 3) is chosen for the next step. Otherwise, h is reduced by a factor of 10, and the computation is repeated from the same condition x_j and y_j. Strictly speaking, YOUT, rather than ZOUT, should be chosen for y_{j+1}, but since in principle the higher-order estimate ZOUT should be more accurate, and since it is available, we adopt the pragmatic viewpoint that it should be used. The reader will note that AKUKU is an obvious modification of subroutine ASIMP for adaptive quadrature (Section 3.8.2).

TABLE 7.23 Subroutine AKUKU for the Adaptive Runge–Kutta Method

```
      SUBROUTINE ARUKU(X,Y,B,M,TOL)
C
C     ************************************************************
C     *  FUNCTION: THIS SUBROUTINE COMPUTES THE SOLUTION OF A    *
C     *            DIFFERENTIAL EQUATION BY ADAPTIVELY CHOOSING  *
C     *            THE STEP SIZE TO LIMIT THE LOCAL ERROR EST-   *
C     *            IMATE WITHIN A GIVEN TOLERANCE. A 4-TH ORDER  *
C     *            RUNGE-KUTTA-FEHLBERG METHOD IS USED           *
C     *  USAGE:                                                  *
C     *       CALL SEQUENCE: CALL ARUKU(X,Y,B,M,TOL)             *
C     *       EXTERNAL FUNCTIONS/SUBROUTINES:                    *
C     *                      SUBROUTINE RKF(XI,YI,H,YOUT,ZOUT)   *
C     *  PARAMETERS:                                             *
C     *       INPUT:                                             *
C     *        X(1)=INDEPENDENT VARIABLE INITIAL VALUE           *
C     *        Y(1)=DEPENDENT VARIABLE INITIAL VALUE             *
C     *           B=SOLUTION INTERVAL ENDPOINT (LAST X VALUE)    *
C     *           M=MAXIMUM NUMBER OF ITERATIONS                 *
C     *      OUTPUT:                                             *
C     *           X=M BY 1 ARRAY OF INDEPENDENT VARIABLE VALUES  *
C     *           Y=M BY 1 ARRAY OF DEPENDENT VARIABLE SOLUTION  *
C     *             VALUES                                       *
C     ************************************************************
C
      DIMENSION X(M),Y(M)
C     *** INITIALIZATION ***
      H=.10E-02
      I=1
      N=0
C     *** COMPUTE SOLUTION ITERATIVELY ***
      DO WHILE(X(I).LE.B)
         N=N+1
         CALL RKF(X(I),Y(I),H,YOUT,ZOUT)
         ERR=ZOUT-YOUT
C     *** TEST IF THE NUMBER OF ITERATIONS EXCEEDED ***
         IF(N.GT.M) THEN
            WRITE(6,1)
    1       FORMAT(1X,'PROGRAM STOPPED TOO MANY ITERATIONS')
            STOP
         END IF
C     *** TEST STEP SIZE ***
         IF(ABS(ERR).LT.TOL) THEN
            I=I+1
            X(I)=X(I-1)+H
            H=3.0*H
            Y(I)=ZOUT
         ELSE
            H=H/10.0
         END IF
      END DO
      M=I
      H=B-X(I-1)
      X(M)=X(I-1)+H
      CALL RKF(X(I-1),Y(I-1),H,Y(M),ZOUT)
      RETURN
      END
```

We serve notice that the code ARUKU is intended only to illustrate the principles of automatic error control. It is inefficient and does not have the safeguards of a professional differential equation program package. More will be said about this matter after the following computational example.

EXAMPLE 7.17

By means of the calling program given in Table 7.24, the automatic step-size routine ARUKU is called on to solve the differential equation

$$y' = y, \qquad y(0) = 1 \tag{7.53}$$

over the interval [0, 1]. We chose this over our "usual" differential equation because in the present case it is easy to compute the exact local error $\hat{y}(x_{j+1}) - y_{j+1}$ and thereby see how well the RKF error estimator is doing. Specifically, the solution of (7.53) that passes through points (x_j, y_j) is

$$\hat{y}(x) = y_j \exp(x - x_j),$$

and if h is the current step size, then the exact local error is given by

$$y_j \exp(h) - \text{YOUT}.$$

TABLE 7.24 Calling Program for the Subroutine ARUKU

```
C     PROGRAM RKFMETH
C
C     ******************************************************************
C     THIS PROGRAM WILL SET UP AND SOLVE NUMERICALLY THE DIFFERENTIAL
C     EQUATION Y'=Y WITH THE INITIAL CONDITION Y(0.)=1.
C     THE SOLUTION IS OBTAINED USING THE AUTOMATIC STEPSIZE ROUTINE
C     USING 4-TH ORDER RUNGE-KUTTA FEHLBERG METHOD
C     CALLS:   ARUKU,RKF (BOTH MODIFIED FOR DOUBLE PRECISION)
C     OUTPUT:
C              X(I)=VALUE OF X FOR I=1,...  (MAX=50)
C              Y(I)=APPROXIMATED VALUE OF Y AT X(I)
C     ******************************************************************
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION X(50),Y(50)
C
C     *** FIRST, THE INITIAL CONDITION AND ENDPOINT ARE ESTABLISHED***
C     *** THE MAXIMUM NUMBER OF ITERATIONS IS SET TO 50            ***
C
      X(1)=0.D0
      Y(1)=1.D0
      B=1.D0
      MAX=50
      TOL=1.D-4
C
C     *** SUBROUTINE ARUKU WILL APPROXIMATE THE SOLUTION           ***
C     *** RETURNING AT MOST 50 VALUES IN ARRAYS X AND Y            ***
C
      CALL ARUKU(X,Y,B,MAX,TOL)
      WRITE(10,*)(X(I),Y(I),I=1,50)
      STOP
      END
C
C     *** FUNCTION F SPECIFIES THE DIFFERENTIAL EQUATION. IT IS    ***
C     *** CALLED BY SUBROUTINE RKF                                 ***
C
      FUNCTION F(X,Y)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      F=Y
      RETURN
      END
```

EGM 5346

COMP. ENG. ANAL.

4/12/06

MULTISTEP METHODS

$$y' = f(x,y)$$

ADAMS METHOD

PASS A QUAD. THRU $x_i, f_i = f(x_i,y_i)$

x_{i-1}, f_{i-1}

x_{i-2}, f_{i-2}

$$f(x,y) = f_i + s(f_i - f_{i-1}) + \frac{(s+1)s}{2}(f_i - 2f_{i-1} + f_{i-2})$$

$$s = \frac{x - x_i}{\Delta x}$$

$$\int_{x_i}^{x_{i+1}} \frac{dy}{dx}\,dx = \int_{x_i}^{x_{i+1}} f(x,y)\,dx$$

$$y_{i+1} = y_i + \frac{\Delta x}{12}[23 f_i - 16 f_{i-1} + 5 f_{i-2}] + O(\Delta x^3)$$

G.E. $O(\Delta x^3)$

NEWTON-GREGORY

CUBIC (x_i, f_i) (x_{i-1}, f_{i-1}) (x_{i-2}, f_{i-2})

(x_{i-3}, f_{i-3})

$$f(x,y) = f_i + s(f_i - f_{i-1}) + \frac{(s+1)s}{2}(f_i - 2f_{i-1} + f_{i-2})$$

$$+ \frac{(s+2)(s+1)s}{6}(f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3})$$

$$y_{i+1} = y_i + \frac{\Delta x}{24}[55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}]$$

$$+ \text{L.E. } O(\Delta x^5)$$

MILNE'S METHOD

$$\int_{y_{i-3}}^{y_{i+1}} \frac{dy}{dx}\,dx = \int f(\bar{x}, y)\,d\bar{x}$$

DEFINE $F(x,y)$ using $i, i-1, i-2, i-3$

use this formula & integrate over $i, i-1, i-2, i-3$

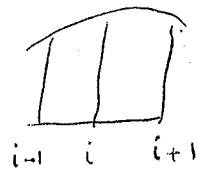
so that $y_{i+1_p} = y_{i-3} + \frac{4\Delta x}{3}(2f_i - f_{i-1} + 2f_{i-2}$

now define $f_{i+1} = f(x_{i+1}, y_{i+1p})$

$$\int_{x_{i-1}}^{x_{i+1}} \frac{dy}{dx}\,dx = \int_{x_{i-1}}^{x_{i+1}} f\,dx$$

let f be quadratic using $i+1, i, i-1$

SIMPSONS RULE



$$y_{i+1_c} = y_{i-1} + \frac{\Delta x}{3}\left(f_{i+1} + 4f_i + f_{i-1}\right)$$

ADAMS MOULTON METHOD

$y' = f(x,y)$

CUBIC THRU $i, i-1, i-2, i-3$

& INTEGRATES IT OVER i TO $i+1$

$$\int_{x_i}^{x_{i+1}} y'\,dx = \int_{x_i}^{x_{i+1}} f(x,y)\,dx$$

$\Rightarrow y_{i+1_p}$

DFN CUBIC over $i+1, i, i-1, i-2$ & INTEGRATE OVER i TO $i+1$

$$y_{i+1_p} = y_i + \frac{\Delta x}{24}\left(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}\right)$$

$$y_{i+1_c} = y_i + \frac{\Delta x}{24}\left(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}\right)$$

GE $O(\Delta x^4)$

EGM 5346

4/7/99

DIFFERENCE EQNS : example $y' = 5 - 5y = f(x,y)$

$$y_i' = 5 - 5y_i \Rightarrow y_i' + 5y_i = 5$$

$$y_i = \text{?}e^{-5x_i}$$

$$\frac{y_{i+1} - y_i}{\Delta x} = 5 - 5y_i$$

$$y_{ip} = D \; ; \; 5D = 5 \; ; \; D = 1$$

$$y_{i+1} + y_i(5\Delta x - 1) = 5\Delta x$$

$$y_{i_p} = C = y_{i+1_p} \Rightarrow C = 1 = y_{ip}$$

HOMOG: $y_{i+1} + y_i(5\Delta x - 1) = 0$

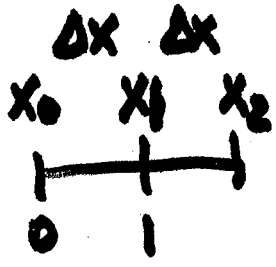
$$y_i = \text{?}\beta^i \Rightarrow \text{?}\beta^i[\beta + (5\Delta x - 1)] = 0$$

$$\beta = (1 - 5\Delta x)$$

$$y_i = \text{?}(1 - 5\Delta x)^i$$

$$e = \lim_{\varepsilon \to 0} (1+\varepsilon)^{1/\varepsilon}$$

$$\frac{x_i}{i} = \Delta x$$



$$y_{i+1_c} = y_{i-1} + \frac{\Delta x}{3}\left(f_{i+1} + 4f_i + f_{i-1}\right)$$

$$f_i \cong \lambda y_i$$

$$y_{i+1} = y_{i-1} + \frac{\lambda \Delta x}{3}\left[y_{i+1} + 4y_i + y_{i-1}\right]$$

$$y_{i+1}\left[1 - \frac{\lambda\Delta x}{3}\right] - 4y_i\frac{\lambda\Delta x}{3} - y_{i-1}\left[1 + \frac{\lambda\Delta x}{3}\right] = 0$$

$$y_i = G\beta^i$$

$$G\beta^{i-1}\left[\beta^2\left(1 - \frac{\lambda\Delta x}{3}\right) - 4\beta\frac{\lambda\Delta x}{3} - \left(1 + \frac{\lambda\Delta x}{3}\right)\right] = 0$$

$$\beta = \frac{4\frac{\lambda\Delta x}{3} \pm \sqrt{16\frac{\lambda^2\Delta x^2}{9} + 4\left(1 - \frac{\lambda\Delta x}{3}\right)\left(1 + \frac{\lambda\Delta x}{3}\right)}}{2\left(1 - \frac{\lambda\Delta x}{3}\right)}$$

$$= \frac{4\frac{\lambda\Delta x}{3} \pm 2\sqrt{1 + \frac{\lambda^2\Delta x^2}{9}}}{2\left(1 - \frac{\lambda\Delta x}{3}\right)}$$

$$= \frac{2\frac{\lambda\Delta x}{3} \pm \sqrt{1 + \frac{\lambda^2\Delta x^2}{3}}}{1 - \frac{\lambda\Delta x}{3}}$$

$$\sqrt{1+\epsilon} = 1 + \frac{1}{2}\epsilon$$

$$\frac{1}{1-\epsilon} = 1 + \epsilon + \epsilon^2 + \ldots + \epsilon^n$$

$$\frac{2\frac{\lambda\Delta x}{3} \pm \left(1 + \frac{1}{6}\lambda^2\Delta x^2\right)}{1 - \frac{\lambda\Delta x}{3}}$$

$$\beta = \left[\frac{2\lambda\Delta x}{3} \pm \left(1 + \frac{1}{6}\Delta x^2\lambda^2\right)\right]\left[1 + \frac{\lambda\Delta x}{3}\right]$$

