

UMAP

**Modules in
Undergraduate
Mathematics
and Its
Applications**

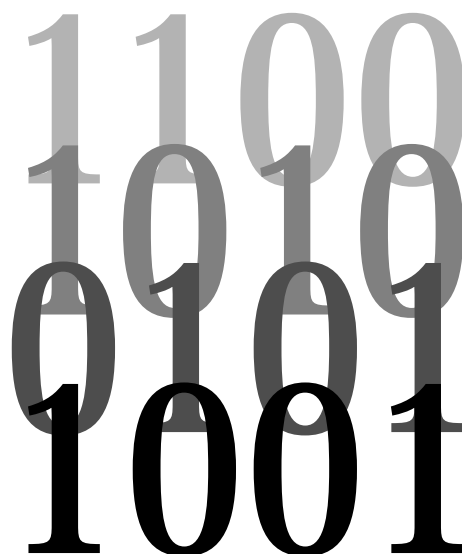
**Published in
cooperation with
the Society
for Industrial
and Applied
Mathematics, the
Mathematical
Association of
America, the
National Council
of Teachers of
Mathematics,
the American
Mathematical
Association of
Two-Year Colleges,
The Institute
of Management
Sciences, and the
American Statistical
Association.**

The logo for COMAP, Inc., featuring the word "COMAP" in a stylized, cursive script font.

Module 708

Computer and Calculator Computation of Elementary Functions

**Richard J. Pulskamp
James A. Delaney**

A graphic consisting of a 3x4 grid of binary digits (0s and 1s). The digits are rendered in a large, bold, serif font. The top row contains the digits 1, 1, 0, 0. The middle row contains 1, 0, 1, 0. The bottom row contains 0, 1, 0, 1. The digits are arranged in a way that they appear to be floating or stacked, with some digits overlapping.

**Applications of Numerical
Analysis and Calculus
to Computation**

INTERMODULAR DESCRIPTION SHEET:	UMAP Unit 708
TITLE:	Computer and Calculator Computation of Elementary Functions
AUTHOR:	Richard J. Pulskamp and James A. Delaney Dept. of Mathematics and Computer Science Xavier University 3800 Victory Parkway Cincinnati, OH 45207
MATHEMATICAL FIELD:	Numerical analysis, Calculus
APPLICATION FIELD:	Computation
TARGET AUDIENCE:	Second-semester calculus students, computer science students
ABSTRACT:	We consider methods used to approximate the elementary functions on digital computers and electronic calculators. The requirements for such approximations are discussed, with brief comments on hardware and error. We emphasize range reduction, polynomial approximations, and especially CORDIC techniques. We treat the square root, trigonometric, exponential, and logarithmic functions in detail.
PREREQUISITES:	Calculus through Taylor's formula. Some of the exercises assume access to a graphing calculator or a computer graphing program. We recommend use of a calculator and a computer in connection with this Module.

This work appeared in *The UMAP Journal* 12 (4) (1991) 315–348 and in *UMAP Modules: Tools for Teaching 1991*, edited by Paul J. Campbell, 1–34. Lexington, MA: COMAP, 1992.

©Copyright 1991, 1992, 1999 by COMAP, Inc. All rights reserved.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice. Abstracting with credit is permitted, but copyrights for components of this work owned by others than COMAP must be honored. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior permission from COMAP.

COMAP, Inc., Suite 210, 57 Bedford Street, Lexington, MA 02173
(800) 77-COMAP = (800) 772-6627, or (781) 862-7878; <http://www.comap.com>

Computer and Calculator Computation of Elementary Functions

Richard J. Pulskamp
James A. Delaney

Dept. of Mathematics and Computer Science
Xavier University
3800 Victory Parkway
Cincinnati, OH 45207

Table of Contents

1. INTRODUCTION	1
2. BACKGROUND INFORMATION	2
2.1 Hardware Aspects	2
2.1.1 The base used for number representation	3
2.1.2 Integer vs. real-number representation and arithmetic	4
2.1.3 Hardware vs. software implementation of operations	4
2.2 Requirements of Function Evaluation Routines	5
2.3 Error Considerations	5
3. THE SQUARE ROOT	7
3.1 Calculator Computation of Square Root	7
3.2 Computer Computation of the Square Root	9
4. ELEMENTARY FUNCTIONS ON COMPUTERS	10
4.1 Preliminary Remarks	10
4.2 Computation of Elementary Functions	12
4.2.1 Computation of the exponential function	12
4.2.2 Evaluation of the natural logarithm	13
5. ELEMENTARY FUNCTIONS ON CALCULATORS	14
5.1 Sines, Cosines, and Tangents via CORDIC	15
5.1.1 The CORDIC algorithm for tangent	16
5.1.2 The CORDIC computation of sines and cosines	20
5.2 A CORDIC Algorithm for Arctangent	20
5.2.1 Derivation of the CORDIC algorithm for arctangent	22
6. SOLUTIONS TO THE EXERCISES	23

REFERENCES	26
ACKNOWLEDGMENTS	28
ABOUT THE AUTHORS	28

MODULES AND MONOGRAPHS IN UNDERGRADUATE MATHEMATICS AND ITS APPLICATIONS (UMAP) PROJECT

The goal of UMAP is to develop, through a community of users and developers, a system of instructional modules in undergraduate mathematics and its applications, to be used to supplement existing courses and from which complete courses may eventually be built.

The Project was guided by a National Advisory Board of mathematicians, scientists, and educators. UMAP was funded by a grant from the National Science Foundation and now is supported by the Consortium for Mathematics and Its Applications (COMAP), Inc., a nonprofit corporation engaged in research and development in mathematics education.

Paul J. Campbell
Solomon Garfunkel

Editor
Executive Director, COMAP

1. Introduction

We routinely tackle computational problems with the aid of calculators or by programming a computer. In the process, we assume the availability of rapid and accurate evaluation of such elementary functions as \sqrt{x} , $\exp x$, $\cos x$, and $\arctan x$.

It may come as a surprise that sometimes the values reported by a machine are inaccurate. For example, BASICA on an IBM PC returns $\ln 1.001$ as 9.994461E-4 instead of the correct value 9.995003E-4; only the first three digits are correct! Yet, the little Casio fx-7000G graphing calculator gives 9.995003331E-4 with all ten digits correct. Obviously, not everyone is doing the same thing.

How the elementary functions can be calculated accurately, rather than having inaccuracies in some existing routines, is the main focus of this Module. In truth, the “obvious” way to compute the value of a function is usually not the best way. As a result, computer and calculator manufacturers have implemented many interesting algorithms for this purpose.

To illustrate how calculations are really performed, we will describe some algorithms in current use. For computer routines, we draw upon the run-time library of the Digital Equipment Corporation VAX series of computers. For calculators, we discuss routines used by Hewlett-Packard in its calculators.

The choice of method for function evaluation is dependent upon many factors. The most important of these is what we might call the hardware constraint. While hardware details are examined in greater detail in Section 2.1, a few remarks are in order now.

Consider first typical calculators. These machines are designed to perform a very limited collection of operations. Algorithms to perform arithmetic and to compute the elementary functions must be closely matched with the electronic components. In a calculator, the transcendental functions are computed using the little-known method called *decimal CORDIC*. “CORDIC” stands for COordinate Rotation DIgital Computer. It is an ingenious method that uses formulas for functions of a sum as a basis for a sequence of recursive computations that converge to the desired function value. The CORDIC procedure has the advantage that a very limited instruction set will permit the evaluation of all of the elementary functions. We will discuss CORDIC at length later in Section 5. To the best of our knowledge, all present-day calculators use a CORDIC method to compute values of the elementary functions.

Computers, on the other hand, being general purpose machines, allow more flexibility in the choice of algorithm. Aside from the CORDIC method, which is used in numeric coprocessors of personal computers, three general classes of procedures are available to compute the elementary functions. These procedures are

- analytic function approximation, such as Taylor series;
- iterative techniques, such as the Newton-Raphson method; or

- table lookup coupled with a means of interpolation.

We close this introduction with a mention of Taylor and Maclaurin series. These series are not universally used in computers, because often there are better approximations. But Taylor series do illustrate some general considerations for all function approximation. Among these are

- computability with only elementary arithmetic operations;
- error estimates, to provide information about accuracy; and
- range reduction.

The reader is encouraged to work the exercises as they are encountered in the text. Some exercises ask that errors be estimated or routines be compared. This can be done most easily by using a function-graphing program (with double precision, if possible) on a computer. When doing this, assume that the computer is able to produce exact values for the elementary functions.

Exercises

1. Determine how many terms of the Maclaurin series for e^x are needed to estimate e^x to six-digit accuracy
 - a) on the interval $[-1, 1]$;
 - b) near $x = 10$.
2. Investigate the absolute error that results in using the first five terms of the Maclaurin series to approximate e^x on the interval $[-1, 1]$. This can be done by graphing

$$y = e^x - \left(1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}\right).$$

Where is the error maximal?

