NUMERICAL EVALUATION OF THE SINE AND COSINE FUNCTIONS

A THESIS

SUBMITTED TO THE FACULTY OF ATLANTA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

BY

ROBIN MARIE HARMON

DEPARTMENT OF MATHEMATICAL SCIENCES

ATLANTA, GEORGIA

DECEMBER 1982

R iv T37

X

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER I

AN INTRODUCTION TO ADA

History and Features

Ada is a high level language developed under the requirements of the United States Department of Defense. The intent of Ada is to have a language that is compatible over a wide range of computing systems. The development of Ada began in 1974. The many contributors to the definition and design of Ada include the military services, industrial organizations, universities, and foreign military departments.[1] The foundation for the language was then compiled into the Steelman Report.[2]

Ada is similar in structure to the Pascal programming language. The language supports the structured programming approach, including control structures, user definable data types and subprograms. Ada's subprograms can, additionally, be packaged, and there is a facility for the separate compilation of these programming units. Other enhancements to the language include real-time programming, parallel task modeling, and exception handling (run-time error handling).

The two types of subprograms in Ada, procedures and functions, are very much like those in Pascal. However, Ada allows any value computed in the function subprogram to be returned to the calling program by means of the RETURN statement.

The package facility is similar to the file facility in Pascal in that it describes a group of logically related entities. A package may include a common pool of data types, a collection of related subprograms, or a set of type declarations and associated operations.[3] Portions of the package can be hidden from the user, thus allowing access only to those logical properties pertinent to the user's application.

A task[4] is the basic programming unit used to describe a sequence of actions that can be executed in parallel with other units. Parallel tasks may be implemented on multicomputers, multiprocessors, or with interleaved execution on a single processor. Task definitions can also be nested.

The data types of Ada are the scalar types, composite types, access types, and private types. Scalar types can either be enumeration or numeric. Computations can be either exact or approximate. Approximate computations use either fixed-point or floating-point types with absolute or relative bound on the error, respectively.[5] The fixed-point type in Ada is written with a fixed number of digits in the mantissa, no exponent, and at least one digit preceding the decimal point.

Composite types allow definition of the structured data types, arrays and records. Access types allow the creation of linked data structures. Private types are used in connection with packages.

Ada also makes it possible for user organization to create their own utility libraries, which is the motivation for the subject of this paper.

Ada/Ed

Ada/Ed was an attempt to implement the design efforts of Ada as soon as possible. Therefore, Ada/Ed was implemented before Ada was completely designed or understood.[6] In other words, Ada/Ed was designed to test the details of Ada as it currently stood.

The Ada/Ed translator and interpreter were developed at New York University for the United States Army. Along with the implementation of Ada/Ed, problems that arose with language definitions had to be solved. Hence Ada/Ed, also the first implementation of Ada, is very close to a complete definition of Ada, with some current limitations. In fact, a primary goal of the implementors was to develop Ada/Ed at such a level that whatever decisions were made could be easily understood and changed if necessary.

The entire Ada/Ed system was programmed using a very high level language developed at NYU, with the help of the National Science Foundation, called SETL (for Set Language).[7] SETL is a design or specification language which can also be compiled and executed. SETL avoids much of the detail of data structuring which makes the logical design more apparent than if conventional languages such as Pascal, BLISS, or SIMULA were used to program Ada/Ed.[8] Ada/Ed is currently implemented in the DEC VAX 11/780 version of SETL.

Even with the advantages of using SETL to run Ada/Ed, the process is very slow. However, inefficiency was sacrified in order to concentrate on the clarity of the implementation. Thus, at this point Ada/Ed is only useful for checking purposes and for becoming familiar with Ada.

CHAPTER II

FUNCTION EVALUATION

Introduction

Approximating functions was first necessary because of insufficient knowledge about certain functions.[9] Today computers are used in approximating functions as a means of saving time and money. With computers, various techniques have been used to compute values of elementary functions to a certain accuracy over a specified interval. The selection of the best technique depends on the function to be evaluated. Approximated values may be the result of the finiteness of any computer and the computer's arithmetic.

There are only two ways to represent a function in a computer:[10] by a table of values or by a subprogram that can compute values of the function at specific points. Because storing a table of values can be wasteful of memory, rational functions (and polynomials) are most often chosen to represent functions to be evaluated on a computer. The coefficients of a polynomial may require little storage space depending on the desired accuracy of the computation. Furthermore, rational functions and polynomial approximations only require the basic arithmetic operations of the computer in order to be evaluated.

Design Requirements

Before the design of a function subroutine can begin there are certain key factors that must be considered. Some of these requirements are as follows:[11]

(1) Accuracy. - In general full-word accuracy is desirable, i.e., the approximation to F(x), say F*(x), should be rounded to the precision of the computation.

(2) Speed; Length. - Both speed and brevity of a function evaluation routine are usually desired. Unfortunately these two objectives conflict to a great extent. Execution time can be reduced if more storage is used. Conversely, a routine can be shortened in exchange for increased execution time.

(3) Special Arguments. - Often a certain argument is required to have the same value for the approximation as the true value of the function to be evaluated. This way errors in the approximation can be easily detected. For example, the design may require that $F^*(0) = 1$ when $F(x) = \cos(x)$.

(4) Invalid Arguments. - A consistent policy is needed concerning error messages and job termination. This is important if an argument is used for which the function is not defined or if overflow or underflow conditions occur during execution.

(5) Bounds on F*(x). - Suppose, for example, that $F(x) = \sin(x)$ is to be evaluated for all x. Then the design may require that $-1 \leq F^*(x) \leq 1$.