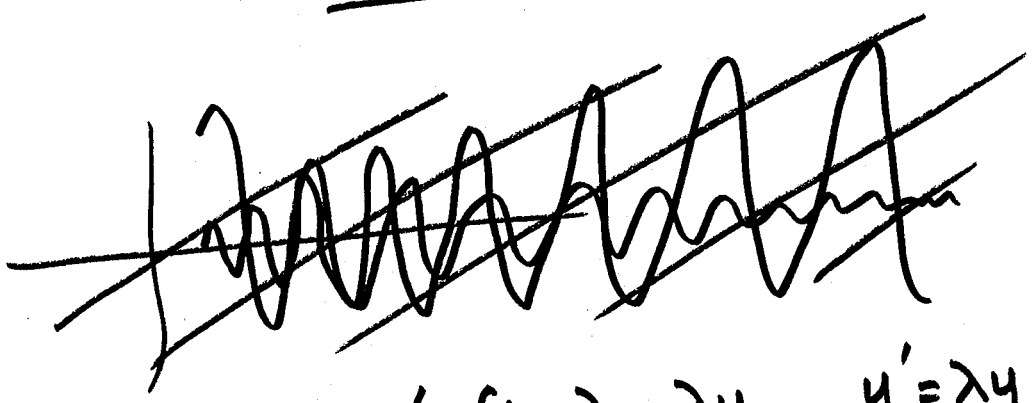
$$= \pm 1 + \frac{2\lambda\Delta x}{3} \pm \frac{\lambda\Delta x}{3} + \text{h.o.t. in } \Delta x^2$$

$$= 1 + \frac{\lambda\Delta x}{3}$$

$$-1 + \frac{\lambda\Delta x}{3}$$

$$y_i = c_1\beta_1^i + c_2\beta_2^i = c_1\left[1+\lambda\Delta x\right]^i + c_2\left[1 - \frac{\lambda\Delta x}{3}\right]^i (-1)^i$$

$$\lim_{\Delta x \to 0} \quad y_i \to c_1 e^{+\lambda x i} + c_2 e^{-\frac{\lambda x i}{3}(-1)^i}$$

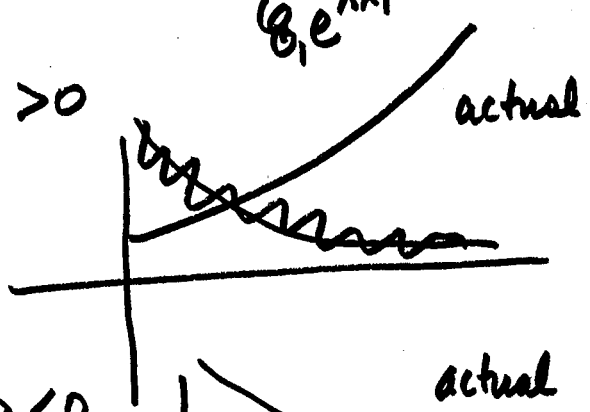


$$y' = f(x,y) = \lambda y \qquad y' = \lambda y$$

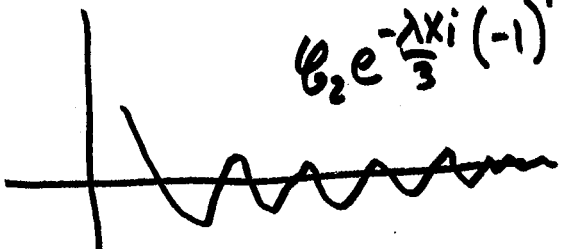
$$y = ce^{\lambda x}$$

$c_1 e^{\lambda x i}$

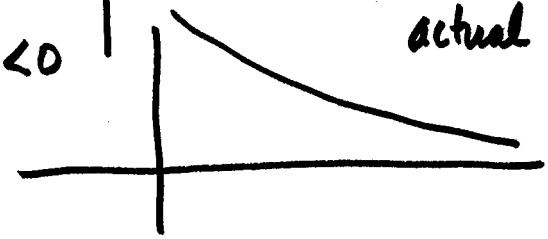
for $\lambda > 0$

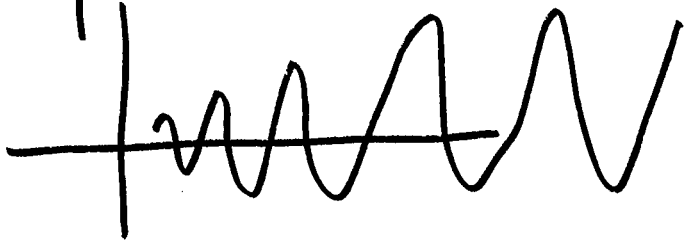


$c_2 e^{-\frac{\lambda x i}{3}(-1)^i}$



for $\lambda < 0$





$$y' = 5 - 5y_i$$

$$y' = \frac{y_{i+1} - y_{i-1}}{2\Delta x} = 5 - 5y_i$$

$$y_{i+1} + 10\Delta x y_i - y_{i-1} = 10\Delta x$$

if $y_{i_p} = C = y_{i+1_p} = y_{i-1_p}$ $\quad y_{i_p} = C = 1$

$$y_i = C\beta^i$$

$$C\beta^{i-1}\left[\beta^2 + 10\Delta x\beta - 1\right] = 0$$

$$\beta = \frac{-10\Delta x \pm \sqrt{(10\Delta x)^2 - 4(1)(-1)}}{2}$$

$$= -5\Delta x \pm \sqrt{1 + (5\Delta x)^2}$$

$$= \pm\left(1 + \frac{25}{2}\Delta x^2\right) - 5\Delta x$$

$$= \pm 1 - 5\Delta x + \text{h.o.t. in } \Delta x^2$$

$$y_i = C_1\beta_1^i + C_2\beta_2^i = C_1(1+5\Delta x)^i(-1)^i + C_2(1-5\Delta x)^i$$

$$y_i \rightarrow C_1 e^{+5x_i}(-1)^i + C_2 e^{-5x_i}$$

Table 6.1 ADAMS FORMULAS

Order of formula	Coefficient of h	Coefficients of f_i	f_{i-1}	f_{i-2}	f_{i-3}	f_{i-4}	f_{i-5}	Local truncation error E_T
1	1	1						$\frac{1}{2}h^2y''(\xi)$
2	$\frac{1}{2}$	3	−1					$\frac{5}{12}h^3y'''(\xi)$
3	$\frac{1}{12}$	23	−16	5				$\frac{3}{8}h^4y''''(\xi)$
4	$\frac{1}{24}$	55	−59	37	−9			$\frac{251}{720}h^5y^{V}(\xi)$
5	$\frac{1}{720}$	1901	−2774	2616	−1274	251		$\frac{475}{1440}h^6y^{VI}(\xi)$
6	$\frac{1}{1440}$	4277	−7923	9982	−7298	2877	−475	$\frac{19087}{60480}h^7y^{VII}(\xi)$

represents most of the local truncation error. From the definition of the third backward difference given in Section C.2 of Appendix C we can write

$$h(\tfrac{3}{8}\nabla^3 f_i) = \tfrac{3}{8}h(f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}). \tag{6.114}$$

Backward finite-difference expressions for derivatives are given by Equations (5.123). From the third of Equations (5.123) we deduce that

$$y'''_i = \frac{y'_i - 3y'_{i-1} + 3y'_{i-2} - y'_{i-3}}{h^3} + O(h)$$

or

$$h^3y'''_i = y'_i - 3y'_{i-1} + 3y'_{i-2} - y'_{i-3} + h^3[O(h)]. \tag{6.115}$$

From Equation (6.115) we determine that

$$(f_i - 3f_{i-1} + 3f_{i-2} - f_{i-3}) = h^3y'''_i - O(h^4). \tag{6.116}$$

Combining Equations (6.116) and (6.114),

$$h(\tfrac{3}{8}\nabla^3 f_i) = \tfrac{3}{8}h^4y''''_i - O(h^5) = \tfrac{3}{8}h^4y''''(\xi)$$

where the fourth derivative of y is evaluated at some unknown x value ξ in the range of x values spanned by the one-step application of the third-order Adams formula. Thus, the local truncation error of the third-order method is

$$E_T = \tfrac{3}{8}h^4y''''(\xi) \tag{6.117}$$

as shown in Table 6.1. In similar fashion we can show that

$$h(\tfrac{1}{2}\nabla f_i) = \tfrac{1}{2}h^2y''(\xi)$$

$$h(\tfrac{5}{12}\nabla^2 f_i) = \tfrac{5}{12}h^3y'''(\xi)$$

$$\vdots$$

for the other local truncation errors shown in Table 6.1.

Carrying out the integration indicated, the general formula is

$$y_{i+1} = y_i + h(f_{i+1} - \tfrac{1}{2}\nabla f_{i+1} - \tfrac{1}{12}\nabla^2 f_{i+1} - \tfrac{1}{24}\nabla^3 f_{i+1} - \tfrac{19}{720}\nabla^4 f_{i+1} - \tfrac{27}{1440}\nabla^5 f_{i+1} - \tfrac{863}{60,480}\nabla^6 f_{i+1} - \cdots). \tag{6.137}$$

To obtain the third-order Adams–Moulton corrector formula from this general formula, Equation (6.137) is truncated to three terms following y_i, which yields

$$y_{i+1} = y_i + h(-f_{i+1} - \tfrac{1}{2}\nabla f_{i+1} - \tfrac{1}{12}\nabla^2 f_{i+1}).$$

Then substituting the backward differences as given in Section C.2 in Appendix C into the above, we find

$$y_{i+1} = y_i + \frac{h}{12}(-f_{i-1} + 8f_i + 5f_{i+1}) \tag{6.138}$$

which is identical with Equation (6.129) derived previously.

The fourth-order Adams–Moulton corrector formula is found by truncating Equation (6.137) to four terms following y_i, yielding

$$y_{i+1} = y_i + h(f_{i+1} - \tfrac{1}{2}\nabla f_{i+1} - \tfrac{1}{12}\nabla^2 f_{i+1} - \tfrac{1}{24}\nabla^3 f_{i+1}).$$

Substituting the backward differences from Section C.2 of Appendix C into the above gives the corrector equation

$$y_{i+1} = y_i + \frac{h}{24}(f_{i-2} - 5f_{i-1} + 19f_i + 9f_{i+1}) \tag{6.139}$$

which is identical with Equation (6.134) found previously.

Using the terms in Equation (6.137), we can obtain Adams–Moulton corrector formulas of orders 1 through 7, or of orders 1 through 6 plus a local truncation error expression for each of the six. These expressions are given in Table 6.2. In the local truncation-error expressions ξ is an unknown x value in the range of x values spanned in the one-step application of a particular Adams–Moulton formula. While explicit numerical values cannot be found for these expressions because the derivatives in them cannot be evaluated, they are useful for comparing the truncation errors of the various Adams–Moulton formulas, and for comparing the truncation errors of these formulas with those of other

Table 6.2 ADAMS–MOULTON FORMULAS AND LOCAL TRUNCATION-ERROR EXPRESSIONS

Order of formula	Coefficient of h	Coefficients of f_{i+1}	f_i	f_{i-1}	f_{i-2}	f_{i-3}	f_{i-4}	Local truncation error E_T
1	1	1						$-\frac{1}{2}h^2y''(\xi)$
2	$\frac{1}{2}$	1	1					$-\frac{1}{12}h^3y'''(\xi)$
3	$\frac{1}{12}$	5	8	−1				$-\frac{1}{24}h^4y''''(\xi)$
4	$\frac{1}{24}$	9	19	−5	1			$-\frac{19}{720}h^5y^{V}(\xi)$
5	$\frac{1}{720}$	251	646	−264	106	−19		$-\frac{27}{1440}h^6y^{VI}(\xi)$
6	$\frac{1}{1440}$	475	1427	−798	482	−173	27	$-\frac{863}{60480}h^7y^{VII}(\xi)$

what about $y' = f(x,y)$

$$\frac{y_{i+1} - y_i}{\Delta x} = f(x_i, y_i)$$

$$y_{i+1} = y_i + \Delta x\, f_i$$

normally we can't solve so we use $y_{i+1} = y_i + \Delta x \cdot \lambda y_i$ where $f_i \cong \lambda y_i$

for Adams - Bashforth $y_{i+1} = y_i + \frac{\Delta x \lambda}{24}(55y_i - 59y_{i-1} + 37y_{i-2} - 9y_{i-3})$

if $y_i = \beta^i$ then $0 = \beta^{i+1} - \beta^i - \frac{\Delta x \lambda}{24}(55\beta^i - 59\beta^{i-1} + 37\beta^{i-2} - 9\beta^{i-3})$

or $\beta^4 - \beta^3 - \frac{\Delta x \lambda}{24}(55\beta^3 - 59\beta^2 + 37\beta - 9) = 0$