2. Background Information

To understand what goes into the design of an algorithm, the reader must be aware of hardware aspects, general function evaluation requirements, and error considerations. A discussion of these follows.

2.1 Hardware Aspects

We need at least a rudimentary understanding of a few hardware concepts, including:

- the base used for number representation,

- integer vs. real-number representation and arithmetic, and
- hardware vs. software implementation of arithmetic operations.

2.1.1 The base used for number representation

Physically, numbers are represented through the use of binary (two-state) elements. Whether these are considered on-off, or up-down, etc., we describe them in terms of the two digits 0 and 1. Thus the most natural base to use for integer storage is two. We can use four bits to represent the integers from 0000 to 1111 binary, equivalent to 0 to 15 decimal. Longer sequences of binary digits will represent larger integers. One byte (eight bits) can represent unsigned integers from 0 to 255, or signed integers from -128 to 127. Frequently, we group the binary digits four at a time and describe them as hexadecimal digits. For example,

0000 binary = 00 decimal = 0 hex,
 1010 binary = 10 decimal = A hex,
 1111 binary = 15 decimal = F hex.

Thus one byte is usually treated as two hex digits. This is an almost universal method adopted for general-purpose digital computers.

Another representation scheme, BCD (*Binary Coded Decimal*), retains base ten. This system is employed in most calculators, which use only the binary configurations 0000 through 1001 to represent the decimal digits 0 through 9. By way of comparison, note that the integer twelve has the bit representation 0001 0010 in BCD, but the representation 0000 1100 in binary (i.e., 0C hex.)

Integers are represented exactly in any base; but the same is not necessarily true for other numbers, because a machine can store only finitely many digits. Whether or not these numbers have exact representations depends on the base and on the number of digits retained. In base ten, the number one-tenth has the exact representation $0.1000 \dots$. Whether our device rounds or truncates, we lose only zeros. By contrast, the number two-thirds in decimal is $0.6666 \dots$, repeating infinitely. Whether we round or truncate, the representation is not exactly two-thirds.

In binary, one-half has the representation $0.1000 \dots$, and one-quarter has the representation $0.0100 \dots$ exactly. But in binary, one-tenth is an infinite repeating nonzero sequence of binary digits following the binary point. No matter where the approximation is terminated, one-tenth cannot be represented exactly in any binary (or essentially binary, such as hexadecimal) number system. This fact is not always obvious from a computer printout. If a machine reads 0.1 and prints it (with a default format), the printout usually shows 0.1. But if on a VAX we specify printing of ten decimal places, the printed value is 0.1000000015.

Another important consequence of the base occurs in multiplication and division. In base ten, multiplications and divisions by a power of ten can be

accomplished by simple shifting of the decimal point. Multiplications by 2, 4, 8, 16, etc., are more cumbersome. In binary, on the other hand, multiplications and divisions by powers of 2 can be accomplished by shifting the binary point, whereas multiplications by 10 are more cumbersome.

2.1.2 Integer vs. real-number representation and arithmetic

Two very common computer number representations are *fixed-point* (integer) and *floating point* (real). Fixed-point corresponds to the familiar representation (without a decimal point) of integer values. Floating-point closely resembles scientific notation, involving two components, the *mantissa* (significant digits) and the *characteristic* (exponent on the floating point base.) Each representation has its own set of arithmetic algorithms. Fixed-point arithmetic is comparatively simple and quick, while floating-point arithmetic is not, because operations must be performed on both mantissa and characteristic.

Exercise

3. Add 0.23×10^5 and 0.53×10^{-4} while retaining scientific notation.

2.1.3 Hardware vs. software implementation of operations

Basic arithmetic operations, such as addition and multiplication, are not themselves among the most fundamental hardware operations. Among the most primitive operations, from which more advanced ones can be built, are bit-by-bit AND, OR, and NOT. With these as building blocks, shifts, compares (as in IF statements), etc., are themselves primitive, hence fast. More advanced still, built up from primitive operations, are bit-by-bit adders with carry, multibit adders, etc., and sometimes multiplication, division, and even square root.

The designers of a calculator or computer decide on the level of complexity that they want to build into the hardware as machine instructions. The instruction set might be quite limited, as for most calculators, or very extensive. The VAX, in fact, has a single machine instruction to evaluate an entire polynomial. The more operations that are built into the hardware, the more complicated and hence more expensive the processor. The operations necessary for problem solution then must be either built into the hardware or programmed as a sequence of built-in instructions (software or firmware.)

In summary, the suitability of a mathematical algorithm for a particular computing device depends in large part on the operations that are built in as machine instructions and therefore are fast. Operations that must be accomplished with programming may be relatively—and sometimes prohibitively—slow.

2.2 Requirements of Function Evaluation Routines

Let's introduce some notation. Denote by f the function to be evaluated and by F the routine that is used to approximate f computationally. Let x denote an element of the domain of f . The list below, adapted from Fike [1968], gives requirements for any algorithm acceptable for general use.

- **Accuracy.** Ideally, for each x , the value $f(x)$ should equal $F(x)$ to the precision of the computations.
- **Speed.** The time required to compute the value of a function is dominated by the number of multiplications (and divisions) performed. The fastest algorithms use only addition, shifting, and comparisons.
- **Special arguments.** We expect the value of a function for special arguments to be exact to the limit of machine precision. For example, if $x = 0$, then $\cos x$ should be reported as exactly 1.
- **Invalid arguments.** There are two aspects to consider. First, a function itself may not be defined for every real number, as, for example, \sqrt{x} is not defined for $x < 0$. The algorithm must test for such values. Second, every machine imposes limits on the magnitudes it can represent. This reduces the domain over which a function is defined computationally.
- **Bounds on F .** When f is bounded, then F must be as well. An obvious example is $\sin x$ or $\cos x$. No approximation of these functions should ever exceed 1 for any value of x .
- **Monotonicity.** If f is an increasing or decreasing function, then so must F be increasing or decreasing.

2.3 Error Considerations

Of paramount importance is the maintenance of an acceptable level of accuracy over the entire domain of the function. In the case of a Maclaurin series, a few terms are usually sufficient to achieve great accuracy if x is near 0. However, as $|x|$ increases, accuracy can only be achieved by using an ever-increasing number of terms of the series. In theory, we could overcome this problem by using different Taylor series representations, each restricted to a particular subinterval of the domain. But this is not a realistic solution, because most domains are quite extensive.

A far superior method, *reduction of range*, is used in practice. The general idea is to find an algorithm for the function which is very accurate and efficient on a comparatively small interval, the *fundamental interval*. For the method to be applicable to values outside of this interval, a transformation is needed to express $f(x)$ for arbitrary x in terms of $f(x)$ in the primary interval. This transformation and its inverse transformation must be easy to compute.

Each function algorithm has its own specific range-reduction procedure. One simple example to illustrate the process is $f(x) = \log_{10} x$. Write x as $z \times 10^n$, where $1 \leq z < 10$ and n is an integer. Then

$$\log_{10} x = \log_{10}(z \times 10^n) = \log_{10} z + \log_{10} 10^n = \log_{10} z + n.$$

Now all that is needed is a very accurate formula for $\log_{10} z$ for $1 \leq z < 10$.

The problem of error really reduces to just one question: How many significant digits will the procedure produce? While it is not the purpose of this Module to discuss error analysis in depth, we need to discuss sources of error.

Suppose we wish to evaluate the function f at the value x , where x itself is exact. (An error in x itself would complicate the error analysis even further.) We approximate f by the formula or algorithm F . The formula F usually cannot be used directly but rather is implemented on a machine. Let's call the implemented formula F^* . Thus, we see that any procedure will involve the three possibly distinct values $f(x)$, $F(x)$, and $F^*(x)$. There are three different error quantities of interest:

- the *truncation error* $\epsilon_1(x) = |f(x) - F(x)|$. The terminology arises from the Taylor series representations of the elementary functions; we may always take F to be the truncated series.
- the *rounding error* $\epsilon_2(x) = |F(x) - F^*(x)|$. Rounding error is the result of the precision with which the computations are performed, the order of operations, whether intermediate values are rounded or truncated, and peculiarities of machine design.
- the *absolute error* $\epsilon(x) = |f(x) - F^*(x)|$. We have $\epsilon(x) \leq \epsilon_1(x) + \epsilon_2(x)$.

Generally, it is possible to obtain a good estimate of the truncation error. As a result, we are able to design formulas that are highly accurate. More difficult to estimate (and to control) is the rounding error and especially how it propagates throughout a formula.

Another source of error (which can arise from the conversion of decimal to binary) is *conditioning* or *sensitivity*. Conditioning measures the effect on the function of an inaccuracy in the argument. Sometimes a small error in an independent variable produces a small error in the dependent variable. However, it may also happen that a small error in the independent variable can produce an enormous error in the dependent variable, over and above any computational errors. Such is the case for $\ln x$ near 1. A small inaccuracy in x , due, for example, to conversion of exact decimal to approximate binary, can produce large relative error in the corresponding logarithm. A VAX 11/785 computes $\ln 1.001$ as 9.99547E-4. The last two digits are in error. However, $\ln(1025/1024)$ is computed correctly as 9.76086E-4. Why the difference? The number 1.001 does not have a terminating binary representation, but 1025/1024 does.