In special cases F*(x) will need to satisfy the same relations as F(x). Also, more efficient programs have been written in assembly language in order to make better use of the particular machine's features (e.g., special register).[12] However, high-level languages are sufficient if their compilers produce efficient object code. Finally, two routines for a function may be required to be compatible in some sense - two routines for one computer but different languages, or for different but compatible computers. One constraint might be that two routines produce the exact, identical result for every argument.

In order to write efficient, accurate routines some mathematical knowledge is necessary in addition to knowing the capabilities of the computer to be used. When choosing a function, one might consider that an associated function may be easier to calculate. Some routines are written with multiple entry points; one routine for different but related functions.

Measuring Error

The error in the approximation of a number can be measured in terms of absolute error and relative error. If x* is an approximation to x then the absolute error in x* is $|x^*-x|$, and the relative error in x* is $|(x^*-x)/x|$ (if x is not zero).[13] Fixed-point computation errors are usually measured in terms of absolute error. If $|x^*-x| = 10^{-d}$, where $d>0$, then the number of correct decimal places in x is approximately equal to d.[14] Since fixed-point numbers must have a fixed number of digits to the right of the decimal (or binary) point, we can think in terms of correct decimal (binary) places and measure error in terms of absolute error.

Floating-point computations are usually analyzed in terms of relative error. The position of the radix point is variable in the floating point representation of a number, and the number range very large. Therefore we can think in terms of the number of significant digits and measure error in terms of relative error. A formula for measuring significant digits is as follows:[15] if y is any approximation to a true value x, then the kth decimal place of y is said to be significant if $|x-y| \leq (\frac{1}{2}) \times 10^{-k}$.

Sources of Error

The difference between the true function value and the generated numerical value is a consequence of truncation error and the round-off errors caused by individual mathematical operations.[16] Another source of error is the approximate numbers used in the function evaluation routines.

Truncation error depends on the evaluation procedure chosen to approximate a function. Truncation error has nothing to do with numerical errors resulting from machine calculations; it is related only to mathematical methods. For example let P(x) be the polynomial used to represent the truncated series for sin(x). Truncation error is the error in P(x), the approximating function. Therefore the absolute truncation error and the relative truncation error would be measured by evaluating $|P(x) - \sin(x)|$ and $|(P(x) - \sin(x))/\sin(x)|$, respectively. $|(P(x) - \sin(x))/$

$\sin(x)|$ can be made as small as possible by retaining enough terms in the power series for $\sin(x)$.[17]

Round-off error is a result of the computations themselves. In the above example, computing $P(x)$ numerically produces an approximation of $\sin(x)$, say $P^*(x)$. The absolute round-off error is $|P^*(x) - P(x)|$, and the relative round-off error is $|(P^*(x) - P(x))/P(x)|$. Combining the absolute truncation error and the absolute round-off error, i.e., $|(P^*(x) - P(x))|$ and $|(P(x) - \sin(x)|$, yields the overall absolute error, $|P^*(x) - \sin(x)|$. In floating-point computations, the magnitude of the relative round-off error should be as small as possible. However, the level of round-off error depends upon the computer used and any special programming techniques that are used, such as extra precision in certain key computations.

Approximate numbers used in function evaluation routines are a third source of error. Coefficients or constants can be stored which are approximate values calculated outside of the function evaluation routine. Also, since the computer is a finite machine, numbers which have an infinite representation must be rounded or the results of multiplication and division operations on finite operands may have to be rounded or truncated. The error introduced in the actual argument will have to be taken into consideration by the overall program of which the function evaluation routine is a part. The argument may already be contaminated with error, so nothing can be done in the function evaluation routine to compensate for this error.

Range Reduction

Polynomial and rational function approximations are the main tools used in supplying accurate approximations in function evaluation routines. In general, the larger the interval for the argument range, the more terms a polynomial or rational approximation must contain in order to approximate a function with a specified accuracy. Therefore, the primary reason for reducing the argument range is to use an approximation that has a small number of terms and can be evaluated more efficiently.

For the many functions with infinite argument ranges, restrictions on the range will be imposed by the inherent limitations of the number representation in the computer used. In most cases of this type, trying to approximate the function accurately over the entire argument range (with a single polynomial or a rational function) would produce too many terms to be practically manageable.

Special properties of some functions, such as periodicity and symmetry, provide rather simple range reduction techniques. Range reduction is also possible in a piecewise fashion when the function has no convenient properties as those above. This can be accomplished by splitting the whole range and using several polynomial or rational approximations, thus obtaining one approximation for each argument.

Transforming an argument in the range reduction routine can cause a considerable increase in the relative round-off error when numerically evaluating a function. Especially, if a function is unstable for certain values of the argument in the range, then a small change in the argument causes a relatively large change in the value of the function. Hence, the range reduction calculations may produce inaccurate results. In order to reduce the probability of round-off error here, extra precision may be used.

CHAPTER III

ROUTINES FOR THE SINE AND COSINE

Polynomial and rational approximations were used here to numerically evaluate sin $(\alpha \pi x)$ and cos $(\beta \pi x)$ for $0 \leq x \leq 1$, where $\alpha = \beta = \frac{1}{4}$. The periodicity of sine and cosine as well as trigonometric identities were used to reduce the range of the approximation to $[0, \pi/4]$. The coefficients for the polynomials and the rational functions used were taken from a table in Hart, et. al. (1968). See Appendix A. Hart presents tables of coefficients for approximating different functions depending on the accuracy desired.