$\rho(\beta) + \Delta x \lambda\, \sigma(\beta) = 0$; as $\Delta x \to 0$ this reduces to $\rho(\beta) = 0$

$\therefore \beta^3(\beta - 1) = 0$ or $\beta_{1,2,3} = 0$ & $\beta_4 = 1$ β

$$\therefore y_i = c_1 \beta_1^i + c_2 i \beta_2^i + c_3 i^2 \beta_3^i + c_4 \beta_4^i$$

as $h \to 0$ $y_i \to$ solution to $y' = \lambda y$ & other solutions are extraneous

Since the original equation is a continuous function of Δx then it follows that for small Δx $|\beta_i| < 1$ for $i = 1,2,3$ & the method is stable

What about Milne's method $y_{n+1} = y_{n-1} + \frac{h}{3}(f_{n+1} + 4f_n + f_{n-1})$

if $f(x,y) = \lambda y$ then we get

$$(\beta^2 - 1) + \frac{\lambda h}{3}(\beta^2 + 4\beta + 1) = 0$$

$$\beta = \frac{+4\frac{\lambda h}{3} \pm \sqrt{\left(4\frac{\lambda h}{3}\right)^2 + 4\left(1 + \frac{\lambda h}{3}\right)\left(\frac{\lambda h}{3} - 1\right)}}{2\left(1 - \frac{\lambda h}{3}\right)}$$

$\beta_1 = 1 + \lambda h$

$\beta_2 = -\left(1 - \lambda h/3\right)$

and the solution is $y_n \to c_1 e^{\lambda x_n} + c_2 (-1)^n e^{-\lambda x_n/3}$

special equation $y' = \lambda y$, λ constant, is usually considered sufficient, however, to give an indication of the stability of a method.

We consider first the Adams-Bashforth fourth-order method. If in (8.47) we set $f(x, y) = \lambda y$ we obtain

$$y_{n+1} - y_n - \frac{h\lambda}{24}(55y_n - 59y_{n-1} + 37y_{n-2} - 9y_{n-3}) = 0 \quad (8.73)$$

The characteristic equation for this difference equation is

$$\beta^4 - \beta^3 - \frac{h\lambda}{24}(55\beta^3 - 59\beta^2 + 37\beta - 9) = 0$$

The roots of this equation are of course functions of $h\lambda$. It is customary to write the characteristic equation in the form

$$\rho(\beta) + h\lambda\sigma(\beta) = 0 \quad (8.74)$$

where $\rho(\beta)$ and $\sigma(\beta)$ are polynomials defined by

$$\rho(\beta) = \beta^4 - \beta^3$$

$$\sigma(\beta) = -\frac{1}{24}(55\beta^3 - 59\beta^2 + 37\beta - 9)$$

We see that as $h \to 0$, (8.74) reduces to $\rho(\beta) = 0$, whose roots are $\beta_1 = 1$, $\beta_2 = \beta_3 = \beta_4 = 0$. For $h \neq 0$, the general solution of (8.73) will have the form

$$y_n = c_1\beta_1^n + c_2\beta_2^n + c_3\beta_3^n + c_4\beta_4^n$$

where the β_i are solutions of (8.74). It can be shown that β_1^n approaches the desired solution of $y' = \lambda y$ as $h \to 0$ while the other roots correspond to extraneous solutions. Since the roots of (8.74) are continuous functions of h, it follows that for h small enough, $|\beta_i| < 1$ for $i = 2, 3, 4$, and hence from the definition of stability that the Adams-Bashforth method is strongly stable. All multistep methods lead to a characteristic equation in the form (8.74) whose left-hand side is sometimes called the **stability polynomial**. The definition of stability can be recast in terms of the stability polynomial. A method is **strongly stable** if all the roots of $\rho(\beta) = 0$ have magnitude less than one except for the simple root $\beta = 1$.

We investigate next the stability properties of Milne's method (8.64*b*) given by

$$y_{n+1} = y_{n-1} + \frac{h}{3}(f_{n+1} + 4f_n + f_{n-1}) \quad (8.75)$$

Again setting $f(x, y) = \lambda y$ we obtain

$$y_{n+1} - y_{n-1} - \frac{h\lambda}{3}(y_{n+1} + 4y_n + y_{n-1}) = 0$$

and its characteristic equation becomes

$$\rho(\beta) + h\lambda\sigma(\beta) = 0 \quad (8.76)$$

with

$$\rho(\beta) = \beta^2 - 1$$

$$\sigma(\beta) = \beta^2 + 4\beta + 1$$

This time $\rho(\beta) = 0$ has the roots $\beta_1 = 1$, $\beta_2 = -1$, and hence by the definition above, Milne's method is not strongly stable. To see the implications of this we compute the roots of the stability polynomial (8.76). For h small we have

$$\beta_1 = 1 + \lambda h + \mathcal{O}(h^2)$$

$$\beta_2 = -(1 - \lambda h/3) + \mathcal{O}(h^2) \quad (8.77)$$

Hence the general solution of (8.75) is

$$y_n = c_1(1 + \lambda h + \mathcal{O}(h^2))^n + c_2(-1)^n(1 - \lambda h/3 + \mathcal{O}(h^2))^n$$

If we set $n = x_n/h$ and let $h \to 0$, this solution approaches

$$y_n = c_1 e^{\lambda x_n} + c_2(-1)^n e^{-\lambda x_n/3} \quad (8.78)$$

In this case stability depends upon the sign of λ. If $\lambda > 0$ so that the desired solution is exponentially increasing, it is clear that the extraneous solution will be exponentially decreasing so that Milne's method will be stable. On the other hand if $\lambda < 0$, then Milne's method will be unstable since the extraneous solution will be exponentially increasing and will eventually swamp the desired solution. Methods of this type whose stability depends upon the sign of λ for the test equation $y' = \lambda y$ are said to be **weakly stable**. For the more general equation $y' = f(x, y)$ we can expect weak stability from Milne's method whenever $\partial f/\partial y < 0$ on the interval of integration.

In practice all multistep methods will exhibit some instability for some range of values of the step h. Consider, for example, the Adams-Bashforth method of order 2 defined by

$$y_{n+1} = y_n + \frac{h}{2}\{3f_n - f_{n-1}\}$$

If we apply this method to the test equation $y' = \lambda y$, we will obtain the difference equation

$$y_{n+1} - y_n - \frac{h\lambda}{2}\{3y_n - y_{n-1}\} = 0$$

and from this the stability polynomial

$$\beta^2 - \beta - \frac{h\lambda}{2}\{3\beta - 1\}$$

or the equation

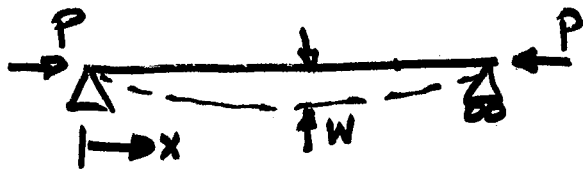
$$\beta^2 - \left(1 + \frac{3h\lambda}{2}\right)\beta + \frac{h\lambda}{2} = 0$$

EGM 5346

4/19/06 LECTURE

COMP. ENG. ANALYSIS

$$EI\,w'' + Pw = 0$$



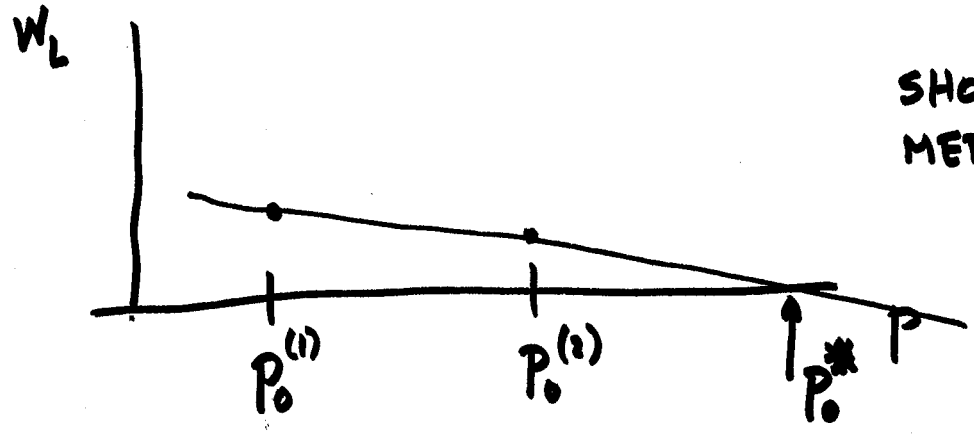
$$\left.\begin{aligned} w' &= p \\ p' = w'' &= -\frac{Pw}{EI} \end{aligned}\right\} \quad \begin{aligned} &w(x=0) = 0 \\ &p(x=0) = p_0 = ? \end{aligned} \qquad w(x=L) = 0$$

Choose $p_0^{(1)} \Rightarrow w(x=L) = w_L^{(1)}(p_0^{(1)}) \neq 0$

example using Euler

$$\begin{aligned} w_{i+1} &= w_i + p_i \Delta x \\ p_{i+1} &= p_i - \left(\frac{P}{EI}\right)_i w_i \cdot \Delta x \end{aligned}$$

Choose $p_0^{(2)} \Rightarrow w(x=L) = w_L^{(2)}(p_0^{(2)}) \neq 0$

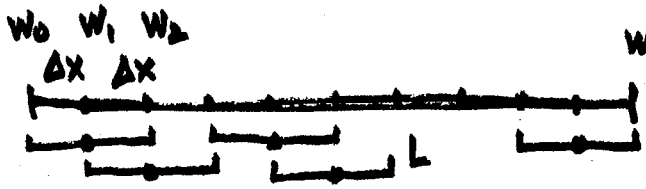


SHOOTING METHOD

$$p_0^* = p_0^{(1)} + \frac{p_0^{(2)} - p_0^{(1)}}{w_L^{(2)} - w_L^{(1)}} \cdot \left(w - w_L^{(1)}\right)$$

$$w'' = f(w, x, w') = -\frac{Pw}{EI} \qquad \frac{P}{EI} = \lambda$$

$$\frac{w_{i+1} - 2w_i + w_{i-1}}{\Delta x^2} = -\lambda w_i$$



$$\Delta x = \frac{L}{N}$$

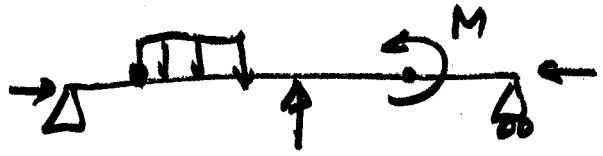
$$w_{i+1} + w_i(\lambda \Delta x^2 - 2) + w_{i-1} = 0$$

$$\begin{bmatrix} \lambda\Delta x^2-2 & 1 & 0 & \cdots & 0 \\ 1 & \lambda\Delta x^2-2 & 1 & \cdots & 0 \\ 0 & 1 & \lambda\Delta x^2-2 & \cdots & 0 \\ \vdots & & & \ddots & \\ 0 & \cdots & 0 & 1 & \lambda\Delta x^2-2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{N-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$\det A = 0 \Rightarrow$ gives the λ's $\quad \lambda$ - buckling loads

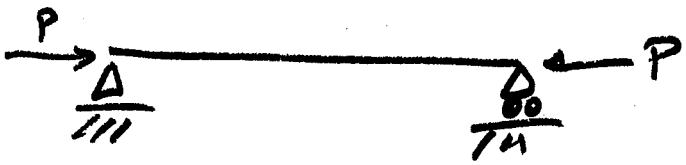
$\det A \neq 0 \Rightarrow w_i = 0$

$$w'' + \frac{P}{EI} w = \frac{M(x)}{EI}$$



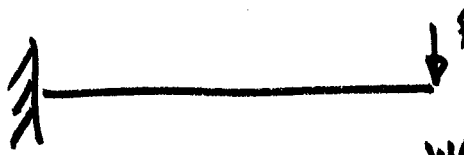
$$w_{i+1} + w_i(\lambda \Delta x^2 - 2) + w_{i-1} = \frac{M_i}{EI_i}$$

$$\begin{bmatrix} \lambda\Delta x^2-2 & 1 & & & \\ 1 & \ddots & \ddots & & \\ & \ddots & \ddots & 1 & \\ & & 1 & \lambda\Delta x^2-2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{N-1} \end{bmatrix} = \begin{bmatrix} M_1/EI_1 \\ M_2/EI_2 \\ \vdots \\ M_{N-1}/EI_{N-1} \end{bmatrix}$$



$$EIw'' + Pw = 0$$

$w(x=0)=0$ $\qquad$ $w(x=L)=0$ $\Big\}$ Boundary Value Problem



$w(x=0)=0$ $\qquad$ $w(x=L) = \frac{PL^3}{3EI}$

CAN USE THE FD OR SHOOTING METHOD

FD ON LINEAR ODE'S LEAD TO TRIDIAGONAL SYS.