3. The Square Root

There are several reasons why we discuss the computation of the square root in detail. The square root is used within other algorithms, such as those involving trigonometric identities. The calculator version (Section 3.1) of the computation shows some of the features that distinguish the CORDIC from analytic procedures. The computer version (Section 3.2) is the only example of an iterative analytic procedure in this Module. Both procedures are short and simple.

3.1 Calculator Computation of Square Root

In order to compute \sqrt{x} , let us first write x in scientific notation so that $x = z \times 10^k$. We may always choose k even and $1 \leq z < 100$. Since $\sqrt{x} = \sqrt{z} \times 10^{k/2}$, we need only compute \sqrt{z} .

The reader must keep in mind that the algorithm must be one that is easily implemented in hardware. The key to the computation is the fact that the sum of the first n odd numbers is precisely n^2 : for example, $1 + 3 + 5 + 7 + 9 = 5^2$.

The procedure that we describe is essentially that used by Hewlett-Packard in its calculators (see [Egbert 1977a]). By design, the square root is computed from below, digit by digit. Let a_1 be the one-digit approximation to \sqrt{z} ; and, in general, let a_j be the j -digit approximation. Given a_j , we wish to compute the next digit b in the approximation. Notice that a_1 is the units digit (since $1 \leq a_1 < 10$). Therefore, $a_{j+1} = a_j + b \times 10^{-j}$, for $j = 1, 2, 3, \dots$.

Let $R_j = z - a_j^2$; R_j is a measure of the “goodness” of a_j as \sqrt{z} . By approaching z from below, we have $R_j \geq 0$ for all j , with

$$\begin{aligned} R_{j+1} &= z - a_{j+1}^2 = z - (a_j + b \times 10^{-j})^2 \\ &= z - (a_j^2 + 2a_jb \times 10^{-j} + b^2 \times 10^{-2j}) \\ &= z - a_j^2 - (2a_jb \times 10^{-j} + b^2 \times 10^{-2j}) \\ &= R_j - (2a_jb \times 10^{-j} + b^2 \times 10^{-2j}). \end{aligned}$$

The strategy is to choose b so as to make R_{j+1} a minimum. The form above is not optimal for implementation on a calculator, because the term containing a_j has other factors. To improve it, first multiply through by 5 and then eliminate the factor of a power of 10 from the term a_jb . This yields

$$10^{j-1} \times 5R_{j+1} = 10^{j-1} \times 5R_j - (a_jb + 5b^2 \times 10^{-j-1}).$$

Using the fact that $b^2 = \sum_{i=1}^b (2i-1)$ and writing a_jb as $\sum_{i=1}^b a_j$, we obtain

$$10^{j-1} \times 5R_{j+1} = 10^{j-1} \times 5R_j - \sum_{i=1}^b (a_j + 5(2i-1) \times 10^{-j-1}).$$

Since $5(2i-1) = 05, 15, 25, \dots, 95$ as $i = 1, 2, 3, \dots, 10$, we may represent $5(2i-1)$ as $(i-1)|5$ and write

$$10^{j-1} \times 5R_{j+1} = 10^{j-1} \times 5R_j - \sum_{i=1}^b (a_j + (i-1)|5 \times 10^{-j-1}). \quad (1)$$

We may initialize the process with $a_0 = 0$ and $R_0 = z$ in (1) (see **Exercise 4**). Indeed, this is what is done in the calculator. By hand, it is easier for us to find a_1 and R_1 directly. After this point, (1) can be used repeatedly. We must emphasize that it is not necessary to know R_j itself; the calculation requires only that we know $10^{j-1} \times 5R_j$.

Example. Compute $\sqrt{27.17954}$.

$R_1 = 27.17954 - (1 + 3 + 5 + 7 + 9) = 2.17954$. Clearly, $a_1 = 5$.

To find a_2 , we put $j = 1$ in (1). This gives

$$\begin{aligned} 5R_2 &= 5R_1 - \sum_{i=1}^b (a_1 + (i-1)|5 \times 10^{-2}) \\ &= 10.8977 - \sum_{i=1}^b (5 + (i-1)|5 \times 10^{-2}) \\ &= 10.8977 - (5.05 + 5.15) \\ &= 0.6977. \end{aligned}$$

Therefore, $b = 2$ (since we subtracted two terms) and $a_2 = 5.2$. This is the only time we use multiplication by 5.

To find a_3 , put $j = 2$ in (1). This gives

$$\begin{aligned} 10 \times 5R_3 &= 10 \times 5R_2 - \sum_{i=1}^b (a_2 + (i-1)|5 \times 10^{-3}) \\ &= 6.977 - \sum_{i=1}^b (5.2 + (i-1)|5 \times 10^{-3}) \\ &= 6.977 - (5.205) \\ &= 1.772. \end{aligned}$$

Therefore, $b = 1$ (one subtraction) and $a_3 = 5.21$.

To find a_4 , put $j = 3$ in (1). This gives

$$\begin{aligned} 10^2 \times 5R_4 &= 10^2 \times 5R_3 - \sum_{i=1}^b (a_3 + (i-1)|5 \times 10^{-4}) \\ &= 17.72 - \sum_{i=1}^b (5.21 + (i-1)|5 \times 10^{-4}) \\ &= 17.72 - (5.2105 + 5.2115 + 5.2125) \\ &= 2.0855. \end{aligned}$$

Therefore, $b = 3$ and $a_4 = 5.213$.

To find a_5 , put $j = 4$ in (1). This gives

$$\begin{aligned}
 10^3 \times 5R_5 &= 10^3 \times 5R_4 - \sum_{i=1}^b (a_4 + (i-1)|5 \times 10^{-5}) \\
 &= 20.855 - \sum_{i=1}^b (5.213 + (i-1)|5 \times 10^{-5}) \\
 &= 20.855 - (5.21305 + 5.21315 + 5.21325 + 5.21335) \\
 &= 0.0022.
 \end{aligned}$$

Therefore, $b = 4$ and $a_5 = 5.2134$. We will stop here.

Note the simplicity of the operations involved. With the exception of the multiplication by 5 in step 1, which could be accomplished with additions, the only operations are subtractions, additions, shifts, and comparisons.

Exercise

4. Using the procedure of this section, compute $\sqrt{27.17954}$, beginning with $a_0 = 0$ and $R_0 = z$.

3.2 Computer Computation of the Square Root

Our goal is to compute \sqrt{x} when x is a positive number. One strategy employed is the iterative scheme known as the Newton-Raphson procedure, often called *Newton's method*. A reader who is unfamiliar with this method should refer to any calculus textbook; a reader who is familiar with the method should verify that (2) below is the correct instance for \sqrt{x} of the general procedure.

Assume that the computer is a binary machine. A binary-point representation of x is $x = z \times 2^k$, where $\frac{1}{2} \leq z < 1$ and k is an integer. For the moment, assume that k is even. Then $\sqrt{x} = \sqrt{z} \times 2^{k/2}$. We need only compute \sqrt{z} and accomplish the multiplication by $2^{k/2}$ via shifting. The procedure we describe is that used in the VAX system library (see [VAX 1988]).

Given the initial approximation a_0 to \sqrt{z} , we form the sequence of estimates a_1, a_2, \dots by the Newton-Raphson recursion

$$a_{i+1} = \frac{1}{2} \left(a_i + \frac{z}{a_i} \right). \quad (2)$$

All that remains is the determination of the initial value a_0 . This is done by approximating \sqrt{x} on the interval $[\frac{1}{2}, 1]$ by a line segment. The segment, chosen to minimize the error at the midpoint and endpoints, is on the line given by the equation $L(x) = 0.5857864x + 0.4204951$. We take $a_0 = L(z)$.

If k is odd, rewrite x as $x = \frac{z}{2} \times 2^{k+1}$. We now must compute $\sqrt{\frac{z}{2}}$, where $\frac{1}{4} \leq \frac{z}{2} < \frac{1}{2}$. Again, we construct a sequence of estimates a_1, a_2, \dots using the Newton-Raphson procedure (2) and starting with $a_0 = 0.8284271 \left(\frac{z}{2}\right) + 0.2973349$. This value of a_0 is obtained from a linear approximation to \sqrt{x} on the interval $\left[\frac{1}{4}, \frac{1}{2}\right]$.

The sequence a_0, a_1, a_2, \dots converges quite rapidly to \sqrt{z} . In fact, the VAX needs only two iterations for single-precision accuracy and three for double-precision accuracy.