The discussion in the remainder of this chapter will concentrate primarily on the approximations used here for the sine function. The details for the cosine function are similar. The polynomial approximation for sin $(\frac{1}{4}\pi x)$, for x in $[0,1]$ uses a third degree polynomial of the form $xP(x^2)$ with maximum relative error of $10^{-8.49}$. This means obtaining approximately eight-digit accuracy. It can be shown that by examining the coefficients of the polynomial, that Hart used the coefficients of the representative Chebyshev polynomial. Horner's rule (or nested multiplication) was employed to evaluate the polynomials.

Two approaches were used to try to achieve the desired accuracy. When the argument fell in $[\pi/4, \pi/2]$, the first approach used the double angle formula for the sine to further reduce the range to $[0, \pi/4]$. The second approach takes advantage of rewriting sin(x) in terms of cos $((\pi/2) - x)$ to achieve better accuracy when the argument reduces only to $[\pi/4, \pi/2]$ See Appendix C. The reason for the second approach was because the cosine routine had better accuracy between $[0, \pi/4]$ than the sine in $[\pi/4, \pi/2]$. Negative arguments were treated the same as positive arguments by taking the absolute value of the argument before any other computation was done. However, neither approach produced sufficiently accurate results for arguments greater than $\pi/2$. See Appendix C.

For arguments greater than 2π (or less than -2π), steps were taken to evaluate y MOD 2π, for sin(y). To accomplish this, a series of subtractions were performed. Multiples of 2π were subtracted from the argument until the argument was in the range $[0, 2\pi]$. Another method of performing the MOD operation, separating the integer and fraction part of $y/2\pi$, proved to be inefficient for large y.

The reason for losing accuracy for values of the arguments greater than 2π was a result of the subtractions performed. The restriction that Ada/Ed does not support floating-point precision with digits greater than seven, forced all computations to be performed with single precision. On the VAX 11/780, described in the next chapter, single precision provides approximately seven decimal digit precision. Hence, the result of the subtractions were rounded in order to fit the precision of the computation. At different times in the testing of these routines constants were used in place of having the computer perform certain operations. However, the subtractions and the fact that Ada/Ed does not support multiple-precision calculations, outweighed these attempts to improve the accuracy.

The routine for the rational approximation of the sine function in this paper produced similar results. Sin $(\frac{\pi x}{4})$ was represented by a rational function of the form $xP(x^2) / Q(x^2)$. The degree of the numerator was three and the degree of the denominator was one. The maximum relative error desired was $10^{-8.63}$. The cosine function was approximated with a third degree polynomial of the form $P(x^2)$ with a maximum relative error of $10^{-7.49}$, and with a rational function, $P(x^2) / Q(x^2)$, with a maximum relative error of $10^{-7.54}$.

To compare the different routines for efficiency the number of operations performed had to be counted and then compared based on the time it took the computer to perform those operations. Instruction times for the VAX 11/780 were found in the Datapro Research Corporation publication, April 1981. Floating-point instruction times, in microseconds, for 32-bit operands are 0.8 for addition and subtraction, 1.2 for multiplication, and 4.2 for division.

Table 1 shows the number of operations that differ between the routines for the sine function. Table 1 assumes that the original argument was greater than 2π and reduces to the interval $[0, \pi/2]$, after first being reduced to $[0, 2\pi]$ and then $[0, \pi]$. That way, the maximum number of operations are performed.

TABLE 1

NUMBER OF OPERATIONS THAT DIFFER FOR EACH ROUTINE*

Routine	Addition/Subtraction	Multiplication	Division
SINE	1	9	6
SINEX	1	4	4
SINCHG	1	11	1
SIN PAT	1	9	8

*Operations that were the same for each routine were not counted.

The accuracy of the routines in Table 1 can be compared by studying the table of sample output in Appendix C.

CHAPTER IV

FACTS ABOUT THE DEC VAX 11/780

The VAX 11/780 is a 32-bit multi-user, multiprogramming system that features 4.3 billion bytes of virtual address spaces, a maximum program size of 2 billion bytes, 244 instructions (9 address modes, 6 data types), and 4 hierarchical protection modes. The VAX 11/780 is manufactured by Digital Equipment Corporation (DEC), Maynard, Massachusetts, which is the world's largest manufacturer of minicomputer systems.

The processor of the VAX 11/780 provides 32-bit addressing, sixteen 32-bit general registers, and 32 interrupt priority levels. The effective memory access time is 280 nanoseconds; cycle time, 600 nanoseconds. The instruction set operates on integer and floating-point operands, character and packed decimal strings, and bit fields, and supports nine fundamental addressing modes.

The basic data unit is a 32-bit word. Integers can be 8-bit bytes, 16-bit words, 32-bit long words, and 64-bit quadwords. Two floating-point formats are available: single precision that uses a 4-byte format, and double precision that uses an 8-byte format. The 4-byte format provides approximately 7 decimal digits of precision, while the 8-byte format provides approximately 16 decimal digits of precision.

The instruction set provides 32-bit addressing, 32-bit I/O instructions, and 32-bit arithmetic. The 244 basic instruction set can be grouped into related classes based on their functions and use. The internal code is ASCII for test-oriented data; binary for calculations.

This information on the VAX 11/780 was taken from Datapro Corporation's publication on minicomputers. More information on the VAX 11/780 as well as many other minicomputers can be found in this guide.

CHAPTER 9

CONCLUSION

Presented in this paper were routines to numerically evaluate the sine and cosine functions. The routines for the sine were only discussed because the routines and analysis for the cosine were similar. In fact, any routine which computes sin(x) can do so for cos(x) by using the identity cos(x) = sin ($(\frac{\pi}{2})$ - x). Depending on the specifications of the package which is to combine these routines with other function routines, only one routine may be used for the sine and cosine together. Also expected from the package which will contain these routines is portability, which is one of the design objectives of the Ada programming language.