FD ON NONLINEAR ODE'S " " " " BUT NEED TO USE ITERATIVE TECHNIQUES TO SOLVE

ADVANTAGES OF FD : DON'T NEED A SET OF 1ST ORDER EQS.

ACCURACY DEPENDS ON MESH SIZE (SMALLER $\Delta x \longleftrightarrow$ ACC $\uparrow$)

SHOOTING METHODS CAN BE USED FOR LINEAR & NONLINEAR

- EASY TO IMPLEMENT SINCE WE CAN USE (R-K, ADAMS, A-M...)
- NO GUARANTEE OF CONVERGENCE.
- IF THEY CONVERGE, THEY CONVERGE FASTER THAN FD

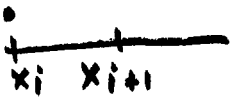
Total Error in Solution of D.E.

depends on

1) Round off error (due to significant digits error)

2) Truncation error due to representation of y' by an approximate formula (stepsize error)

3) e_{n+1} depends on errors that existed before hand (at some step) i.e. $e_{n+1} = f(e_n)$



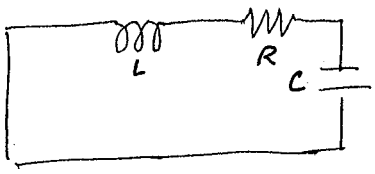
- Double Precision can combat round off error.
- RK or RKF or formulas that have high α $O(h^\alpha)$ can take care of truncation errors.
- Stable methods can reduce effects of truncation errors.

SELECTING A NUMERICAL INTEGRATION METHOD

- IF RANGE OF INTEGRATION MODERATE (COMPUTING TIME NOT A FACTOR) & NO NEED FOR STEP SIZE CONTROL USE R-K
- IF MANY INTEGRATIONS MUST BE DONE (TIME IS IMPORTANT) BUT ACCURACY NOT REQUIRED USE MODIFIED EULER
- IF RANGE OF INTEGRATION LARGE $\Rightarrow$ COMPUTING TIME IMPORTANT USE HIGH ORDER ACCURACY LIKE ADAMS-MOULTON
- IF COMPUTING TIME IS NOT IMPORTANT BUT NEED ACCURACY USE RKSFxS

STIFF ODEs ARISE

- WHEN SYSTEM IS MODELED BY DIFF. EQS WHOSE EIGENVALUES VARY WIDELY
  - CONTROL SYSTEMS
  - SYSTEMS W/ FLUID BOUNDARY LAYERS
  - ELECTRICAL CIRCUITS
- CAUSE PROBLEMS OF STABILITY & ACCURACY



$\frac{R}{L} = 10{,}001 \ \frac{1}{sec}$

$\frac{1}{LC} = 10{,}000 \ \frac{1}{sec^2}$

Consider circuits in which

$$\frac{d^2v}{dt^2} + \frac{R}{L}\frac{dv}{dt} + \frac{1}{LC}v = 0$$

with $v(0) = 1$ volt & $\frac{dv}{dt}(0) = -1$ volt/sec

solution will be $v = ce^{st}$ & $(s^2 + \frac{R}{L}s + \frac{1}{LC})ce^{st} = 0 \Rightarrow s^2 + 10{,}001s + 10{,}000 = 0$

and $v = ae^{-t} + be^{-10000t}$

- EV are -1 & $-10\,000$ i.e. $(s+10000)(s+1) = 0$
- IC give that $a = 1$ $b = 0$ $\Rightarrow v = e^{-t}$

5th order or Adams Moulton

however: Runge Kutta blows up (10^{293} volts in 0.2 sec) using $\Delta t = .001$

- unstable at $\Delta t = 5 \times 10^{-4}$
- using $\Delta t = 1 \times 10^{-4}$ sec works ie Δt is governed by EV which is larger
- must use larger $c\bar{v}$ to determine Δt

Answer

Florida International University
Department of Mechanical Engineering

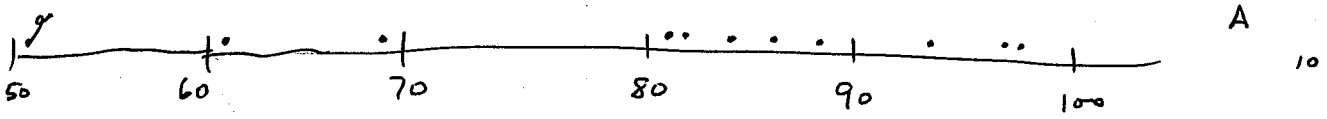
EGM 5346 EXAMINATION NO. 1 - B March 11, 1998

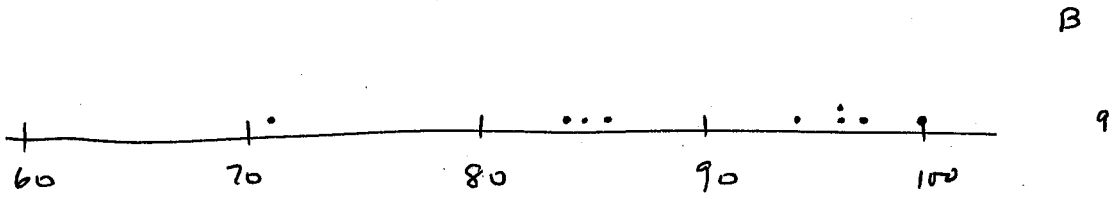
This exam is 80 minutes long. You may use your books and your notes but nothing else. Do all problems and show all work!!!!!!

Print your name and sign the following statement:

I will not give nor take any unpermitted aid during this examination. I understand that violation of this statement will lead to automatic failure of the examination.

PRINT NAME SIGN NAME





Problem 1 [**35 points**]. For the following equation

$$f(x) = x^3 - 3x^2 - 3x + 6$$

a) Determine the number of positive and negative roots as well as the complex roots;

b) Determine the region in which you would search for these roots

c) Using the Newton Raphson Method and an initial guess of $x_0 = 0$, compute the next three iterations x_1, x_2, x_3.

$f(x) = x^3 - 3x^2 - 3x + 6$

2 sign changes at most 2 + roots
2 − 2 = 0 ✓
2 − 1 = 1 ✗
2 − 0 = 2 ✓

$f(-x) = -x^3 - 3x^2 + 3x + 6$

1 sign change at most 1 − root
1 − 1 = 0 ✓
1 − 0 = 1 ✗

10

Only possibilities

P	N	C
2	1	0
0	1	2

$P(x) = x^3 - 3x^2 - 3x - 6$

x = 2 f(x) = 8 − 12 − 6 − 6 = −16
x = 3 f(x) = 27 − 27 − 9 − 6 = −15
x = 4 f(x) = 64 − 48 − 12 − 6 = −2] root
x = 5 125 − 75 − 15 − 6 = 29]

$Q(x) = x^3 + 3x^2 + 3x - 6$

x = 0 −6] root
x = 1 7

10

5 × 2

$0 \le |x| \le 5$

$f(x) = x^3 - 3x^2 - 3x + 6$
$f'(x) = 3x^2 - 6x - 3$

$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0 - \frac{6}{-3} = 2$ 5

$x_2 = 2 - \frac{f(x_1)}{f'(x_1)} = 2 - \frac{-4}{-3} = 2/3$ 5

$x_3 = 2/3 - \frac{f(x_2)}{f'(x_2)} = 2/3 - \frac{(8/27 - 4/3 - 2 + 6)}{(4/3 - 4 - 3)}$ 5

$\frac{80}{27}$ $\frac{-17}{3}$

$= 1.1895$

Problem 2 **[33 points]**

a) For the matrix, **A**, shown below, **use the Faddeev-Laverrier method** to deterrmine the matrix's characteristic equation.

b) Also find the matrix' inverse, $\mathbf{A}^{-1}$

c) Find the 1-norm, the 2-norm and the Euclidean norm of **A**

$$\begin{vmatrix} 4 & -2 \\ -2 & 4 \end{vmatrix} \quad \begin{pmatrix} -4 & -2 \\ -2 & -4 \end{pmatrix}$$

$B_1 = A = \begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix}$ $\quad p_1 = \text{tr}\, B_1 = 8$ 5

$B_2 = A\left[B_1 - p_1 \mathbb{A}\right] = \begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix}\begin{bmatrix} -4 & -2 \\ -2 & -4 \end{bmatrix} = \begin{bmatrix} -12 & 0 \\ 0 & -12 \end{bmatrix}$ $\quad p_2 = \frac{1}{2}\text{tr}\, B_2 = -12$ 13

$A^{-1} = \frac{1}{p_2}\left(B_1 - p_1 I\right) \quad +\frac{1}{12}\begin{bmatrix} +4 & +2 \\ +2 & +4 \end{bmatrix}$ 4

$p(\lambda) = (-1)^2\left(\lambda^2 - 8\lambda + 12\right)$ 3

$\|A\|_1 = 6$ $\quad \|A\|_2 = \sqrt{16+16+4+4} = 2\sqrt{10} \cong 6.3246$ 8

$\|A\|_2 = \max \lambda = \max(6,2) = 6$

Problem 3 [**32 points**]

Given the following data

x	1.000	1.100	1.300	1.400
y	3.800	4.200	5.200	5.700

a) Obtain the divided difference table from these data points

b) Evaluate y (x=1.15)using a third order interpolation polynomial passing through the **first** 3 points.

c) Is the result that you got for b) accurate? Why or why not?

1	3.8			
		4		
1.1	4.2		3.333	
		5		−8.333
1.3	5.2			
		5	0	
1.4	5.7			

(B) $P_3(x) = 3.8 + 4(x-1) + 3.333(x-1)(x-1.1) - 8.333(x-1)(x-1.1)(x-1.3)$

$P_3(1.15) = 3.8 + 4(.15) + 3.333(0.15)(0.05) - 8.333(0.15)(0.05)(-0.15)$

4.4344 6

(A) $P_3(x) = 5.7 + 5(x-1.4) + 0(x-1.4)(x-1.3) - 8.333(x-1.4)(x-1.3)(x-1.1)$

$P_3(1.25) = 5.7 + 5(-0.15) + 0 - 8.333(-0.15)(-0.05)(0.15)$

4.9406

$P_3(1.15)$ should be accurate } since it lies in the center of the interval. 6

$P_3(1.25)$ should be accurate

EGM 5346 MIDTERM EXAMINATION 5 March 1995

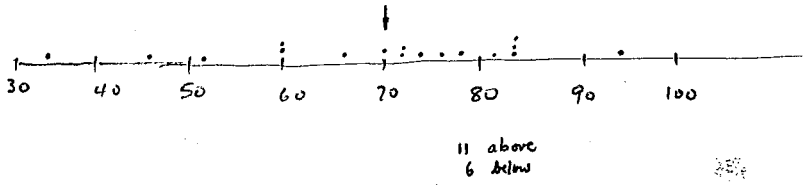
This is a 75 minute examination. You may use your books and your notes but nothing else.

Please sign the following:

> I certify that I will neither receive nor give unpermitted aid on this examination. Violation of this may result in failure of the exam.