Exercise

5. Using the procedure described in this section, compute
 - a) $\sqrt{0.3}$;
 - b) $\sqrt{0.7}$.

4. Elementary Functions on Computers

Most general-purpose computers handle the evaluation of the elementary functions by an explicit function approximation (formula). After some preliminary remarks and examples in Section 4.1, in which various polynomial and rational function approximations to $\exp x$ are briefly discussed, we will exhibit specific methods in Section 4.2 for the calculation of $\exp x$ and $\ln x$.

4.1 Preliminary Remarks

As mentioned earlier, the truncated Taylor series is sometimes a poor choice for the evaluation of a function. It tends to be very accurate only near the point about which it is expanded. There exist polynomials of the same degree, known as *economized polynomials*, which distribute the error more uniformly over an interval.

For $f(x) = e^x$, one economized polynomial for the interval $[-1, 1]$, based upon the Chebyshev polynomials (with coefficients given to 5 decimal places) is

$$C(x) = 1.00005 + 0.99730x + 0.49916x^2 + 0.17736x^3 + 0.04384x^4.$$

We compare this with the truncated Maclaurin series of degree 4,

$$M(x) = 1.00000 + 1.00000x + 0.50000x^2 + 0.16667x^3 + 0.04167x^4.$$

To see the advantage of the economized polynomial, examine the graphs of $E_C(x) = |e^x - C(x)|$ and $E_M(x) = |e^x - M(x)|$ in **Figure 1**.

An improvement over the Taylor polynomial of degree n can sometimes be achieved by approximating the function by a rational function $R(x) = p(x)/q(x)$, where the degree of p and the degree of q sum to n . One frequently

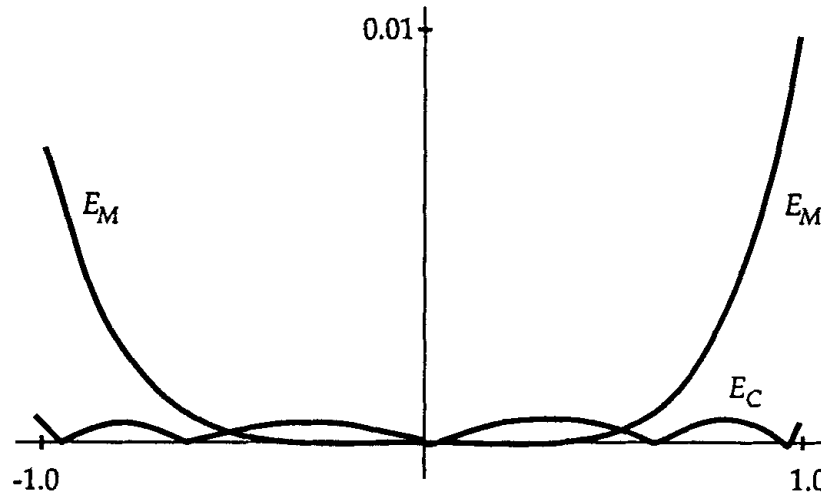


Figure 1. E_C compared to E_M .

employed rational approximation is the *Padé approximation*, which has an error term different from that of the Taylor polynomial. Again using the exponential function as an example, the Padé approximation $R(x)$ obtained by expanding about the origin is

$$R(x) = \frac{1 + x/2 + x^2/12}{1 - x/2 + x^2/12}.$$

To see the improvement over the corresponding fourth-degree Taylor polynomial, refer to **Figure 2**, which compares the graphs of E_M and $E_R(x) = |e^x - R(x)|$.

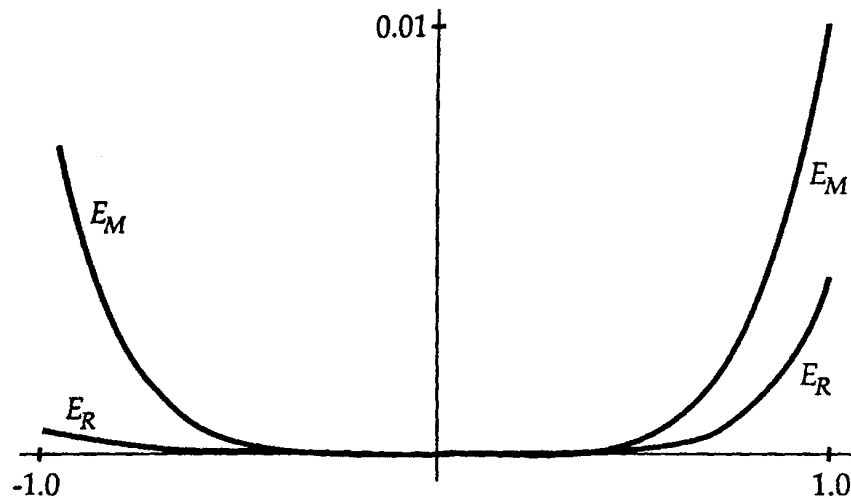


Figure 2. E_R compared to E_M .

For a treatment of Chebyshev polynomials and Padé approximations, see

Gerald and Wheatley [1989]. Although in general both the economized polynomial and the Padé approximation are often improvements over the Taylor polynomial, still further improvements can be made. Since some aspects of function approximation remain an art, many different methods are used in practice.

4.2 Computation of Elementary Functions

The first step in the approximation of a function f is to exploit one or more of its properties to achieve a reduction in range. Under such a transformation, every element x in the domain of the function is mapped to a corresponding element z in a small interval, the fundamental interval. The second step is to construct on the fundamental interval a polynomial or rational function $p(z)$, the fundamental approximation formula, which approximates the function f with a high degree of accuracy. To obtain the desired value of f , it is necessary that there be a simple relationship between $f(x)$ and $f(z)$, so that the reduction of range transformation can be easily reversed.

4.2.1 Computation of the exponential function

Since $e^x = 2^{x \log_2 e}$, it suffices to develop a procedure for the computation of the latter—a task well-suited to a binary computer.

Let's examine in detail how the VAX computes e^x in ordinary single-precision floating-point (see [VAX 1988]). First, tests are made for three exceptional cases:

- If $x > 88$, the machine signals overflow.
- If $x \leq -89$, the machine returns $e^x = 0$.
- If $|x| < 2^{-25}$, the machine returns $e^x = 1$.

Otherwise, put $y = x \log_2 e$. Consider y in hexadecimal notation with three components:

- u = the integer part,
- v = the hex digit in the first position right of the hexadecimal point, and
- z = the remaining fractional part.

For example, if $y = 41.B2C$, then $u = 41$, $v = B$, and $z = .02C$. Since $y = u + v/16 + z$, it follows that $2^y = 2^u \times 2^{v/16} \times 2^z$. The crucial facts to observe are

- multiplication by 2^u is achieved by a shift of the binary point or by adjusting the characteristic of the floating point representation;
- the value of $2^{v/16}$ can be found by an easy table lookup (there are only 31 possible values); and

- $\frac{-1}{16} < z < \frac{1}{16}$, thereby making a polynomial approximation to 2^z quite feasible. In fact, the Maclaurin polynomial of degree 4 has a maximum error on this interval of order 10^{-9} .

This procedure requires that $\log_2 e (= 1/\ln 2)$ and $\ln 2$ be known with full precision. Historical methods are available for computing isolated values of particular constants such as these. For example, the series

$$\frac{3}{4} - \sum_{n=1}^{\infty} \frac{1}{2n(4n^2 - 1)^2}$$

converges quite rapidly to $\ln 2$. A standard reference [Abramowitz and Stegun 1965] lists values up to 25 significant digits. These can be stored in memory and made available to any routine that requires them.

Exercise

6. Find the Maclaurin polynomial $p(x)$ of degree 4 for 2^x . Verify that

$$|2^x - p(x)| < 3 \times 10^{-9}$$

on $[-\frac{1}{16}, \frac{1}{16}]$.

4.2.2 Evaluation of the natural logarithm

For any $x > 0$, x may be written as $2^n \times z$, where $\frac{1}{2} \leq z < 1$. It follows that $\ln x = n \ln 2 + \ln z$. One should note that the Maclaurin series

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots, \quad |x| < 1, \quad (3)$$

converges quite rapidly in a neighborhood of 0. An efficient algorithm can be based upon this series.

To understand how the computations are carried out, we first make some preliminary observations. The details are relegated to **Exercise 7**.

It is easy to show that

$$\ln \left(\frac{1+x}{1-x} \right) = 2 \left(x + \frac{x^3}{3} + \frac{x^5}{5} + \cdots \right), \quad \text{for } |x| < 1. \quad (4)$$

Consequently, if $X > 0$, there are choices of $A > 0$ so that $V = (X - A)/(X + A)$ satisfies $|V| < 1$. By direct substitution into (4), we obtain

$$\ln \left(\frac{X}{A} \right) = 2 \left(V + \frac{V^3}{3} + \frac{V^5}{5} + \cdots \right), \quad \text{for } |V| < 1,$$

or equivalently,

$$\ln X = \ln A + 2V \left(1 + \frac{V^2}{3} + \frac{(V^2)^2}{5} + \cdots \right). \quad (5)$$

In practice, we choose A in such a way that $|V| \ll 1$. The values of A and $\ln A$ can be stored in a table. When x is near 1, a very short table can contain enough values to assure that V is close to 0. With these ideas in mind, the computation of $\ln x$ is quite straightforward.