The main problem of accuracy in the routines was the result of Ada/Ed's single precision limitation. Computations performed with multi-precision minimize the round-off error apparent in single-precision computations.

APPENDIX A

Below are the polynomial and rational approximations used to numerically evaluate sin ($\alpha \pi x$) and cos ($\beta \pi x$) for $0 \leq x \leq 1$, where $\alpha = \beta = \frac{1}{4}$.

Polynomial Approximations

sin ($\frac{1}{4}\pi x$) $\simeq 0.7853981x - 0.0807454\ x^3 + 0.0024900\ x^5 - 0.0000359\ x^7$; maximum relative error = $10^{-8.49}$.

cos ($\frac{1}{4}\pi x$) $\simeq 0.9999999 - 0.3084241\ x^2 + 0.0158496\ x^4 - 0.0003187\ x^6$; maximum relative error = $10^{-7.49}$.

Rational Approximations

$$\sin\left(\tfrac{1}{4}\pi x\right) \approx \frac{52.818601\ x - 4.6448004\ x^3 + 0.0867545\ x^5}{67.250731\ x + 1.0\ x^3};$$

maximum relative error = $10^{-8.63}$.

$$\cos\left(\tfrac{1}{4}\pi x\right) \approx \frac{47.687292 - 13.708000\ x^2 + 0.4478223\ x^4}{47.687290 + 1.0\ x^2};$$

maximum relative error = $10^{-7.54}$.

APPENDIX B

Routine SINE

```
--THIS PROGRAM IS A POLYNOMIAL APPROXIMATION FOR THE SIN(X), FOR ALL X,
--USING AN APPROXIMATION RANGE OF [0,PI/4]
WITH TEXT_IO; USE TEXT_IO;
PACKAGE MY_IO IS NEW FLOAT_IO(FLOAT);
WITH MY_IO; USE MY_IO;
PROCEDURE SINE IS
        PI,X,Z,B,B1,V,ANS,X1,X2,X3:FLOAT;
        FLAG:BOOLEAN;
        FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS SEPARATE;
BEGIN
        PI:=3.1415926;
        FOR J IN 1..15 LOOP
                PUT_LINE('INPUT VALUE OF X FOR SIN(X)');
                PUT_LINE('?');
                FLAG:=TRUE; --SINE IS POS
                GET(X);
                Z:=ABS(X);
                --FOLLOWING WHILE STATEMENTS WILL COMPUTE Z MOD 2*PI
                WHILE Z>100.0*PI LOOP
                        Z:=Z-100.0*PI;
                        PUT('Z= ');PUT(Z);NEW_LINE;
                END LOOP;
                WHILE Z>20.0*PI LOOP
                        Z:=Z-20.0*PI;
                        PUT('Z= ');PUT(Z);NEW_LINE;
                END LOOP;
                WHILE Z>2.0*PI LOOP
                        Z:=Z-2.0*PI;
                        PUT('Z= ');PUT(Z);NEW_LINE;
                END LOOP;
                X1:=Z;
                PUT('ARGUMENT NOW IN [0,2*PI]');NEW_LINE;
                --CHECK FOR INTERVAL OF X1=Z=ABS(X)
                IF X1>PI THEN
                        X2:=X1-PI;
                        PUT('X2= ');PUT(X2);NEW_LINE;
                        FLAG:=FALSE;
                ELSE
                        X2:=X1;
                END IF;
                IF X2>PI/2.0 THEN
                        X3:=PI-X2;  --BETWEEN 0 AND PI/2
                        PUT('X3= ');PUT(X3);NEW_LINE;
                ELSE
                        X3:=X2;
                END IF;
                IF X3>PI/4.0 THEN --USE DOUBLE ANGLE FORMULA FOR SIN(X)
                                  --TO REDUCE ARGUMENT TO [0,PI/4]
                        B:=X3/2.0; -- FIRST, HALVE ARGUMENT
                        B1:=PI/2.0-B;
                        PUT('B1=PI/2-B, B IN [0,2*PI] IS ');PUT(B1);NEW_LINE;
                        B:=B*(4.0/PI);
                        B1:=B1*(4.0/PI);
                        V:=2.0*COMPUTE(B);  --2SIN(B)
                        ANS:=V*COMPUTE(B1);  --2SIN(B)SIN(PI/2-B)=
                                             --  2SIN(B)COS(B)=SIN(2B)
                ELSE
                        X3:=X3*(4.0/PI);
                        ANS:=COMPUTE(X3);  --SIN(X3)
```

Routine SINE - Continued

```
                END IF;
                IF FLAG=FALSE THEN
                        ANS:=-ANS;
                END IF;
                IF X<0.0 THEN
                        ANS:=-ANS;
                END IF;
                PUT ('THE SINE OF '); PUT(X); PUT(' IS '); PUT(ANS);
                NEW_LINE;NEW_LINE;
        END LOOP;
        PUT_LINE('DONE');
END SINE;
        SEPARATE(SINE)
                FUNCTION COMPUTE(Z1:FLOAT) RETURN FLOAT IS
                        --A FUNCTION THAT COMPUTES SIN(Z1)
                        P:ARRAY(0..3)OF FLOAT;
                        N:INTEGER;
                        S,VALUE:FLOAT;
                        BEGIN
                        N:=3; --DEGREE OF POLY
                --COEFFICIENTS OF Z1*P(Z1**2) FOR SIN(1/4*PI*X), 0<=X<=1
                        P(0):=0.7853981;
                        P(1):=-0.0807454;
                        P(2):=0.0024900;
                        P(3):=-0.0000359;
                        S:=P(N);
                        FOR I IN REVERSE 0..N-1 LOOP
                                S:=(S*(Z1*Z1))+P(I);
                        END LOOP;
                        VALUE:=Z1*S; PUT('VALUE= '); PUT(VALUE); NEW_LINE;
                        RETURN VALUE;
                        END COMPUTE;
```