PRINT NAME _____ SIGN NAME _____

Do all problems. **Read each question carefully. Show all work!!!!!**



11 above
6 below

PROBLEM 1 [40 POINTS]

a) For the matrix shown below, **use the Faddeev-Laverrier method** to determine the matrix's characteristic equation.

b) Also find the inverse of the matrix.

c) Find the 1-norm and the 2-norm of the matrix.

$$\begin{vmatrix} 4 & -1 \\ -1 & 4 \end{vmatrix}$$

$B_1 = A = \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}$ 4

$p_1 = \text{tr}\, B_1 = 4+4 = 8$ 1

$B_2 = A(B_1 - p_1 I) = \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}\begin{bmatrix} -4 & -1 \\ -1 & -4 \end{bmatrix} = \begin{bmatrix} -15 & 0 \\ 0 & -15 \end{bmatrix}$ 6

$p_2 = \frac{1}{2}\text{tr}\, B_2 = \frac{1}{2}[-30] = -15$ 1

$p(\lambda) = (-1)^2[\lambda^2 - p_1\lambda - p_2]$

$= (-1)^2[\lambda^2 - 8\lambda + 15]$ 3

b) $A^{-1} = \frac{1}{p_2}(B_1 - p_1 I) = -\frac{1}{15}\begin{bmatrix} -4 & -1 \\ -1 & -4 \end{bmatrix} = \frac{1}{15}\begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}$ 4

c) $\|A\|_1 = \max \text{col} \sum_{i=1}^{n} |a_{ij}| = 5$ 1

$\|A\|_2 = \max \left(\sum\sum (a_{ij}^2)\right)^{1/2} = \sqrt{34}$ 1

$\begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix}$ $p_1 = 8$

$\begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix}\begin{bmatrix} 4 & -2 \\ -2 & -4 \end{bmatrix} = \begin{bmatrix} -12 & \\ & -12 \end{bmatrix}$

$(4-\lambda)^2 - 4$ $p_2 = \frac{-24}{2} = -12$

$16 - 8\lambda + \lambda^2 - 4 =$

$p(\lambda) = (-1)^2[\lambda^2 - 8\lambda + 12]$

$\lambda = 6, \lambda = 2$

PROBLEM 2 [40 POINTS]

For $f(x) = x^3 - 4x^2 - 2x + 7$

a) Use Descartes' Rule of signs and the Cauchy Theorem to find the number of positive and negative real roots as well as the complex roots;

b) Determine the bounds of the roots.

c) Use the secant method to find a root of the equation START with initial guesses $x_0 = 0$ and $x_1 = 0.5$ AND COMPUTE x_2 and x_3 **ONLY**.

Cauchy $P(x) = x^3 - 4x^2 - 2x - 7$

$x=0$ $P(x) = -7$
$x=1$ $P(x) = -13$
$x=2$ $P(x) = -19$
$x=3$ $P(x) = -22$
$x=4$ $P(x) = -15$
$x=5$ $P(x) = 8$

$Q(x) = x^3 + 4x^2 + 2x - 7$

$x=0$ $Q(x) = -7$
$x=1$ $Q(x) = 0$

check between $0 \le |x| \le 5$ (4.735)

Descartes 2 sign changes → 3 possible roots → at most 2+ roots

$2-0=2$
$2-1=1$ x 3
$2-2=0$

$P(-x) = -x^3 - 4x^2 + 2x + 7$
1 sign change → at most 1 neg root
$1-0=1$ x 2
$1-1=0$ ok

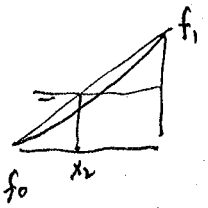
if 2+ 1 neg ✓ 2
0+ 3 neg x

$x_0 = 0$ $F(x_0) = 7$.
$x_1 = .5$ $F(x_1) = 5.125$.
$x_2 = 1.8667$. $F(x_2) = -4.1668$.
$x_3 = 1.2538$. $F(x_3) = .1753$. 6

$$x_{new} = \frac{+x_n F(x_{n+1}) \mp x_{n+1} F(x_n)}{F(x_{n+1}) - F(x_n)}$$

use for evaluation of $F(x)$

$F(x) = \big(((x-4)x-2)x+7\big)$



20

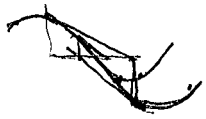
$\frac{0-f_0}{x_2-x_0} = \frac{f_1-f_0}{x_1-x_0}$

$-f_0(x_1-x_0) = (x_2-x_0)(f$

$-f_0x_1 + x_0f_1 = x_2($

$\frac{}{f_1-f_0}$

$x^3 - 3x^2 - 3x + 1 = f(x)$
$3x^2 - 6x - 3$ $f'(x)$
$x_1 = 0 - \frac{1}{-3} = 2$
$x_2 = 2 - \frac{-4}{-3} = \frac{2}{3}$



$1 - \frac{1}{-6} = \frac{7}{6}$

$\frac{8}{27} - \frac{4}{3} - 2 + 6 \approx \frac{-3}{-8}$
$4 - 4 - 3$

$\frac{2}{3} \approx \frac{3}{8}$

PROBLEM 3 [20 POINTS]

To speed up convergence, we learned about the Steffensen iterative technique with immediate replacement. Use this method on the results of problem 2, to determine the next two iterations.

$x_0 = 0 \quad x_1 = .5 \quad x_2 = 1.8667 \quad x_3 = 1.2538$

$\Delta x_n = d = \Delta x_1 = x_2 - x_1 = 1.3667$ $\quad d = \Delta x_2 = -.6129$

$\Delta x_{n-1} = \Delta x_0 = x_1 - x_0 = 0.5$ $\quad \Delta x_1 = 1.3667$

$r = 0.5/1.3667 = .3658 = \frac{\Delta x_{n-1}}{\Delta x_n}$ $\quad r = 1.3667/-.6129 = -2.2299$

$y_0 = 0$ $\quad y_1 = 1.8667 + 1.3667/(-.6342) = 1.4027$ $\quad y_2 = 1.2538 + .6129/(-3.2299) = 1.4436$

$y_n = x_{n+1} + \Delta x_n/(r-1)$

$\Delta^2 x_0 = \Delta x_1 - \Delta x_0 = .8667$

$(\Delta x_1)^2 = (1.3667)^2 = 1.8679$

$\hat{x}_0 = x_2 - \frac{1.8679}{.8667} = 1.8667 -$

$\hat{x}_n = x_{n+1} - \Delta x$

$x^3 - 4x^2 - 2x + 7 = f(x)$

let $x = \frac{x^3 - 4x^2 + 7}{2}$

$g' = \frac{3x^2 - 8x + 7}{2}$ $\quad$ for $0 < x < 5$ $\quad g' \neq < 1$

$x = 1 \quad g' = 1$

$x = 2 \quad g' = 1/2$

$x = \sqrt{\frac{x^3 - 2x + 7}{4}}$

$x = g(x) \quad g'(x) = \frac{(3x^2 - 2)/4}{2\sqrt{\frac{x^3 - 2x + 7}{4}}}$

$g'(0) = \left|\frac{-2/4}{2\sqrt{7/4}}\right| < 1$ $\quad$ ok.

$g'(2) = \frac{10/4}{2\sqrt{11/4}} < 1$

$g'(5) = \frac{73/4}{2\sqrt{\frac{122}{4}}}$

$\therefore \quad x = \frac{1}{2}\sqrt{x^3 - 2x + 7}$

$x = \sqrt[3]{4x^2 + 2x - 7}$

$\frac{1}{3}\frac{8x + 2}{\sqrt{4x^2 + 2x - 7}}$

1	3.8			
		4		
1.1	4.2		3.33	
		5		
1.3	5.2		0	8.33
		5		
1.4	5.7			

P_3 $\quad 3.8 + 4(x-1) + 3.33(x-1)(x-1.1) + 8.33(x-1)(x-1.1)(x-1.3)$ Center is 1.15 1.2

P_3 $\quad 5.7 + 5(x-1.4) + 5(x-1.4)(x-1.3) + 0(x-1.4)(x-1.3)(x-1.1)$ 1.25

$5.7 + 5(-.2) + 5(-.2)(-.1) = 5.7 - 1.+.$ = 4.8 //

$3.8 + 4(.2) + 3.33(.2)(.1) + 8.33(.2)(.1)(-.1)$

$(-.002)$

$4.6 + .0666 \quad -.0167$

4.65 ←

EGM 5346 SPRING 1998 DR. C. LEVY

FINAL EXAMINATION

APRIL 22, 1998

General Instructions -- This examination is 160 minutes long. Put away all your books and other material except for:

a) your notes and book
b) your calculator and straight edge.

Please sign the following:

I certify that I will neither receive nor give unpermitted aid on this examination. Violation of this will result in failure of the course and possibly other academic disciplinary actions.

_____ _____

Print your name Sign your name

This examination consists of five problems with several parts to each of the problems. You are to answer all the problems!

GOOD LUCK!

Problem #	Breakdown by Problem	Score
1	22%	
2	20%	
3	15%	
4	16%	
5	27%	
	TOTAL	

PROBLEM 1. (22%)

a) Find the 3rd degree interpolating polynomial for the following data points

x	-2.0	-1.0	0.0	1.0
f(x)	6.0	4.0	3.0	3.0

b) For the data of the table, find the Lagrange form of the 3rd degree interpolating polynomial

PROBLEM 2. (20%)

For the data given below, get the integral

$$\int_{0.1}^{1.0} f(x)\, dx$$

to the highest possible accuracy that you can find

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
f(x)	0.1	0.199	0.296	0.389	0.479	0.565	0.642	0.717	0.783	.841

PROBLEM 3 (15%)

The following experimental results for f(x) were obtained:

x	0.0	0.5	1.0	1.5	2.0	2.5	3.0
y	1.00	0.80	0.20	0.25	0.31	0.38	0.44

(2% each)

a) Use the $O(\Delta x^2)$ forward difference formula and determine f'(1.5)

b) Use the $O(\Delta x^2)$ central difference formula and determine f'(1.5)

c) Use the $O(\Delta x^2)$ backward difference formula and determine f'(1.5)

d) Explain the differences between the three values (a-c)

(3%) e) Explain how you would solve the following integral

$$I = \int_0^2 \frac{\sin x \bullet \sin (x-2)}{x \bullet (x-2)} \, dx$$

Explain in detail the difficulties in solving the integral and how to overcome them.

(4%) f) Given the following differential equation

$$y'' + 25y' + 100y = 74 \quad \text{where } y=y(t)$$

Explain how you would you determine the step size to use to solve this problem numerically.

PROBLEM 4. (16%)

Solve

$$y'' = 3y/x \quad \text{where } y = y(x)$$

given the following initial conditions:

$$x_o = 1,\ y_o = 1 \text{ and } y'_o = 1.$$

You are to use a Runge-Kutta scheme which is simple and whose error per step is $O(\Delta x^3)$. You are to find the values of y and y' at x = 0.1, and x = 0.2 using a stepsize of $\Delta x = 0.1$.

PROBLEM 5. (27%)

Given the following differential equation

$$y'' - 3\,y' + (25/4)\,y = 6.25$$

Assuming an error of $O(\Delta x^2)$ in each derivative,

(22%) a) show that the solution to the difference equation does converge to the solution of the differential equation

(5%) b) using standard methods of ODEs, find the solution to the **differential** equation

interpolating

1. a) Find the 3rd degree polynomial for the following data points

x	-2	-1	0	1
f(x)	6	4	3	3

3rd degree interpolating

b) Find the lagrange form of the polynomial

-2	6			
		-2		
-1	4		0.5	
		-1		0
0	3		0.5	
		0		
1	3			

$f(x) = 6 - 2(x+2) + 0.5(x+2)(x+1) + 0(x+2)(x+1)$

$= 6 - 2x - 4 + \frac{x^2}{2} + \frac{3}{2}x + 1 = \frac{x^2}{2} - \frac{1}{2}x + 3$ ✓

6 for div diff + 10 for f(x) = 16 parts

$-(x^3 - x) + 2(x^3 + x^2 - 2x) - \frac{3}{2}[x^3 - x + 2x^2 - 2]$

b) $f(x) = \frac{6x(x+1)(x-1)}{-6} + \frac{4(x+2)x(x-1)}{2} + \frac{3(x+2)(x+1)(x-1)}{-2}$

$+ \frac{3x(x+2)(x+1)}{6} = (-1 + 2 - \frac{3}{2} + \frac{1}{2})x^3 + x^2(2 - 3 + \frac{3}{2}) + x(1 - 4 + \frac{3}{2} + 1) + 3$ ✓

$l_j = \frac{\prod_{i \neq j}^{3}(x - x_i)}{\prod_{j \neq i}^{3}(x_j - x_i)}$

$l_0 = \frac{x(x+1)(x-1)}{(-2-(-1))(-2-0)(-2-1)}$ $l_1 = \frac{(x+2)(x)(x-1)}{(-1-(-2))(-1-0)(-1-1)}$ $l_2 = \frac{(x+2)(x+1)(x-1)}{(0-(-2))(0-(-1))(0-1)}$

$l_3 = \frac{x(x+2)(x+1)}{(1-(-2))(1-(-1))(1-0)}$ $f(x) = \sum_{i=0}^{3} x_i l_i$

0.5 for each pt

24 pts 6 for each l_i + 4 for f(x) = 28 parts

2.