We now describe the strategy employed by the VAX routines (see [VAX 1988]). Let us first define the variables N and Z . Write $x = Z \times 2^N$, where $1 \leq Z < 2$ if $x \geq 1$ and $\frac{1}{2} \leq Z < 1$ if $x < 1$. If $|Z - 1| < 2^{-5}$, then let $W = Z - 1$ and use the polynomial of degree 4 obtained from equation (3) to compute $\ln Z = \ln(1 + W)$. Otherwise, use a table lookup to find the value of A with which to set $V = (Z - A)/(Z + A)$. The polynomial in V of degree 5 derived from (5) is evaluated.

Exercises

7. This exercise develops the approximation of $\ln x$.

a) Prove that

$$\ln \left(\frac{1+x}{1-x} \right) = 2 \left(x + \frac{x^3}{3} + \frac{x^5}{5} + \cdots \right), \quad \text{for } |x| < 1.$$

b) For $V = (X - A)/(X + A)$, derive (5). (Hint: Replace x by V in (4).)

8. Compute

a) $\ln 1.001$

b) $\ln 0.78$ using $A = 0.75$ and $\ln A = -0.28768$.

5. Elementary Functions on Calculators

Computations of elementary functions on an electronic calculator are usually performed by some variant of a decimal CORDIC algorithm. This technique was first employed to compute the arctangent and simultaneously the sine and the cosine (see [Volder 1959]). The only machine operations required are shifts, adds, subtracts, compares, and references to stored constants. Thus, CORDIC is ideally suited for implementation on an electronic calculator. We will examine the CORDIC algorithms for sine, cosine, and arctangent only; and we assume that all angles are rendered in radians, not in degrees. Similar algorithms have been developed for the exponential function, the natural logarithm, and, in fact, all the elementary functions.

We will not repeat here the details of range reduction given earlier in the module but assume such a reduction as needed by individual algorithms.

5.1 Sines, Cosines, and Tangents via CORDIC

We begin by investigating a CORDIC algorithm for $\tan x$, where we assume that $x > 0$. A first step is to write x as a finite sum of very special angles, usually the angles whose tangents are the T 's in **Table 1**. A larger table would be used in practice, with more decimal places.

Table 1.
Arctangents of powers of 10.

T	1	0.1	0.01
$\arctan T$	0.785398	0.099669	0.010000

The decomposition of x is obtained by successively subtracting multiples of $\arctan 1$, $\arctan 0.1$, $\arctan 0.01$, etc., until the remaining angle a_0 is very small, and while the reduced angles remain nonnegative. With our choice of T 's, we have $a_0 < 0.01$. A record must be maintained of the T 's used in the decomposition, and of a_0 . The examples below illustrate that the decomposition of x is accomplished computationally by simple table lookups followed by subtraction.

Example. The decompositions of $x = 0.99474$, $x = 3$, and $x = 0.5$ are:

$$\begin{aligned} 0.99474 &= 1 \arctan 1 + 2 \arctan 0.1 + 1 \arctan 0.01, \\ 3 &= 3 \arctan 1 + 6 \arctan 0.1 + 4 \arctan 0.01 + 0.005792, \\ 0.5 &= 0 \arctan 1 + 5 \arctan 0.1 + 0 \arctan 0.01 + 0.001655. \end{aligned}$$

We outline the CORDIC procedure in **Algorithm 1** and illustrate it with an easy example.

Algorithm 1.
The CORDIC algorithm for $\tan x$.

$\tan x = P_n/Q_n$		
where	$x = a_0 + \sum_{i=1}^n \arctan T_i$	$T_i \in \{1, 0.1, 0.01, \dots\}$
	$P_0 = a_0$	$Q_0 = 1$
	$P_i = P_{i-1} + T_i Q_{i-1}$	$Q_i = Q_{i-1} - T_i P_{i-1}$

Example. Compute $\tan 0.99474$.

From the example above, we have

$$0.99474 = 1 \arctan 1 + 2 \arctan 0.1 + 1 \arctan 0.01.$$

Table 2.
The CORDIC algorithm for $\tan 0.99474$.

	$P_0 = 0.00000$	$Q_0 = +1.00000$
$T_1 = 1$	$+T_1 Q_0 = 1.00000$ $P_1 = 1.00000$	$-T_1 P_0 = -0.00000$ $Q_1 = +1.00000$
$T_2 = 0.1$	$+T_2 Q_1 = 0.10000$ $P_2 = 1.10000$	$-T_2 P_1 = -0.10000$ $Q_2 = +0.90000$
$T_3 = 0.1$	$+T_3 Q_2 = 0.09000$ $P_3 = 1.19000$	$-T_3 P_2 = -0.11000$ $Q_3 = +0.79000$
$T_4 = 0.01$	$+T_4 Q_3 = 0.00790$ $P_4 = 1.19790$	$-T_4 P_3 = -0.01190$ $Q_4 = +0.77810$

Hence we may take $T_1 = 1$, $T_2 = T_3 = 0.1$, $T_4 = 0.01$, and $a_0 = 0$. Using the recurrence relations, we have the results in **Table 2**.

Notice that the multiplication of P or Q times T is simple shifting. We find that $\tan 0.99474 = P_4/Q_4 = 1.1979/0.7781 = 1.53952$.

The details of the derivation of this algorithm follow in Section 5.1.1. The reader may, however, proceed directly to Section 5.1.2 to see how CORDIC is used to compute sines and cosines.

5.1.1 The CORDIC algorithm for tangent

The idea behind the CORDIC method for computing $\tan x$ is that $\sin x$ and $\cos x$ each have a “workable” formula for the sum of two angles, namely,

$$\begin{aligned}\sin(a + q) &= \cos q \sin a + \sin q \cos a, \\ \cos(a + q) &= \cos q \cos a - \sin q \sin a.\end{aligned}\tag{6}$$

Hence, if we can write $x = a_0 + q_1 + q_2 + \cdots + q_n$ for some set of angles a_0 and q_i , we can apply (6) repeatedly and eventually arrive at a formal representation for $\sin x$ and $\cos x$. Define

$$a_i = a_0 + q_1 + q_2 + \cdots + q_i = a_{i-1} + q_i.$$

Note that $x = a_n$. Assuming that $\sin q_i$ and $\cos q_i$ are known for each i , we have the procedure of **Table 3**.

Whether the sequences in **Table 3** have any practical computational value depends on whether the repeated applications of the trigonometric identities in (6) can be done efficiently. That is, the computation of the intermediate sines and cosines must be *much* simpler than the direct computation of the sine and

Table 3.

Details behind the CORDIC algorithm for $\tan x$. The ellipses represent the computational application of the trigonometric identities (6).

i	$\sin a_i$	$\cos a_i$
0	$\sin a_0$	$\cos a_0$
1	$\sin(a_0 + q_1) = \dots$	$\cos(a_0 + q_1) = \dots$
2	$\sin(a_1 + q_2) = \dots$	$\cos(a_1 + q_2) = \dots$
\vdots	\vdots	\vdots
n	$\sin x = \sin(a_{n-1} + q_n) = \dots$	$\cos x = \cos(a_{n-1} + q_n) = \dots$

cosine. The fact that this can be done, as we will now see in detail, is what makes the CORDIC method practical.

Consider the general formulas from **Table 3**:

$$\begin{aligned}\sin(a_{i-1} + q_i) &= \cos q_i \sin a_{i-1} + \sin q_i \cos a_{i-1}, \\ \cos(a_{i-1} + q_i) &= \cos q_i \cos a_{i-1} - \sin q_i \sin a_{i-1}.\end{aligned}\tag{7}$$

Assume that $\sin a_{i-1}$ and $\cos a_{i-1}$ are known from the previous step in the iteration. Then the simplicity of the recurrence relations depends predominantly on the simplicity of $\sin q_i$ and $\cos q_i$, both in themselves and as multiplication factors in the current iteration. The heart of the method is to choose the q_i to obtain this simplicity.

When computations are done in decimal, multiplication by an integral power of 10 is just a shift of the decimal point in the other factor. An attempt to find an angle q such that both $\sin q$ and $\cos q$ are integral powers of 10 is fruitless. But finding an angle q for which the ratio of $\sin q$ to $\cos q$ is an integral power of 10 is straightforward.

Let $T_i = 10^{p_i}$ represent an integral (usually nonpositive) power of 10. Define

$$q_i = \arctan T_i, \quad 0 < q_i < \pi/2.$$

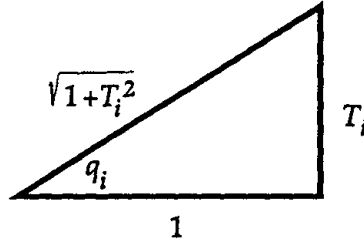
From the fundamental identities illustrated by **Figure 3**, we note that if $w_i = \sqrt{1 + T_i^2}$, then

$$\sin q_i = \frac{T_i}{w_i}, \quad \cos q_i = \frac{1}{w_i}.\tag{8}$$

Table 4 illustrates some possible choices of T_i . Notice that angles with different subscripts (2 and 3) need not be different in value.