Routine SINCHG

```
--THIS PROGRAM COMPUTES SIN(X), FOR ALL X, USING A POLYNOMIAL
--APPROXIMATION, AND AN APPORXIMATION RANGE OF [0,PI/4]
WITH TEXT_IO; USE TEXT_IO;
PACKAGE MY_IO IS NEW FLOAT_IO(FLOAT);
WITH MY_IO; USE MY_IO;
PROCEDURE SINE IS
        S,I,PI,PII,PIH,PIL,X,Z,B,B1,V,ANS,X1,X2,X3:FLOAT;
        FLAG:BOOLEAN;
        FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS SEPARATE;
BEGIN
        PI:=3.1415926;
        PII:=0.3183098; --1/PI
        PIH:=0.7853981; --PI/4.0
        PIL:=1.5707963; --PI/2.0
        FOR J IN 1..15 LOOP
                PUT_LINE('INPUT VALUE OF X FOR SIN(X)');
                PUT_LINE('?');
                FLAG:=TRUE; --SINE IS POS
                GET(X);
                Z:=ABS(X);
                --THE FOLLOWING WHILE STATEMENTS WILL COMPUTE Z MOD 2*PI
                WHILE Z>314.15926 LOOP    --Z>100*PI?
                        Z:=Z-314.15926;
                        PUT('Z= '); PUT(Z); NEW_LINE;
                END LOOP;
                WHILE Z>62.831853 LOOP    --Z>20*PI?
                        Z:=Z-62.831853;
                        PUT('Z= '); PUT(Z); NEW_LINE;
                END LOOP;
                WHILE Z>6.2831853 LOOP     --Z>2*PI?
                        Z:=Z-6.2831853;
                        PUT('Z= '); PUT(Z); NEW_LINE;
                END LOOP;
                X1:=Z;    --Z<=2*PI
                PUT('ARGUMENT NOW IN [0,2*PI]'); NEW_LINE;
                --CHECK FOR INTERVAL OF X1=Z=ABS(X)
                IF X1>PI THEN
                        X2:=X1-PI;
                        PUT('X2= ');PUT(X2);NEW_LINE;
                        FLAG:=FALSE;
                ELSE
                        X2:=X1;
                END IF;
                IF X2>PIL THEN
                        X3:=PI-X2;  --BETWEEN 0 AND PI/2
                        PUT('X3= ');PUT(X3);NEW_LINE;
                ELSE
                        X3:=X2;
                END IF;
                IF X3>PIH THEN  --USE DOUBLE ANGLE FORMULA FOR SIN(2X) TO
                                --REDUCE ARGUMENT TO [0,PI/4]
                        B:=X3/2.0;  --FIRST, HALVE ARGUMENT
                        B1:=PIL-B;  --PI/2-B
                        PUT('B1 = PI/2-B, B IN [0,PI/4] = ');PUT(B1);NEW_LINE;
                        B:=B*(4.0*PII);
                        B1:=B1*(4.0*PII);
                        V:=2.0*COMPUTE(B); --2*SIN(B)
                        ANS:=V*COMPUTE(B1); --2*SIN(B)*SIN(PI/2-B)=
                                            --  2*SIN(B)*COS(B)=SIN(2B)
```

Routine SINCHG - Continued

```
                ELSE
                        X3:=X3*(4.0*PII);
                        ANS:=COMPUTE(X3);   --SIN(X3)
                END IF;
                IF FLAG=FALSE THEN
                        ANS:=-ANS;
                END IF;
                IF X<0.0 THEN
                        ANS:=-ANS;
                END IF;
                PUT ('THE SINE OF '); PUT(X); PUT(' IS '); PUT(ANS);
                NEW_LINE;NEW_LINE;
        END LOOP;
        PUT_LINE('DONE');
END SINE;
        SEPARATE(SINE)
                FUNCTION COMPUTE(Z1:FLOAT) RETURN FLOAT IS
                        --A FUNCTION THAT COMPUTES SIN(Z1)
                        P:ARRAY(0..3)OF FLOAT;
                        N:INTEGER;
                        S,VALUE:FLOAT;
                        BEGIN
                        N:=3; --DEGREE OF POLY
                --COEFFICIENTS OF Z1*P(Z1**2) FOR SIN(1/4*PI*X), 0<=X<=1
                        P(0):=0.7853981;
                        P(1):=-0.0807454;
                        P(2):=0.0024900;
                        P(3):=-0.0000359;
                        S:=P(N);
                        FOR I IN REVERSE 0..N-1 LOOP
                                S:=(S*(Z1*Z1))+P(I);
                        END LOOP;
                        VALUE:=Z1*S; PUT('VALUE= '); PUT(VALUE); NEW_LINE;
                        RETURN VALUE;
                        END COMPUTE;
```