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
f(x)	0.1	0.199	.296	0.389	0.479	0.565	0.642	0.717	0.783	0.841

h=.1 Simpson $\frac{h}{3}[f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + 4f_6 + 2f_7] + \frac{3}{8}h[f_7 + 3f_8 + 3f_9 + f_{10}] \leq .23013 + .22436 \leq .454$

simpson $\frac{3h}{8}[f_1 + 3f_2 + 3f_3 + 2f_4 + 3f_5 + 3f_6 + 2f_7 + 3f_8 + 3f_9 + f_{10}] = .4545$

h=.2 simpson $\frac{h}{3}[f_2 + 4f_4 + 2f_6 + 4f_8 + f_{10}] = .4396$

~~simpson $\frac{3h}{8}[f_2 + 3f_4 + 3f_6$~~

11 simp I_h
5 simp I_{2h}
4 improved
20 pts

$x_0 = 1$ $\quad p_0 = 1$ $\quad y_0 = 1$ $\quad y_{1p} = 1 + 1(.1) = 1.1$ $\quad p_{1p} = 1 + 3(.1) = 1.3$

$y_{1c} = 1 + (1 + 1.3)\,0.05 = 1.115$ $\quad p_{1c} = 1 + (1 + 1)\,1.5(.1) = 1.3$

$x_1 = 1.1$ $\quad p_1 = 1.3$ $\quad y_1 = 1.115$ $\quad y_{2p} = 1.115 + 1.3(.1) = 1.245$ $\quad p_{2p} = 1.3 + 3\left(\frac{1.115}{1.1}\right).1 = 1.6041$

$y_{2c} = 1.115 + (1.3 + 1.6041)\,.05 = 1.26021$ $\quad p_{2c} = 1.3 + \left(\frac{1.115}{1.1} + \frac{1.245}{1.2}\right)1.5(.1) = 1.607$

$x_2 = 1.2$ $\quad p_1 = 1.6077$ $\quad y_2 = 1.245$

4 for eqs + 8 comp. + 4 answ = 16 pts

5. b. ~~Given $y'' + 4y = 2$~~ Given $y'' + 4y = 2$ $\qquad y'' - 3y' + \frac{25}{4}y = \frac{25}{4}$ $\qquad y'' - 3y' + \frac{25}{4}y$ $\qquad \frac{3 \pm \sqrt{9-25}}{2} = \frac{3 \pm 4i}{2}$

~~using a second order finite difference representation~~

~~Obtain the solution~~ Show that the solution to the difference equation ~~is~~ Assuming a error of $O(\Delta x^2)$ in each derivative does converge to the solution of the differential equation

$y'' + 4y = 0 \Rightarrow y = A\sin 2x + B\cos 2x = e^{\frac{3}{2}x}\left[A\sin 2x + B\cos 2x\right]$

$y_p : y'' + 4y = 2 \quad y_p = \text{const} \quad \therefore 4C = 2 \quad C = 1/2$

$y = A\sin 2x + B\cos 2x + 1/2$

Diff 6 1 partic 4 homog

FD 6 eq 1 y_p 3 char 8 β^i (4 each) limit 4 process — 27 pts

$\frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2} + 4y_i = 2$

$\frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2} - 3\left(\frac{y_{i+1} - y_{i-1}}{2\Delta x}\right) + \frac{25}{4}y_i$

$(1 - \frac{3\Delta x}{2})y_{i+1} - 2y_i + (1 + \frac{3}{2}\Delta x)y_{i-1}$

$(-2 + \frac{25}{4}\Delta x^2)y_i$

$-16\Delta x^2 + \frac{625}{64}\Delta x^4$

$-16\Delta x^2(1 - \frac{625}{64}\Delta x^2)$

$1 - \frac{625}{128}\Delta x^2$

$-2 \pm \frac{25}{4}\Delta x^2 \pm \sqrt{(4 - 25\Delta x^2 + \frac{625}{16}\Delta x^4}$

$-2 \pm \frac{25}{4}\Delta x^2 \pm \sqrt{}$ $\quad -4(1 - \frac{3}{4}\Delta x^2)$

$y_{i+1} + (-2 + 4\Delta x^2)y_i + y_{i-1} = 2\Delta x^2$

for particular let $y_{i+1} = y_i = y_{i-1} = A \Rightarrow A + (4\Delta x^2 - 2)A + A = 2\Delta x^2 = 4\Delta x^2 A = 2\Delta x^2$ or $A = 1/2$

for other $\beta^{i+1} + (-2 + 4\Delta x^2)\beta^i + \beta^{i-1} = 0 \qquad \beta^{i-1}\left[\beta^2 + (-2 + 4\Delta x^2)\beta + 1\right] = 0$

$\beta = \frac{2 - 4\Delta x^2 \pm \sqrt{(2 - 4\Delta x^2)^2 - 4}}{2} = \frac{2 - 4\Delta x^2 \pm \sqrt{4 - 16\Delta x^2 + 16\Delta x^4} - 4}{2}$

$= 1 - 2\Delta x^2 \pm \frac{4\Delta x\sqrt{-1 + \Delta x^2}}{2}$

$= 1 - 2\Delta x^2 \pm \frac{4\Delta x i}{2}\left(1 - \frac{\Delta x^2}{2} + \dots\right)$

$\approx 1 \pm 2\Delta x i$

$\frac{-2 \pm \frac{25}{4}\Delta x^2 \pm (2\Delta x i - \frac{625}{128}\Delta x^3 i)}{1 - \frac{3}{2}\Delta x} \sim (-1 \pm 2\Delta x i)(1 + \frac{3}{2}\Delta x) \sim (-1 + \frac{3}{2}\Delta x \pm 2\Delta x i) \Rightarrow e^{\frac{3}{2}x_i \pm 2jx_i}$

$\beta = (1 + 2\Delta x j) \qquad \beta = (1 - 2\Delta x j)$

$= 1(1 + 2\frac{x_i}{i}j) \qquad = 1(1 - 2\frac{x_i}{i}j)$

$e^{i\,2x_i j} \qquad e^{2jx_i} \qquad e^{i(1 - 2x_i j)\frac{-i\,2x_i j}{-2x_i}} \qquad e^{-2jx_i}$

thus $y_i \to C_1 e^{2jx_i} + C_2 e^{-2jx_i} + 1/2$ since $e^{\pm axi} = \cos ax \pm i \sin ax$

or $\bar{C}_1 \sin 2x_i + \bar{C}_2 \cos 2x_i + 1/2$

(f) $y'' + 25y' + 100y = 74$ char eq $(s^2 + 25s + 100) = 0$

$(s+20)(s+5) = 0$

e^{-20t} e^{-5t}

$\Delta t = \frac{1}{20}$ $\Delta t = \frac{1}{5}$

use this

(e) $\int_0^2 \frac{\sin x}{x} \cdot \frac{\sin(x-2)}{x-2}\, dx$ problem @ $x=0$ & $x=2$

@ $x=0$ $\frac{\sin x}{x} \sim \cos x$ or $1 - \frac{x^2}{2}$

@ $x=2$ $\frac{\sin x+2}{x-2} \sim \cos(x-2) \sim 1 - \frac{(x-2)}{2}$

removable singularity at $x=0, 1$

$f'(x)_{CD} = \frac{y_{i+1} - y_{i-1}}{2\Delta x} = \frac{[.31 - .2]}{1} = .11$ (b)

$-y_{i+2} = -y_i + y_i' \cdot 2\Delta x + y_i'' \frac{4\Delta x^2}{2}$

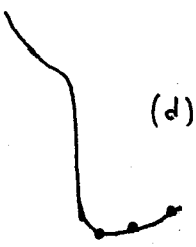
$4y_{i+1} = 4y_i + 4y_i' \Delta x + 4y_i'' \frac{\Delta x^2}{2}$

$-3y_i = -3y_i$

$+2y_i' \Delta x$

$f'(x)_{FD} = \frac{-y_{i+2} + 4y_{i+1} - 3y_i}{2\Delta x} = \frac{-.38 + 4(.31) - .75}{1} = \frac{1.24 - 1.13}{1} = .11$ (a)

$f'(x)_{BD} = \frac{3y_{i-2} - 4y_{i-1} + 3y_i}{2\Delta x} = \frac{.8 - 4(.2) + .75}{1} = \frac{.75}{1} = .75$?? (c)



(d) → FD + CD give good results since changes are not rapid

→ BD doesn't due to rapid changes.

$y' = p = f(x,y,p)$ $\qquad y_{i+1} = y_i + \frac{1}{2}(k_1 + k_2)$

$y'' = p' = \frac{3y}{x} = g(x,y,p)$ $\qquad p_{i+1} = p_i + \frac{1}{2}(l_1 + l_2)$

$k_1 = h f(x_i, y_i, p_i) = h p_i$

$k_2 = h f(x_i + \Delta x,\ y_i + k_1,\ p_i + l_1) = h(p_i + l_1)$

$l_1 = h g(x_i, y_i, p_i) = h\left(\frac{3y_i}{x_i}\right)$

$l_2 = h g(x_i + \Delta x,\ y_i + k_1,\ p_i + l_1) = h\left(\frac{3(y_i + k_1)}{x_i + \Delta x}\right)$

$y_{i+1} = y_i + \int p\,dx = y_i + \frac{1}{2}(k_1 + k_2)$ $\qquad \therefore\ k \leftrightarrow f$

$p_{i+1} = p_i + \int \frac{3y}{x}\,dx = p_i + \frac{1}{2}(l_1 + l_2)$ $\qquad l \leftrightarrow g$

$y_{i+1} = y_i + \frac{1}{2}\left[h p_i + h(p_i + l_1)\right]$ $\qquad p_{i+1_p} = p_i + l_1 = p_i + h\left(\frac{3y_i}{x_i}\right)$

$p_{i+1} = p_i + \frac{1}{2}\left[h\frac{3y_i}{x_i} + h\frac{3(y_i + k_1)}{x_i + \Delta x}\right]$ $\qquad y_{i+1_p} = y_i + k_1 = y_i + h p_i$

$I_{imp} = I_h + Ch^4$

$\hat{I}_{imp} = I_{2h} + 16Ch^4$

$$I_{imp} = I_h + \frac{(\hat{I}_{imp} - I_{2h})}{16}$$

$$16(I_{imp}) - \hat{I}_{imp} = 16\,I_h - I_{2h}$$

$$I_{imp}$$

$$I_{imp} = \frac{16\,I_h - I_{2h}}{15} = \frac{16(.4545) - .4396}{15}$$

$$\cong .4555$$

$$\hat{I}_{imp} = I_{2h} + 16\,(I_{imp} - I_h) \Rightarrow \hat{I}_{imp} - 16 I_{imp} = I_{2h} - 16 I_h$$

3. Use problem on finding derivatives from previous test

Given $y' = \frac{3y}{x}$ where $y = \Phi$ $x = 1$

$$\frac{y'}{y} = \frac{3}{x} \quad \text{or} \quad \ln y = 3\ln x + \ln C$$

$$y = Cx^3 \quad y' = 3Cx^2$$

$y = 1$ $x = 1$ $C = 1$

Using a representation for y' that is $O(\Delta x)^2$

$$\frac{y_{i+1} - y_{i-1}}{2\Delta x} = \frac{3y_i}{x_i} = \frac{3y_i}{i\Delta x}$$

$$i(y_{i+1} - y_{i-1}) = 3y_i \quad \text{or} \quad i y_{i+1} - 3y_i - i y_{i-1} = 0$$

let $y_i = \beta^i$

$$i\beta^{i+1} - 3\beta^i - i\beta^{i-1} = 0$$

$$\beta^{i-1}[i\beta^2 - 3\beta - i] = 0$$

$$\beta = \frac{3 \pm \sqrt{9 + 4i^2}}{2i}$$

4. Solve $y'' = \frac{3y}{x}$ with initial values @ $x = 1$ $y = 1$ $y' = 1$ using a RK scheme whose error per step is $O(\Delta x$

Find the values of y & y' @ $x = .1$ & $x = .2$ for $\Delta x = 0.1$

same as modified euler

$y' = p \Rightarrow y_{i+1,p} = y_i + p_i \Delta x$ $\quad y_{i+1,c} = y_i + (p_i + p_{i+1,p})\frac{\Delta x}{2}$

$p' = \frac{3y}{x}$ $\quad p_{i+1} = p_i + \frac{3y_i}{x_i}\Delta x$ $\quad p_{i+1} = p_i + \left(\frac{y_i}{x_i} + \frac{y_{i+1,p}}{x_{i+1}}\right)\frac{3\Delta x}{2}$

Florida International University
Department of Mechanical Engineering

EGM 5346 EXAMINATION NO. 1 February 26, 1997

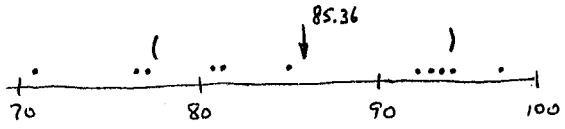
This exam is a take-home exam and must be returned by 1200 hours on 28 February 1997. Do all work on engineering paper and turn in all computer printouts. Show all work!!!!!!