Consider again the recurrence relations of (7), but now in the particular case that $q_i = \arctan T_i$ and angle a_0 is so small that $\sin a_0 \approx a_0$ and $\cos a_0 \approx 1$. Let

$$\begin{aligned}S_i &= \sin a_i = \sin(a_0 + q_1 + \dots + q_i), \\ C_i &= \cos a_i = \cos(a_0 + q_1 + \dots + q_i).\end{aligned}$$

**Figure 3.** Triangle for fundamental identities.**Table 4.**
Possible choices for T_i .

i	T	$q_i = \arctan T_i$	q_i	w_i	$\cos q_i$	$\sin q_i$
1	10^0	$\arctan 1$	0.78540	1.41421	$1/w_1$	$1/w_1$
2	10^{-1}	$\arctan 0.1$	0.09967	1.00499	$1/w_2$	$0.1/w_2$
3	10^{-1}	$\arctan 0.1$	0.09967	1.00499	$1/w_3$	$0.1/w_3$
4	10^{-2}	$\arctan 0.01$	0.01	1.00005	$1/w_4$	$0.01/w_4$

Table 5.
Steps in the CORDIC technique for $\tan x$.

$i = 0$	$S_0 = \sin a_0 = a_0$	$C_0 = \cos a_0 = 1$
$i = 1$	$S_1 = \sin(a_0 + q_1)$ $= \cos q_1 \sin a_0 + \sin q_1 \cos a_0$ $= \frac{1}{w_1} \sin a_0 + \frac{T_1}{w_1} \cos a_0$ $= \frac{1}{w_1} (S_0 + T_1 C_0)$	$C_1 = \cos(a_0 + q_1)$ $= \cos q_1 \cos a_0 - \sin q_1 \sin a_0$ $= \frac{1}{w_1} \cos a_0 - \frac{T_1}{w_1} \sin a_0$ $= \frac{1}{w_1} (C_0 - T_1 S_0)$

Then, using (8) extensively, we have the results in **Table 5**.

The steps shown in detail for $i = 1$ may be paralleled for $i > 1$ to obtain the general recursion relations

$$S_i = \frac{1}{w_i} (S_{i-1} + T_i C_{i-1}), \quad C_i = \frac{1}{w_i} (C_{i-1} - T_i S_{i-1}). \quad (9)$$

These recurrence relations are formally simple but still not sufficiently efficient computationally to be useful. For although the multiplication by T_i can be done by shifting the decimal point, the division by w_i remains prohibitively costly computationally.

A slight modification of (9) will eliminate the problem of the divisions by w_i , but at the expense of changing the meanings of the recursive variables. The new recursion formulas are motivated by rewriting the elements of (9) with the w_i 's as factors on the left, rather than as divisors on the right. That is,

$$\begin{array}{ll}
S_0 &= a_0, & C_0 &= 1 \\
w_1 S_1 &= S_0 + T_1 C_0, & w_1 C_1 &= C_0 - T_1 S_0 \\
&= a_0 + T_1 & &= 1 - T_1 a_0 \\
\vdots & & \vdots &
\end{array}$$

Now define the left sides of the relations above as the new recursive variables and generalize them as follows:

$$\begin{array}{ll}
P_0 = S_0 = a_0, & Q_0 = C_0 = 1 \\
P_1 = w_1 S_1, & Q_1 = w_1 C_1.
\end{array}$$

Letting $W_i = w_1 w_2 \cdots w_i$, for $i = 1, 2, \dots, n$, put

$$P_i = W_i S_i, \quad Q_i = W_i C_i, \quad (10)$$

where the definitions of S_i , C_i , and a_i are still as given before. We now collect the last few remarks into the following:

Theorem. P_i and Q_i as defined in (10) satisfy the recursion relations

$$\begin{aligned}
P_0 &= a_0 \ (\approx \sin a_0), \quad Q_0 = 1 \ (\approx \cos a_0), \\
\text{and for } i > 1, \quad P_i &= P_{i-1} + T_i Q_{i-1}, \quad Q_i = Q_{i-1} - T_i P_{i-1}.
\end{aligned} \quad (11)$$

Proof. Since $S_0 = a_0$, $P_0 = a_0$. Likewise, since $C_0 = 1$, $Q_0 = 1$. Below we do the proof for general P , by induction. The proof for Q is left as **Exercise 9**.

i) Proof for $i = 1$ (recall (8)):

$$\begin{aligned}
P_1 &= w_1 S_1 \\
&= w_1 \sin(a_0 + q_1) \\
&= w_1 (\cos q_1 \sin a_0 + \sin q_1 \cos a_0) \\
&= w_1 \left(\frac{1}{w_1} \sin a_0 + \frac{T_1}{w_1} \cos a_0 \right) \\
&= P_0 + T_1 Q_0.
\end{aligned}$$

ii) Proof for $i + 1$, assuming true for i :

$$\begin{aligned}
P_{i+1} &= W_{i+1} S_{i+1} \\
&= w_{i+1} W_i \sin(a_i + q_{i+1}) \\
&= w_{i+1} W_i (\cos q_{i+1} \sin a_i + \sin q_{i+1} \cos a_i) \\
&= w_{i+1} W_i \left(\frac{1}{w_{i+1}} \sin a_i + \frac{T_{i+1}}{w_{i+1}} \cos a_i \right) \\
&= W_i S_i + W_i T_{i+1} C_i \\
&= P_i + T_{i+1} Q_i. \quad \square
\end{aligned}$$

The application of the theorem is a CORDIC method for computing $\tan x$. Computationally, all that is required is multiplication performed by shifting, addition, and subtraction. Recall that

$$x = a_0 + q_1 + q_2 + \cdots + q_n = a_n,$$

so $P_n = W_n S_n = W_n \sin x$ and $Q_n = W_n C_n = W_n \cos x$. The algorithm is completed by dividing: $\tan x = P_n/Q_n$, independent of W_n .

Exercise

9. Complete the induction proof of the theorem above.

5.1.2 The CORDIC computation of sines and cosines

The keys to CORDIC computation of sines and cosines are the trigonometric identities

$$\sin x = \frac{\tan x}{\sqrt{1 + \tan^2 x}}, \quad \cos x = \frac{1}{\sqrt{1 + \tan^2 x}}.$$

Since $\tan x = P_n/Q_n$, it follows that we may use the fast square-root routine to calculate sines and cosines:

$$\begin{aligned} \sin x &= \frac{\tan x}{\sqrt{1 + \tan^2 x}} = \frac{P_n}{\sqrt{P_n^2 + Q_n^2}}, \\ \cos x &= \frac{1}{\sqrt{1 + \tan^2 x}} = \frac{Q_n}{\sqrt{P_n^2 + Q_n^2}}. \end{aligned} \tag{12}$$

Example. Compute $\sin 0.99474$ and $\cos 0.99474$.

In the previous example, we obtained $\tan 0.99474$ as $P_4/Q_4 = 1.1979/0.7781$. We apply (12) to obtain $\sin 0.99474 = 0.83861$ and $\cos 0.99474 = 0.54473$.

Exercise

10. Compute $\sin 2$ and $\cos 2$ using the CORDIC algorithm. Let $a_0 = 0.00053$.

5.2 A CORDIC Algorithm for Arctangent

With a change in initialization and a clever reinterpretation, the CORDIC algorithm for solving

$$\text{find } t = \tan x, \text{ given } x \text{ (by finding } W \sin x \text{ and } W \cos x)$$

can be used to solve the inverse problem

$$\text{find } x = \arctan t, \text{ given } t.$$

Algorithm 2.

The CORDIC algorithm for $\arctan t$. The T_i 's are chosen so that $P_i \downarrow 0$.

	$\arctan t = a_n$		
where	$a_0 = 0$		
	$a_i = a_{i-1} + \arctan T_i$	$T_i \in \{1, 0.1, 0.01, \dots\}$	
	$P_0 = t$	$Q_0 = 1$	
	$P_i = P_{i-1} - T_i Q_{i-1}$	$Q_i = Q_{i-1} + T_i P_{i-1}$	

We outline the CORDIC procedure for computing arctangent in **Algorithm 2** and illustrate it with an easy example. Then, in section 5.2.1, we derive the algorithm.

Example. Compute $\arctan 0.21349$.

Recall that

$$\begin{aligned}\arctan 1 &= 0.78540, \\ \arctan 0.1 &= 0.09967, \\ \arctan 10^{-j} &\approx 10^{-j} \text{ for } j \geq 2.\end{aligned}$$

We generate a sequence a_i by successively adding to a_{i-1} the largest angle $q_i (= \arctan T_i)$ that keeps P_i nonnegative. (We do not show explicitly the test for choice of q_i .) The process terminates when P_n is sufficiently close to zero. At that point $x = a_n$. Hence $x = \arctan 0.21349 = a_4 = 0.21034$. We show the successive calculations in **Table 6**.