Routine SINEX

```
-- THIS PROGRAM IS A POLYNOMIAL APPROXIMATION FOR SIN(X) IN AN APPROXIMATION
-- RANGE OF [0,PI/4]
WITH TEXT_IO; USE TEXT_IO;
PACKAGE MY_IO IS NEW FLOAT_IO(FLOAT);
WITH MY_IO; USE MY_IO;
PROCEDURE SINEX IS
        PI,ANS,X1,X2,X3,X,Z:FLOAT;
        FLAG:BOOLEAN;
        FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS SEPARATE;
        FUNCTION CHANGE(Z2:FLOAT)RETURN FLOAT IS SEPARATE;
BEGIN
        PI:=3.1415926;
        FOR J IN 1..15 LOOP
                PUT_LINE('INPUT VALUE OF X FOR SIN(X)');
                PUT_LINE('?');
                GET(X);
                FLAG:=TRUE; --COS POS
                Z:=ABS(X);
                -- THE FOLLOWING WHILE STATEMENTS COMPUTE Z MOD 2*PI
                WHILE Z>100.0*PI LOOP
                        Z:=Z-100.0*PI;
                END LOOP;
                WHILE Z>20.0*PI LOOP
                        Z:=Z-20.0*PI;
                END LOOP;
                WHILE Z>2.0*PI LOOP
                        Z:=Z-2.0*PI;
                END LOOP;
                --ONCE Z IS IN [0,2*PI], BEGIN TO REDUCE TO [0,PI/4] USING
                --TRIG IDENTITIES
                X1:=Z;
                IF X1>PI THEN
                        X2:=X1-PI;
                        FLAG:=FALSE;
                ELSE
                        X2:=X1;
                END IF;
                IF X2>PI/2.0 THEN
                        X3:=PI-X2;  --BETWEEN 0 AND PI/2
                ELSE
                        X3:=X2;
                END IF;
                IF X3>PI/4.0 THEN  --IF ARGUMENT NOW GREATER THAN PI/4
                                   --BUT LESS THAN PI/2, USE IDENTITY
                                   --SIN(X)=COS(PI/2-X), WHICH REDUCES
                                   --ARGUMENT TO [0,PI/4]
                        X3:=PI/2.0-X3;
                        X3:=X3*(4.0/PI);
                        ANS:=CHANGE(X3);  --COS(PI/2-X3)
                ELSE
                        X3:=X3*(4.0/PI);
                        ANS:=COMPUTE(X3);
                END IF;
                IF FLAG=FALSE THEN
                        ANS:=-ANS;
                END IF;
                IF X<0.0 THEN
                        ANS:=-ANS;
                END IF;
```

Routine SINEX - Continued

```
                PUT('THE SINE OF '); PUT(X); PUT(' IS '); PUT(ANS);
                NEW_LINE;NEW_LINE;
        END LOOP;
        PUT_LINE('DONE');
END SINEX;
        SEPARATE(SINEX)
                FUNCTION CHANGE(Z2:FLOAT) RETURN FLOAT IS
                        --A FUNCTION THAT COMPUTES COS(Z2)=COS(PI/2-X3)=SIN(X3)
                        P:ARRAY(0..3)OF FLOAT;
                        N:INTEGER;
                        S,VALUE:FLOAT;
                        BEGIN
                        N:=3; --DEGREE OF POLY
                        P(0):=0.9999999;
                        P(1):=-0.3084241;
                        P(2):=0.0158496;
                        P(3):=-0.0003187;
                        S:=P(N);
                        FOR I IN REVERSE 0..N-1 LOOP
                                S:=(S*(Z2*Z2))+P(I);
                        END LOOP;
                        VALUE:=S; PUT('VALUE= ');PUT(VALUE);NEW_LINE;
                        RETURN VALUE;
                        END CHANGE;
        SEPARATE(SINEX)
                FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS
                -- THE VALUE OF SIN(Z1)
                        P:ARRAY(0..3)OF FLOAT;
                        N:INTEGER;
                        S,VALUE:FLOAT;
                        BEGIN
                        N:=3; --DEG OF POLY
                        P(0):=0.7853981;
                        P(1):=-0.0807454;
                        P(2):=0.0024900;
                        P(3):=-0.0000359;
                        S:=P(N);
                        FOR I IN REVERSE 0..N-1 LOOP
                                S:=(S*(Z1*Z1))+P(I);
                        END LOOP;
                        VALUE:=Z1*S; PUT('VALUE= ');PUT(VALUE);NEW_LINE;
                        RETURN VALUE;
                        END COMPUTE;
```

Routine SINRAT

```
--THIS PROGRAM IS A RATIONAL APPROXIMATION FOR SIN(X), FOR ALL X,
--USING AN APPROXIMATION INTERVAL OF [0,PI/4]
WITH TEXT_IO; USE TEXT_IO;
PACKAGE MY_IO IS NEW FLOAT_IO(FLOAT);
WITH MY_IO; USE MY_IO;
PROCEDURE SINRAT IS
PI,X,Z,B,B1,V,ANS,X1,X2,X3:FLOAT;
FLAG:BOOLEAN;
FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS SEPARATE;
BEGIN
        PI:=3.1415926;
        FOR J IN 1..15 LOOP
                PUT_LINE('INPUT VALUE OF X FOR SIN(X)');
                PUT_LINE('?');
                FLAG:=TRUE; --SINE POS
                GET(X);
                Z:=ABS(X);
                -- THE FOLLOWING WHILE STATEMENTS WILL COMPUTE Z MOD 2*PI
                WHILE Z>100.0*PI LOOP
                        Z:=Z-100.0*PI;
                END LOOP;
                WHILE Z>20.0*PI LOOP
                        Z:=Z-20.0*PI;
                END LOOP;
                WHILE Z>2.0*PI LOOP
                        Z:=Z-2.0*PI;
                END LOOP;
                X1:=Z;
                --CHECK FOR INTERVAL OF X1=Z=ABS(X)
                IF X1>PI THEN
                        X2:=X1-PI;
                        FLAG:=FALSE;
                ELSE
                        X2:=X1;
                END IF;
                IF X2>PI/2.0 THEN
                        X3:=PI-X2;  --NOW BETWEEN 0 AND PI/2
                ELSE
                        X3:=X2;
                END IF;
                IF X3>PI/4.0 THEN --USE DOUBLE ANGLE FORMULA FOR SIN(2X)
                        B:=X3/2.0;  --FIRST, HALVE ARGUMENT
                        B1:=PI/2.0-B;
                        B:=B*(4.0/PI);
                        B1:=B1*(4.0/PI);
                        V:=2.0*COMPUTE(B);  --2SIN(B)
                        ANS:=V*COMPUTE(B1);  --2SIN(B)SIN(PI/2-B)=
                                             --  2SIN(B)COS(B)=SIN(2B)
                ELSE
                        X3:=X3*(4.0/PI);
                        ANS:=COMPUTE(X3);  --SIN(X3)
                END IF;
                IF FLAG=FALSE THEN
                        ANS:=-ANS;
                END IF;
                IF X<0.0 THEN
                        ANS:=-ANS;
                END IF;
```