Print your name and sign the following statement:

I will not give nor take any unpermitted aid during this examination. I understand that violation of this statement will lead to automatic failure of the examination.

PRINT NAME _____ SIGN NAME _____

PROBLEM 1	15 PTS	_____
PROBLEM 2	35 PTS	_____
PROBLEM 3	25 PTS	_____
PROBLEM 4	25 PTS	_____



1. Find the roots of the equation

$$f(x) = \sin x - |\ln x|$$

Indicate the numerical method you used, the starting point of the iterations, the number of iterations and the epsilon used. The roots must be accurate to five places after the decimal point.

Note: start at $x = 0.1$

$x = .57871$ answer

2.21911

2. Solve the system $Ax = b$ where

$$A = \begin{vmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{vmatrix} \qquad b = \begin{vmatrix} 1.00000 \\ 0.58333 \\ 0.43333 \\ 0.35000 \end{vmatrix}$$

a) Use the LU decomposition and give me the entries of the matrices L and U, and the solution **x**.

b) Finally obtain the LARGEST eigenvalue of A and its corresponding eigenvector by

1) using the Fadeev-Laverrier method (you have a program for it) to get the characteristic polynomial and find the roots of the polynomial;

2) then solve the equation $Ax = v\,x$ by the power method to get v and **x**.

Use as your first guess for **x**, $y^T = [1,1,0,1]$. If you do not converge to the first eigenvalue and eigenvector, give me a good reason why this is happening and, if you can, prove your reason in a simple manner.

c) Find A^{-1} and then determine the condition number for A using the infinity norm.
REMEMBER cond(A) = $||A||\ ||A^{-1}||$

$\underset{\sim}{x}$: [1, -2, 3, 0] 4

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ .5 & .0833 & 0 & 0 \\ .\bar{3} & .0833 & .005 & 0 \\ .25 & .075 & .008\bar{3} & .000357 \end{bmatrix} \begin{bmatrix} 1 & .5 & .\bar{3} & .25 \\ 0 & 1 & 1 & .9 \\ 0 & 0 & 1 & 1.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ 16

(1/12, 1/12, 1/180; $\frac{3}{40}$, $\frac{1}{120}$, $\frac{1}{2800}$)

A^{-1}

$$\begin{bmatrix} 16 & -120 & 240 & -140 \\ -120 & 1200 & -2700 & 1680 \\ 240 & -2700 & 6480 & -4200 \\ -140 & 1680 & -4200 & 2800 \end{bmatrix}$$ 16

Coeffs of Faddeev $(-1)^4[\lambda^4 - 1.67619\lambda^3 - (-.265159)\lambda^2 - (.001735)\lambda + .1643242 \times 10^{-6}] = 0$ 4

$\lambda_1 = .16914$
$\lambda_2 = 1.5002138$
$\lambda_3 = ~~0~~$ .0000967 4
$\lambda_4 = ~~.006838~~$ .0067383

Power $\lambda = ~~1.522789~~$ 1.5002 $\underline{v} = (1, ~~.5?4?~~ .570164, ~~.469?03~~ .40678, ~~.32????~~ .318141?)$ 4

$||A||_\infty = 2.083333$ $||A^{-1}||$ ~~13619.80566~~ 13620. Cond A = ~~28374.5973~~ 28375 3

3. For the following data points:

(1.7,-0.1) (2.3,0.2) (6.2,6.6) (0.1,0.0) (8.5,13.8) (-1.0,1.2)

Find the least squares fit in the form

$$f(x) = a + b\,x + c\,x^2$$

$= .2087 - .6001x + .2597\,x^2$

4. The US Gross National Product (GNP) for the years 1950-1980 are given below.

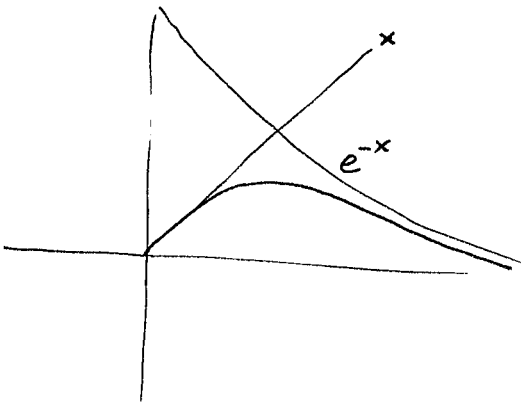
YR	Trillions of Dollars
1950	0.530
1955	0.656
1960	0.733
1965	0.934
1970	1.091
1975	1.230
1980	1.484

a) Determine the divided difference table

b) Using a 3rd order polynomial starting with 1950, predict the GNP for 1965.

c) Using a fourth order polynomial starting with the year 1955, determine the GNP for 1963

find the best fit to the following data (plot the data first to get an idea of what might be an appropriate function



$k_1 x e^{-k_2 x}$ +

$\ln k_1 x$ + $k_2 x$

$f(x) = c + ax + bx^2$

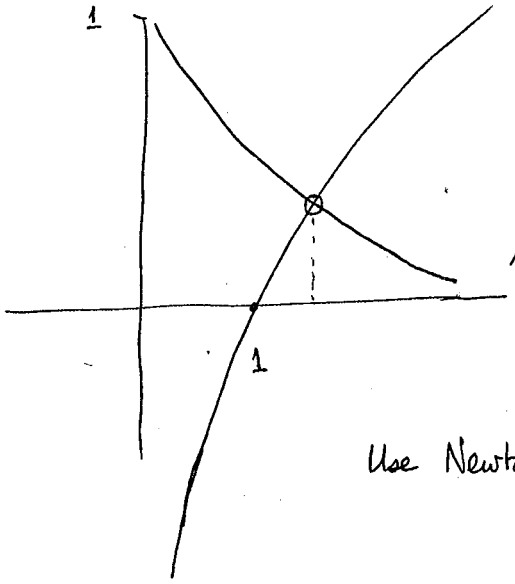
given (1.7, -0.1) (2.3, 0.2) (6.2, 6.6) (0.1, 0.0)
(8.5, 13.8) (-1.0, 1.2)

The US GNP for the years 1950-1980 are given

Yr	Trillions of dollars
1950	0.530
1955	0.656
1960	0.733
1965	0.934
1970	1.091
1975	1.230
1980	1.484

- Determine the divided difference table
- Using a 3rd order polynomial starting with 1960 predict the GNP of 1965
- Using a 4th order polynomial starting with the year 1955 predict the GNP of 1963

1



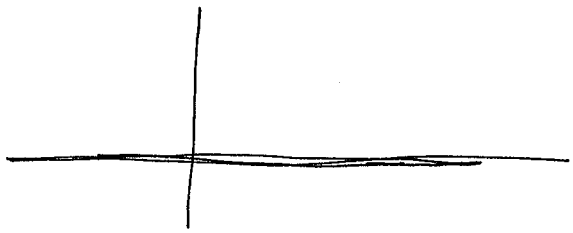
$\sin x - \ln \frac{x}{2} + e^{-x} = f(x)$

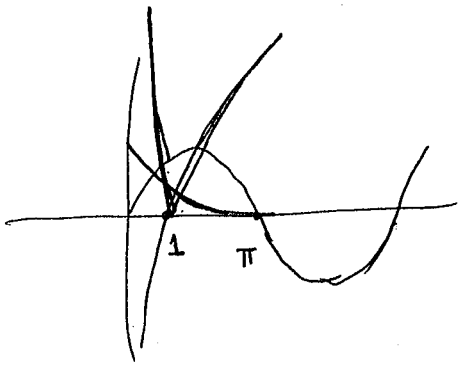
$\frac{1}{2x} + e^{-x} = f'(x)$

$\sin x - \ln \frac{x}{2} + e^{-x}$

Use Newton Raphson method and ~~[illegible]~~ find the root of $\ln x - e^{-x}$

$\ln \pi/2$





$f(x) = \sin x - \frac{|\ln x|}{[illegible]}$

2.

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1.083 \\ [illegible] \\ 0.450 \\ [illegible] \end{bmatrix}$$

Solve $Ax=b$ where $A = [\,]$

find the ~~[illegible]~~ LU decomposition of A

- Solve $Ax=b$ to get $\underline{x}$
- Now let $\underline{x}$ be the first approximation to $\underline{x}$ and solve $A\underline{e} = \underline{b} - A\underline{x} = \underline{r}$
- find A^{-1}

$12 - 6 + 4 + 3 = 13/12$

$30 - 20 + 15 + 12 = 37/60 = 1.17/60 =$

$20 - 15 + 12 + 10 = 27/60 = 9/20 \quad 19/20 \quad .950$

$210 + 168 + 140 + 120 = 638/840$

$1/840$

$1 + .5 + .33 + .25 = 2.083$

$.333 + .25 + .2 + .167 = .95$

$.5$

$1 - 2 \quad 3 \quad 0$

$12 - 12 + 12 + 0 = 1$

$30 - 40 + 45 + 0 = 35/60 = .5833$

$20 - 30 + 36 + 0 = 26/60 = .4333$

$210 - 336 + 420 = 294/840 = .35$

$.01666 \quad .35$

$$60 \overline{)1.000}$$
$$.60$$
$$.40$$
$$.36$$
$$.040$$

- what is the condition number for A

[illegible]

```
A( 1, 3) =  .0000000E+00
A( 1, 4) =  .0000000E+00
A( 1, 5) =  .1000000E+01
A( 2, 1) =  .0000000E+00
A( 2, 2) =  .1000000E+01
A( 2, 3) =  .0000000E+00
A( 2, 4) =  .0000000E+00
A( 2, 5) =  .1000000E+01
A( 3, 1) =  .0000000E+00
A( 3, 2) =  .0000000E+00
A( 3, 3) =  .1000000E+01
A( 3, 4) =  .0000000E+00
A( 3, 5) =  .0000000E+00
A( 4, 1) =  .0000000E+00
A( 4, 2) =  .0000000E+00
A( 4, 3) =  .0000000E+00
A( 4, 4) =  .1000000E+01
A( 4, 5) =  .2000000E+01


    X(1)  THROUGH X(N)

       .1000000E+01
       .1000000E+01
       .0000000E+00
       .2000000E+01
 4 SIMULTANEOUS EQS. ARE TO BE SOLVED

AUGMENTED MATRIX ELEMENTS ARE:
A( 1, 1) =  .1000000E+01
A( 1, 2) =  .5000000E+00
A( 1, 3) =  .3333333E+00
A( 1, 4) =  .2500000E+00
A( 1, 5) =  .1000000E+01
A( 2, 1) =  .5000000E+00
A( 2, 2) =  .3333333E+00
A( 2, 3) =  .2500000E+00
A( 2, 4) =  .2000000E+00
A( 2, 5) =  .5833333E+00
A( 3, 1) =  .3333333E+00
A( 3, 2) =  .2500000E+00
A( 3, 3) =  .2000000E+00
A( 3, 4) =  .1666667E+00
A( 3, 5) =  .4333333E+00
A( 4, 1) =  .2500000E+00
A( 4, 2) =  .2000000E+00
A( 4, 3) =  .1666667E+00
A( 4, 4) =  .1428571E+00
A( 4, 5) =  .3500000E+00


    X(1)  THROUGH X(N)

       .1000008E+01
      -.2000081E+01
       .3000181E+01
      -.1128588E-03
 3 SIMULTANEOUS EQS. ARE TO BE SOLVED

AUGMENTED MATRIX ELEMENTS ARE:
```

7 place accuracy

EGM 5346 SPRING 1999 DR. C. LEVY

FINAL EXAMINATION

APRIL 21, 1999

General Instructions -- This examination is 120 minutes long. Put away all your books and other material except for:

a) your notes and book
b) your calculator and straight edge.

Please sign the following:

I certify that I will neither receive nor give unpermitted aid on this examination. Violation of this will result in failure of the course and possibly other academic disciplinary actions.

_____ _____

Print your name Sign your name

This examination consists of five problems with several parts to each of the problems. You are to answer all the problems!

GOOD LUCK!