Table 6.
The CORDIC algorithm for $\arctan 0.21349$.

	$P_0 = +0.21350$	$Q_0 = +1.00000$	$a_0 = 0$
$T_1 = 0.1$	$-T_1 Q_0 = -0.10000$ $P_1 = +0.11350$	$+T_1 P_0 = +0.02135$ $Q_1 = +1.02135$	$q_1 = 0.09967$ $a_1 = 0.09967$
$T_2 = 0.1$	$-T_2 Q_1 = -0.10214$ $P_2 = +0.01136$	$+T_2 P_1 = +0.01135$ $Q_2 = +1.03270$	$q_2 = 0.09967$ $a_2 = 0.19934$
$T_3 = 0.01$	$-T_3 Q_2 = -0.01033$ $P_3 = +0.00103$	$+T_3 P_2 = +0.00011$ $Q_3 = +1.03281$	$q_3 = 0.01000$ $a_3 = 0.20934$
$T_4 = 0.001$	$-T_4 Q_3 = 0.00103$ $P_4 = +0.00000$	$+T_4 P_3 = +0.00001$ $Q_4 = +1.03282$	$q_4 = 0.00100$ $a_4 = 0.21034$

5.2.1 Derivation of the CORDIC algorithm for arctangent

Let's first review **Algorithm 1**, the CORDIC algorithm for computing $t = \tan x$, with a view toward solving the inverse problem $x = \arctan t$. In that algorithm, we construct the sequence of angles a_i and the associated sequences P_i and Q_i , which are pairwise proportional to $\sin a_i$ and $\cos a_i$. Hence $\tan x = P_n/Q_n$. A fundamental aspect of the algorithm is the termination of the process. This occurs when $a_n = x$, which can be observed directly.

Any attempt to apply this algorithm to solve the inverse problem $x = \arctan t$ would require a different termination test. Even though the sequence of a 's still converges to x , testing whether $a_n = x$ directly is not possible, since x , being the answer, is not known numerically. We might consider terminating the algorithm when $P_n/Q_n = t$, hence $a_n = x$. The trouble with this is not theoretical but computational; the divisions at each step violate the spirit of CORDIC, namely, to divide only by shifting.

A slight modification of the meaning of P_i and Q_i provides an algorithm for $\arctan t$. We introduce another sequence of angles, $\theta_0, \theta_1, \dots, \theta_n$ (which are never seen in the calculations). The key is to choose the θ 's so that testing whether $a_n = x$ is equivalent to testing whether $\tan \theta_n = 0$. We also introduce P_i and Q_i to be pairwise proportional to $\sin \theta_i$ and $\cos \theta_i$. In fact, the proportionality constants will be such that $Q_i \geq 1$, hence $P_i \leq \sin \theta_i$.

The construction is as follows:

$$\begin{aligned} a_0 + \theta_0 &= 0 + \theta_0 = x \\ (a_0 + q_1) + (\theta_0 - q_1) &= a_1 + \theta_1 = x \\ &\vdots \qquad \qquad \qquad \vdots \qquad \qquad \vdots \\ (a_{n-1} + q_n) + (\theta_{n-1} - q_n) &= a_n + \theta_n = x. \end{aligned}$$

Note that $\theta_0 = x$. This value is not known numerically, but what is known is $\tan \theta_0$, that is, t . At the point where $\theta_n = 0$, a_n will equal x , the required answer. Note further that $\tan \theta_n$ is zero.

In the initial step, we choose $P_0 = t$ and $Q_0 = 1$. An identity yields recurrence formulas for P_i and Q_i . Recall that $P_i/Q_i = \tan \theta_i$. The q_i are still the special angles $\arctan T_i$, chosen to drive the θ 's toward zero. Now

$$\begin{aligned} \tan \theta_{i+1} &= \tan(\theta_i - q_{i+1}) \\ &= \frac{\tan \theta_i - \tan q_{i+1}}{1 + \tan \theta_i \tan q_{i+1}} \\ &= \frac{P_i/Q_i - T_{i+1}}{1 + (P_i/Q_i)T_{i+1}} \\ &= \frac{P_i - Q_i T_{i+1}}{Q_i + P_i T_{i+1}}. \end{aligned} \tag{13}$$

This shows that we should define:

$$P_{i+1} = P_i - Q_i T_{i+1}, \qquad Q_{i+1} = Q_i + P_i T_{i+1}.$$

(This definition assures that $Q_i \geq 1$.) Furthermore, we may dispense with an explicit concern for the auxiliary sequence of θ 's. Since $\lim_{\theta \rightarrow 0} (\tan \theta)/\theta = \lim_{\theta \rightarrow 0} (\sin \theta)/\theta = 1$, then $\tan \theta$, $\sin \theta$, and θ all have essentially the same value near zero. Testing whether θ is sufficiently small is equivalent to testing whether $\sin \theta$ is small. Since $P_n \geq \sin \theta$, then P_n may be tested computationally.

Collecting these ideas yields **Algorithm 2**. In practice, it is not necessary to drive the P 's completely to 0. We may make use of the fact that when x is small, $\tan x \approx x$. Suppose that P_i , Q_i , and a_i have been computed and that P_i is small. At this point, let $n = i + 1$. Since $Q_{n-1} \geq 1$, then $\arctan(P_{n-1}/Q_{n-1}) \approx P_{n-1}/Q_{n-1}$. We may terminate the process by taking the last angle to be $q_n = P_{n-1}/Q_{n-1}$. With this choice, we have $P_n = 0$ and $a_n = a_{n-1} + P_{n-1}/Q_{n-1}$.

Exercise

11. Compute $\arctan 2$ using CORDIC.

6. Solutions to the Exercises

1. The truncated series of degree n for e^x is $\sum_{k=0}^n x^k/k!$. By Taylor's theorem, this has remainder $R_n(c) = e^c x^{n+1}/(n+1)!$, where c depends upon x and lies between 0 and x .
 - a) Since $e^1 < 2.8$, to obtain six-digit accuracy, $|R_n(c)|$ must be bounded by 0.5×10^{-6} . One can verify that $n = 10$.
 - b) Since $e^{10} < 23,000$, to obtain six-digit accuracy, $|R_n(c)|$ must be bounded by 0.05. One can verify that $e^{10}10^{36}/36! \approx 0.06$ and $e^{10}10^{37}/37! \approx 0.017$. Hence $n = 36$.
2. Use a function grapher to see that the maximum occurs at $x = 1$.
3. Write the sum as $0.23 \times 10^5 + 0.00000000053 \times 10^5 = 0.23000000053 \times 10^5$. Note, however, that addition on a machine with eight-digit floating-point precision will produce instead the result 0.23000000×10^5 .
4. $a_0 = 0$, $R_0 = 27.17954$. For $j = 0$, we have

$$\begin{aligned}
 10^{-1} \times 5R_1 &= 10^{-1} \times 5R_0 - \sum_{i=1}^b (0 + (i-1)|5 \times 10^{-1}) \\
 &= \frac{1}{2}(27.17954) - \sum_{i=1}^b ((i-1)|5 \times 10^{-1}) \\
 &= 13.58977 - (0.5 + 1.5 + 2.5 + 3.5 + 4.5) \\
 &= 1.08977.
 \end{aligned}$$

Hence $b = 5$ (that is, five subtractions) and $a_1 = 5$. With $j = 1$, we have

$$\begin{aligned} 5R_2 &= 5R_1 - \sum_{i=1}^b (5 + (i-1)|5 \times 10^{-2}) \\ &= 10.8977 - (5.05 + 5.15) \\ &= 0.6977. \end{aligned}$$

Hence $b = 2$ and $a_2 = 5.2$. The rest continues as in the **Example**.