Routine SINRAT - Continued

```
                PUT ('THE SINE OF '); PUT(X); PUT(' IS '); PUT(ANS);
                NEW_LINE; NEW_LINE;
        END LOOP;
        PUT_LINE('DONE');
END SINRAT;
SEPARATE(SINRAT)
        FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS
        --A RATIONAL FUNCTION THAT COMPUTES SIN(Z1)
        P:ARRAY(0..2)OF FLOAT;
        Q:ARRAY(0..1)OF FLOAT;
        N,M:INTEGER;
        S,T,VALUE:FLOAT;
        BEGIN
                N:=2; --DEGREE OF P
                M:=1; --DEGREE OF Q
                P(0):=52.818601;
                P(1):=-4.6448004;
                P(2):=0.0867545;
                Q(0):=67.250731;
                Q(1):=1.0;
                S:=P(N);
                FOR I IN REVERSE 0..N-1 LOOP
                        S:=(S*(Z1*Z1))+P(I);
                END LOOP;
                T:=Q(M);
                FOR I IN REVERSE 0..M-1 LOOP
                        T:=(T*(Z1*Z1))+Q(I);
                END LOOP;
                VALUE:=S/T; --P(X**2)/Q(X**2)
                VALUE:=Z1*VALUE; --X*P(X**2)/Q(X**2)
                RETURN VALUE;
        END COMPUTE;
```

Routine CSINE

```
--THIS PROGRAM IS A POLYNOMIAL APPROXIMATION FOR COS(X), FOR ALL X,
--WITH A RANGE OF APPROXIMATION OF [0,PI/4]
WITH TEXT_IO; USE TEXT_IO;
PACKAGE MY_IO IS NEW FLOAT_IO(FLOAT);
WITH MY_IO; USE MY_IO;
PROCEDURE CSINE IS
        PI,ANS,X1,X2,X3,B,B1,V,X,Z:FLOAT;
        FLAG:BOOLEAN;
        FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS SEPARATE;
BEGIN
        PI:=3.1415926;
        FOR J IN 1..15 LOOP
                PUT_LINE('INPUT VALUE OF X FOR COS(X)');
                PUT_LINE('?');
                GET(X);
                FLAG:=TRUE; --COS POS
                Z:=ABS(X);
                --THE FOLLOWING WHILE STATEMENTS COMPUTE Z MOD 2*PI
                WHILE Z>100.0*PI LOOP
                        Z:=Z-100.0*PI;
                END LOOP;
                WHILE Z>20.0*PI LOOP
                        Z:=Z-20.0*PI;
                END LOOP;
                WHILE Z>2.0*PI LOOP
                        Z:=Z-2.0*PI;
                END LOOP;
                --ONCE Z IS IN [0,2*PI] TAKE STEPS TO REDUCE ARGUMENT
                --TO RANGE [0,PI/4]
                X1:=Z;
                IF X1>PI THEN
                        X2:=X1-PI;
                        FLAG:=FALSE;
                ELSE
                        X2:=X1;
                END IF;
                IF X2>PI/2.0 THEN
                        X3:=PI-X2;
                        IF FLAG=FALSE THEN
                                FLAG:=TRUE;
                        ELSE
                                FLAG:=FALSE;
                        END IF;
                ELSE
                        X3:=X2;
                END IF;
                IF X3>PI/4.0 THEN   --FOR FINAL REDUCTION, IF ARGUMENT GREATER
                                    --THAN PI/4 USE IDENTITY FOR COS(2X)
                        B:=X3/2.0;   --FIRST, HALVE ARGUMENT
                        B:=B*(4.0/PI);
                        B1:=COMPUTE(B)*COMPUTE(B); --COS(B)*COS(B)
                        V:=2.0*B1; --2*COS(B)*COS(B)
                        ANS:=V-1.0;   --(2*COS(B)*COS(B))-1 = COS(2B)
                ELSE
                        X3:=X3*(4.0/PI);
                        ANS:=COMPUTE(X3);
                END IF;
                IF FLAG=FALSE THEN
                        ANS:=-ANS;
```