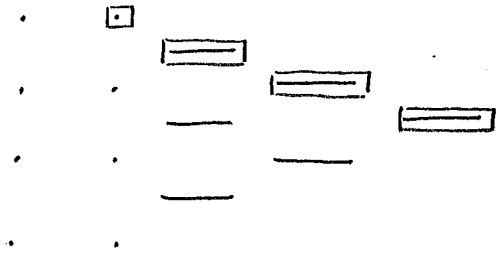
Problem #	Breakdown by Problem	Score
1	25%	
2	25%	
3	25%	
4	25%	
	TOTAL	

PROBLEM 1. (25%)

x	-2.0	-1.0	0.0	1.0
f(x)	6.0	4.0	3.0	3.0

a) For the data of the table, find the Lagrange form of the 3rd degree interpolating polynomial

b) Find the 3rd degree interpolating polynomial for the same data points



$6 + eq\ (ax^3 + bx^2 + cx + d) = 10$

$l_0 y_0 + l_1 y_1 + l_2 y_2 + l_3 y_3$

$l_0 = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_1-x_0)(x_2-x_0)(x_3-x_0)}$ $\quad l_1 = \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_0-x_1)(x_2-x_1)(x_3-x_1)}$ $\quad l_2 = \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_0-x_2)(x_1-x_2)(x_3-x_2)}$

$l_3 = \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_0-x_3)(x_1-x_3)(x_2-x_3)}$ $\quad \frac{24}{2}+4 = 16$ $\quad \therefore$ 3 each + 4 for final

PROBLEM 2. (25%)

a) For the data given below, get the integral

$$\int_{0.1}^{1.0} f(x)\,dx$$

to the highest possible accuracy that you can find

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
f(x)	0.1	0.199	0.296	0.389	0.479	0.565	0.642	0.717	0.783	.841

Use simpson $\frac{1}{3}$ rule for $\int_{0.1}^{0.7}$ + $\frac{3}{8}$ rule for $\int_{0.7}^{1.0}$ ← 13

b) Explain how you would solve the following integral

$$I = \int_0^2 \frac{\sin x \bullet \sin(x-2)}{x \bullet (x-2)}\,dx$$

$x=0$ $x=2$ use $\frac{\sin x}{x} \sim 1$ @ $x=0$ 3

use $\frac{\sin(x-2)}{x-2} \sim 1$ @ $x=2$ 3

Explain in detail the difficulties in solving the integral and how to overcome them.

c) Given the following differential equation

$$y'' + 25y' + 100y = 74 \quad \text{where } y=y(t)$$

Explain how you would you determine the step size to use to solve this problem numerically.

$s^2+25s+100=0$ $s = \frac{-25 \pm \sqrt{25^2-4\cdot 100}}{2} = \frac{-25 \pm 15}{2} = -5$ & -20 3

use $\Delta t = \frac{1}{20} = .05$ 3

PROBLEM 3. (25%)

Solve

$$y'' = 3y/x \quad \text{where } y = y(x)$$

given the following initial conditions:

$$x_0 = 1,\ y_0 = 1 \text{ and } y'_0 = 1.$$

You are to use a Runge-Kutta scheme which is simple and whose error per step is $O(\Delta x^3)$. You are to find the values of y and y' at x = 1.1, and x = 1.2 using a stepsize of $\Delta x = 0.1$.

let $y' = p \Rightarrow y_1 = y_0 + \int p\,dx$

$p' = 3y/x \Rightarrow p_1 = p_0 + \int (3y/x)\,dx$

$k_0 = \Delta x\, p_0 \qquad q_1 = \Delta x\,(3y_0/x_0)$

$k_1 = \Delta x\, p_1 \qquad q_2 = \Delta x\,(3(y_0 + q_1)/x_0 + \Delta x)$

pts

4 ⇒ 8

$y_1 = y_0 + \frac{1}{2}(k_0 + k_1) = y_0$

$p_1 = p_0 + \frac{1}{2}(q_0 + q_1)$

2 ⇒ 4

$k_0 \qquad q_1$

$k_1 \qquad q_2$

4 ⇒ 8

$y_2 = y_1 + \frac{1}{2}(k_0 + k_1)$

$p_2 = p_1 + \frac{1}{2}(q_0 + q_1)$

2 ⇒ 4

24 pts

PROBLEM 4. (25%)

Given the following differential equation

$$y'' - 3y' + (25/4)\,y = 6.25$$

and assuming an error of $O(\Delta x^2)$ in each derivative, show that the solution to the difference equation does converge to the solution of the differential equation.

3

$s^2 - 3s + 25/4 = 0$ 3 $\quad \frac{3 \pm \sqrt{9 - 4\cdot(25/4)}}{2} = \frac{3 \pm 4i}{2} = \frac{3}{2} \pm 2i$ $\quad y_{omo} = C_1 e^{\frac{3}{2}x} + C_2 e^{\bar{\frac{3}{2}}x} + 1$

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2} - 3\left(\frac{y_{i+1} - y_{i-1}}{2\Delta x}\right) + \frac{25}{4} y_i = 6.25 \qquad 7$$

$y_{i+1} = y_i + \Delta x\, y_i' + \frac{\Delta x^2}{2!} y_i'' +$

$y_{i-1} = y_i - \Delta x\, y_i' + \frac{\Delta x^2}{2!} y_i'' -$

$\frac{y_{i+1} - y_{i-1}}{2\Delta x} = 0 + y_i' + 0 + \frac{\Delta x^2}{6} y_i'''$

$$y_{i+1}\left(1 - \frac{3\Delta x}{2\Delta x}\right) - y_i\left(2 - \frac{25}{4}\Delta x^2\right) + y_{i-1}\left(1 + \frac{3\Delta x}{2\Delta x}\right) = 6.25\Delta x^2$$

for particular $y_i = C \Rightarrow C = 1$

let $y_i = C\beta^i$ for homog. eq.

$$C\beta^{i-1}\left[\left(1 - \frac{3}{2}\Delta x\right)\beta^2 - \left(2 - \frac{25}{4}\Delta x^2\right)\beta + \left(1 + \frac{3}{2}\Delta x\right)\right] = 0 \qquad 6$$

$$\beta = \frac{\left(2 - \frac{25}{4}\Delta x^2\right) \pm \sqrt{\left(2 - \frac{25}{4}\Delta x^2\right)^2 - 4\left(1 - \frac{3\Delta x}{2}\right)\left(1 + \frac{3}{2}\Delta x\right)}}{2\left(1 - \frac{3\Delta x}{2}\right)}$$

$4 - 25\Delta x^2 + \frac{625}{16}\Delta x^4 - 4 + 4\frac{9}{4}\Delta x^2$

$\beta \cong 1 + \frac{3}{2}\Delta x \pm i\, 2\Delta x$ $\qquad -16\Delta x^2 + \frac{625}{16}\Delta x^4$

$\left[2 - \frac{25}{4}\Delta x^2 \pm 4i\Delta x\left[\sqrt{1 - \frac{625}{256}\Delta x^2}\right](1 + \frac{3\Delta x}{2})\right]$

$\beta^i = \left(1 + \left(\frac{3}{2} \pm 2i\right)\Delta x\right)^i$ 4 $\qquad \left[1 - \frac{25}{8}\Delta x^2 \pm 2\Delta x i\sqrt{\quad}\right](1 + \frac{3\Delta x}{2})$

$\left(1 + 5\frac{x_i}{i}\right)^{\frac{i}{5}x_i}$

$e^{5x_i} = e^{(3/2 \pm 2i)x_i}$ 2 $\qquad 1 + \frac{3\Delta x}{2} \pm 2\Delta x i$

$1 + \Delta x\left(\frac{3}{2} \pm 2i\right)$

GR GR

$$\frac{\frac{3}{4}a + \frac{1}{4}b}{1} = GR$$

$$\frac{4}{5}(a+b) = \frac{4}{5}a + \frac{4}{5}b.$$

Florida International University
Department of Mechanical Engineering

EGM 5346 FINAL EXAMINATION-PART II April 21, 1999

This part of the exam is take-home and must be returned by 1400 hours on 23 April 1997 to EAS3442. Do all work on engineering paper and turn in all computer printouts. Show all work!!!!!!

Print your name and sign the following statement:

I will not give nor take any unpermitted aid during this examination. I understand that violation of this statement will lead to automatic failure of the examination.

_____ _____

PRINT NAME SIGN NAME

PROBLEM 25 PTS _____

$y' = x^2 y^2$ Using Xcel

0.1

xi	yi act	yi	$y'I = x_i^2 y_i^2$	xi+1	$yi+1 = y_i + y_i' \Delta x$	$y'I+1 = x_{i+1}^2 y_{i+1}^2$
0	1	1	0	0.1	1	0.01
0.1	1.00033344	1.0005	0.01001	0.2	1.001501	0.04012017
0.2	1.0026738	1.00300651	0.04024088	0.3	1.007030597	0.09126996
0.3	1.00908174	1.00958205	0.09173303	0.4	1.018755354	0.166058
0.4	1.02179837	1.0224716	0.16727171	0.5	1.039198773	0.26998352
0.5	1.04347826	1.04433436	0.27265857	0.6	1.07160022	0.41339773
0.6	1.07758621	1.07863718	0.41884494	0.7	1.120521672	0.61522872
0.7	1.12909296	1.13034086	0.62605853	0.8	1.192946714	0.91079799
0.8	1.20578778	1.20718369	0.93266717	0.9	1.300450404	1.36984872
0.9	1.32100396	1.32230948	1.41628692	1	1.463938173	2.14311497
1	1.5	1.50027958	2.25083881	1.1	1.725363457	3.60202366
1.1	1.79748352	1.7929227	3.88963189	1.2	2.181885888	6.85530148
1.2	2.35849057	2.33016937	7.81875257	1.3	3.112044624	16.3673487
1.3	3.73599004	3.53947443	21.172116	1.4	5.656686029	62.7162698
1.4	11.71875	7.73389372	117.2337	1.5	19.45726369	851.816498
	3/(3-A19^3)	H18	A19^2*C19^	A19+0.1	C19+D19*A1	E19^2*F19^2

0.05

0	1	1	0	0.05	1	0.0025
0.05	1.00004167	1.0000625	0.00250031	0.1	1.000187516	0.01000375
0.1	1.00033344	1.0003751	0.0100075	0.15	1.000875477	0.02253941
0.15	1.00112627	1.00118877	0.02255353	0.2	1.002316451	0.04018553
0.2	1.0026738	1.00275725	0.04022088	0.25	1.004768295	0.06309746
0.25	1.0052356	1.00534021	0.06316931	0.3	1.008498675	0.09153626
0.3	1.00908174	1.00920785	0.09166504	0.35	1.013791101	0.12590212
0.35	1.01449888	1.01464703	0.1261148	0.4	1.020952768	0.16677513
0.4	1.02179837	1.02196928	0.16710739	0.45	1.030324646	0.2149677
0.45	1.03132654	1.03152115	0.21546727	0.5	1.042294517	0.27159446
0.5	1.04347826	1.0436977	0.27232622	0.55	1.057314008	0.33816866
0.55	1.05871454	1.05896007	0.33922242	0.6	1.075921189	0.41673831
0.6	1.07758621	1.07785909	0.41824088	0.65	1.09877113	0.5100834
0.65	1.10076595	1.10106719	0.51221744	0.7	1.126678066	0.6220077
0.7	1.12909296	1.12942282	0.625042	0.75	1.160674922	0.75778103
0.75	1.16363636	1.1639934	0.76212035	0.8	1.202099415	0.92482752
0.8	1.20578778	1.20616709	0.931097	0.85	1.252721945	1.13382812
0.85	1.25740032	1.25779022	1.14302119	0.9	1.314941282	1.40054717
0.9	1.32100396	1.32137943	1.41429532	0.95	1.392094197	1.74897844
0.95	1.40015168	1.40046128	1.77006583	1	1.488964567	2.21701548
1	1.5	1.50013831	2.25041494	1.05	1.612659055	2.86723782
1.05	1.62833299	1.62807963	2.92233421	1.1	1.774196338	3.8088049
1.1	1.79748352	1.79635811	3.90455195	1.15	1.991585703	5.245582
1.15	2.02822615	2.02511145	5.42367354	1.2	2.296295131	7.59307871
1.2	2.35849057	2.35053026	7.95598921	1.25	2.748329721	11.8020566

1.25	2.86567164	2.84448141	12.6423039	1.3	3.4765966	20.4265634
1.3	3.73599004	3.67120309	22.7773673	1.35	4.810071453	42.166795
1.35	5.55941626	5.29480715	51.093756	1.4	7.849494945	120.764559
1.4	11.71875	9.59126502	180.305035	1.45	18.60651676	727.890684

$y_{i+1} = y_i + (y'_i + y'_{i+1})\,\Delta x/2$

1.0005
1.003006509
1.009582051
1.022471602
1.044334363
1.078637178
1.130340861
1.207183687
1.322309482
1.500279576
1.792922699
2.330169368
3.539474433
7.73389372
56.1864036

C19+(D19+G19)*A1/2

1.0000625
1.000375102
1.001188775
1.002757251
1.005340209
1.009207849
1.014647028
1.021969276
1.031521153
1.043697697
1.058960069
1.077859087
1.101067194
1.129422822
1.163993398
1.206167095
1.257790222
1.321379431
1.400461275
1.500138308
1.628079627
1.796358105
2.025111454
2.35053026
2.844481407

3.671203089
5.294807146
9.591265019
32.296158