5. **a)** $a_0 = 0.54586303$, $a_1 = 0.54772572$, $a_2 = 0.5477225575$ (only two iterations).
b) $a_0 = 0.83054558$, $a_1 = 0.8366825337$, $a_2 = 0.8366600268$ (only two iterations).
6. $p(x) = 1 + x \ln 2 + x^2(\ln 2)^2/2 + x^3(\ln 2)^3/6 + x^4(\ln 2)^4/24$, with an upper bound for the error on the interval $[-\frac{1}{16}, \frac{1}{16}]$ given by $(\ln 2)^5 2^{1/16} (1/16)^5/60 = 2.655 \times 10^{-9} < 3 \times 10^{-9}$.
7. **a)** The Maclaurin series for $\ln(1+x)$ is $x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \dots$, valid for $|x| < 1$. The series for $\ln(1-x)$ is $-(x + \frac{1}{2}x^2 + \frac{1}{3}x^3 + \frac{1}{4}x^4 + \dots)$. Therefore, for $|x| < 1$,

$$\ln\left(\frac{1+x}{1-x}\right) = \ln(1+x) - \ln(1-x) = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots\right).$$

$$\textbf{b)} \quad \frac{1+V}{1-V} = \frac{1 + \left(\frac{X-A}{X+A}\right)}{1 - \left(\frac{X-A}{X+A}\right)} = \frac{2X}{2A} = \frac{X}{A}.$$

Therefore, for $|V| < 1$,

$$\ln\left(\frac{1+V}{1-V}\right) = \ln\left(\frac{X}{A}\right) = \ln X - \ln A = 2\left(V + \frac{V^3}{3} + \frac{V^5}{5} + \dots\right).$$

8. **a)** From (3), $\ln 1.001 = 0.001 - (0.001)^2/2 + (0.001)^3/3 = 0.0009995$.
b) Let $V = (0.78 - 0.75)/(0.78 + 0.75) = 0.019608$. From (5),

$$\begin{aligned} \ln 0.78 &= -0.28768 + 2(0.019608) \left(1 + \frac{(0.019608)^2}{3} + \frac{(0.019608)^4}{5}\right) \\ &= -0.248459. \end{aligned}$$

9. i) For $i = 1$:

$$\begin{aligned}
Q_1 &= w_1 C_1 \\
&= w_1 \cos(a_0 + q_1) \\
&= w_1 (\cos a_0 \cos q_1 - \sin a_0 \sin q_1) \\
&= w_1 \left((\cos a_0) \frac{1}{w_1} - (\sin a_0) \frac{T_1}{w_1} \right) \\
&= Q_0 - T_1 P_0.
\end{aligned}$$

ii) Proof for $i + 1$, assuming true for i :

$$\begin{aligned}
Q_{i+1} &= w_{i+1} C_{i+1} \\
&= w_{i+1} W_i \cos(a_i + q_{i+1}) \\
&= w_{i+1} W_i (\cos a_i \cos q_{i+1} - \sin a_i \sin q_{i+1}) \\
&= w_{i+1} W_i \left((\cos a_i) \frac{1}{w_{i+1}} - (\sin a_i) \frac{T_{i+1}}{w_{i+1}} \right) \\
&= Q_i - T_{i+1} P_i.
\end{aligned}$$

10. $2 = 2 \arctan 1 + 4 \arctan 0.1 + 3 \arctan 0.01 + 0.00053$. The intermediate computations are shown in **Table 7**. Using the identities (12), we obtain $\sin 2 = 0.909297$ and $\cos 2 = -0.41615$.

Table 7.
Computations for **Exercise 10**.

i	T_i	P_i	Q_i
0		0.00053	+1.00000
1	1	1.00053	+0.99947
2	1	2.00000	-0.00106
3	0.1	1.99989	-0.20106
4	0.1	1.97978	-0.40105
5	0.1	1.93968	-0.59990
6	0.1	1.87969	-0.79300
7	0.01	1.87176	-0.81180
8	0.01	0.86364	-0.83051
9	0.01	0.85534	-0.84915

11. After the last line of computations in **Table 8**, we claim that P_6 is small. Hence $\arctan 2 = a_7 = a_6 + P_6/Q_6 = 1.10715$.

Table 8.
Computations for **Exercise 11.**

i	T_i	P_i	Q_i		a_i
0		2.00000	1.00000		0
1	1	1.00000	3.00000	$\arctan 1 =$	0.78540
2	0.1	0.70000	3.10000	$a_1 + \arctan 0.1 =$	0.88507
3	0.1	0.39000	3.17000	$a_2 + \arctan 0.1 =$	0.98474
4	0.1	0.07300	3.20900	$a_3 + \arctan 0.1 =$	1.08441
5	0.01	0.04091	3.20973		1.09441
6	0.01	0.00881	3.21014		1.10441

References

There are many aspects to computer function approximation that we could not discuss in a Module of this size. The brief comments that we provide below with the references could serve as a starting point toward further investigation. Mention of the CORDIC algorithm in numerical analysis or numerical methods texts is rare. Hence there are few such entries in this bibliography.

We call the reader's attention to two particular areas of CORDIC. First, there is an enlightening geometric interpretation to the CORDIC sequences. Second, CORDIC algorithms for all of the elementary functions have essentially the same form. The main differences among them are initialization and the particular table of values that are stored. Hence, it is possible to construct a *unified algorithm* for CORDIC computation.

Abramowitz, M., and I.A. Stegun, eds. 1965. *Handbook of Mathematical Functions*. New York: Dover.

A standard reference book for special functions.

Cody, W J., and W. Waite. 1980. *Software Manual for the Elementary Functions*. Englewood Cliffs, NJ: Prentice-Hall.

Provides accuracy tests for the elementary functions.

Duncan, Ray. 1989–1990. Arithmetic routines for your computer programs, Parts 1–7. *PC Magazine* 8(19) (14 November 1989): 423–429; 8(20) (28 November 1989): 345–352; 8(21) (12 December 1989): 337–354; 8(22) (26 December 1989): 271–277; 9(1) (16 January 1990): 311–332; 9(3) (13 February 1990): 297–307; 9(5) (13 March 1990): 353–370.

These columns describe the IEEE standards for floating-point representation, integer and floating-point arithmetic routines, and computation of elementary functions, using math coprocessors.

Egbert, W.E. 1977–1978. Personal calculator algorithms I: Square roots. II. Trigonometric functions. III. Inverse trigonometric functions. IV. Logarithmic functions. *Hewlett-Packard Journal* 28(9) (May 1977): 22–24; 28(10) (June 1977): 17–20; 29(3) (November 1977): 22–23; 29(8) (April 1978): 29–32.

These columns describe in detail the CORDIC algorithms as developed for the HP-35.

Fike, C.T. 1968. *Computer Evaluation of Elementary Functions*. Englewood Cliffs, NJ: Prentice-Hall.

A classic.

Gerald, C.F., and P.O. Wheatley. 1989. *Applied Numerical Analysis*. 4th ed. Reading, MA: Addison-Wesley.

Discussion of Chebyshev and Padé approximations, mention of CORDIC.

Glass, L. Brent. 1990. Math coprocessors. *Byte* 15(1) (January 1990): 340.

Mentions a math coprocessor recently introduced by Cyrix which uses polynomial approximations rather than CORDIC.

Hart, John F. 1978. *Computer Approximations*. Huntington, NY: Krieger.

Source of analytic approximations.

Kropa, J.C. 1978. Calculator algorithms. *Mathematics Magazine* 51(2): 106–109.

Includes geometric interpretation of CORDIC.

Ruckdeschel, F.R. 1981. *Basic Scientific Subroutines*. Vol. 2. New York: Byte/-McGraw-Hill.

Contains a discussion of CORDIC and gives source listings of BASIC programs that imitate CORDIC.

Schelin, C.W. 1983. Calculator function approximation. *American Mathematical Monthly* 90(5): 317–325.

Discussion of unified CORDIC algorithm, with an extensive bibliography.

Schmid, Herman. 1974. *Decimal Computations*. New York: Wiley.

Discussion of CORDIC and other techniques, with emphasis on hardware.

Spafford, E., and J. Flaspohler. 1985. A report on the accuracy of some floating point math functions on selected computers. Technical Report GIT-ICS 85/06. Atlanta, GA: School of Information and Computer Science, Georgia Institute of Technology.

Comparison of accuracy in library routines for a large class of computers.

Swartzlander, E.E., ed. 1980. *Computer Arithmetic*. Stroudsburg, PA: Dowden, Hutchinson & Ross.

A collection of papers, including those of Volder and Walther.

VAX Programming, Vol. 5A: Run-Time Library. 1988. Maynard, MA: Digital Equipment Corporation.

Volder, J.E. 1959. The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computation* EC-8: 330-334 (September 1959).

The paper that started it all (if one doesn't count Briggs).

Walther, J.S. 1971. A unified algorithm for elementary functions. *Spring Joint Computer Conference Proceedings* (1971): 379-385.

A unified CORDIC algorithm with hardware emphasis.

Acknowledgments

The CORDIC section of this module is the result of assistance from many sources. Special thanks to Ron Tabor of Hewlett-Packard and Jeff Crumb and C.B. Wilson of Texas Instruments, who responded to our inquiries and furnished us with information about their machines. The authors also wish to thank the referees and the editor for many helpful suggestions.

About the Authors

When this Module was written, Jim Delaney was a professor in the Dept. of Mathematics and Computer Science at Xavier University, where he had been a member since 1963. Prior to that, he was employed for six years as a programmer-analyst and applied mathematician by the General Electric Co. in Evendale, Ohio. He received a Ph.D. in mathematics from the University of Cincinnati. His interests include computer science education, computer accuracy, and applied mathematics. He is now retired.

Dick Pulskamp is an associate professor of mathematics at Xavier. He received his Ph.D. in Mathematics in 1988 from the University of Cincinnati. While his main area of interest is mathematical statistics, he is also interested in problems of numerical computations.