Routine CSINE - Continued

```
                END IF;
                PUT('THE COSINE OF '); PUT(X); PUT(' IS '); PUT(ANS);
                NEW_LINE;NEW_LINE;
        END LOOP;
        PUT_LINE('DONE');
END CSINE;
        SEPARATE(CSINE)
                FUNCTION COMPUTE(Z1:FLOAT) RETURN FLOAT IS
                        --THE VALUE OF COS(Z1)
                        P:ARRAY(0..3)OF FLOAT;
                        N:INTEGER;
                        S,VALUE:FLOAT;
                        BEGIN
                        N:=3; --DEGREE OF POLY
                        --COEFFICIENTS OF POLY P:
                        P(0):=0.9999999;
                        P(1):=-0.3084241;
                        P(2):=0.0158496;
                        P(3):=-0.0003187;
                        S:=P(N); --NESTED MULTICPLICATIO TO EVALUATE P
                        FOR I IN REVERSE 0..N-1 LOOP
                                S:=(S*(Z1*Z1))+P(I);
                        END LOOP;
                        VALUE:=S; PUT('VALUE= ');PUT(VALUE);NEW_LINE;
                        RETURN VALUE;
                        END COMPUTE;
```

Routine CSINX

```
-- THIS PROGRAM IS A POLYNOMIAL APPROXIMATION FOR COS(X) IN AN APPROXIMATION
-- RANGE OF [0,PI/4]
WITH TEXT_IO; USE TEXT_IO;
PACKAGE MY_IO IS NEW FLOAT_IO(FLOAT);
WITH MY_IO; USE MY_IO;
PROCEDURE CSINX IS
        PI,ANS,X1,X2,X3,X,Z:FLOAT;
        FLAG:BOOLEAN;
        FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS SEPARATE;
        FUNCTION CHANGE(Z2:FLOAT)RETURN FLOAT IS SEPARATE;
BEGIN
        PI:=3.1415926;
        FOR J IN 1..15 LOOP
                PUT_LINE('INPUT VALUE OF X FOR COS(X)');
                PUT_LINE('?');
                GET(X);
                FLAG:=TRUE; --COS POS
                Z:=ABS(X);
                -- THE FOLLOWING WHILE STATEMENTS COMPUTE Z MOD 2*PI
                WHILE Z>100.0*PI LOOP
                        Z:=Z-100.0*PI;
                END LOOP;
                WHILE Z>20.0*PI LOOP
                        Z:=Z-20.0*PI;
                END LOOP;
                WHILE Z>2.0*PI LOOP
                        Z:=Z-2.0*PI;
                END LOOP;
                --ONCE Z IS IN [0,2*PI], BEGIN TO REDUCE TO [0,PI/4] USING
                --TRIG IDENTITIES
                X1:=Z;
                IF X1>PI THEN
                        X2:=X1-PI;
                        FLAG:=FALSE;
                ELSE
                        X2:=X1;
                END IF;
                IF X2>PI/2.0 THEN
                        X3:=PI-X2;
                        IF FLAG=FALSE THEN
                                FLAG:=TRUE;
                        ELSE
                                FLAG:=FALSE;
                        END IF;
                ELSE
                        X3:=X2;
                END IF;
                IF X3>PI/4.0 THEN   --IF ARGUMENT NOW GREATER THAN PI/4
                                    --BUT LESS THAN PI/2, USE IDENTITY
                                    --COS(X)=SIN(PI/2-X), WHICH REDUCES
                                    --ARGUMENT TO [0,PI/4]
                        X3:=PI/2.0-X3;
                        X3:=X3*(4.0/PI);
                        ANS:=CHANGE(X3);  --SIN(PI/2-X3)
                ELSE
                        X3:=X3*(4.0/PI);
                        ANS:=COMPUTE(X3);
                END IF;
```

Routine CSINX - Continued

```
                IF FLAG=FALSE THEN
                        ANS:=-ANS;
                END IF;
                PUT('THE COSINE OF '); PUT(X); PUT(' IS '); PUT(ANS);
                NEW_LINE;NEW_LINE;
        END LOOP;
        PUT_LINE('DONE');
END CSINX;
        SEPARATE(CSINX)
                FUNCTION COMPUTE(Z1:FLOAT) RETURN FLOAT IS
                        --THE VALUE OF COS(Z1)
                        P:ARRAY(0..3)OF FLOAT;
                        N:INTEGER;
                        S,VALUE:FLOAT;
                        BEGIN
                        N:=3; --DEGREE OF POLY
                        P(0):=0.9999999;
                        P(1):=-0.3084241;
                        P(2):=0.0158496;
                        P(3):=-0.0003187;
                        S:=P(N);
                        FOR I IN REVERSE 0..N-1 LOOP
                                S:=(S*(Z1*Z1))+P(I);
                        END LOOP;
                        VALUE:=S; PUT('VALUE= ');PUT(VALUE);NEW_LINE;
                        RETURN VALUE;
                        END COMPUTE;
        SEPARATE(CSINX)
                FUNCTION CHANGE(Z2:FLOAT)RETURN FLOAT IS
                -- THE VALUE OF SIN(Z2)=SIN(PI/2-X3)=COS(X3)
                        P:ARRAY(0..3)OF FLOAT;
                        N:INTEGER;
                        S,VALUE:FLOAT;
                        BEGIN
                        N:=3; --DEG OF POLY
                        P(0):=0.7853981;
                        P(1):=-0.0807454;
                        P(2):=0.0024900;
                        P(3):=-0.0000359;
                        S:=P(N);
                        FOR I IN REVERSE 0..N-1 LOOP
                                S:=(S*(Z2*Z2))+P(I);
                        END LOOP;
                        VALUE:=Z2*S; PUT('VALUE= ');PUT(VALUE);NEW_LINE;
                        RETURN VALUE;
                        END CHANGE;
```

Routine COSRAT

```
--THIS PROGRAM COMPUTES COS(X) FOR ALL X, USING AN APPROXIMATION RANGE OF
--[0,PI/4].  A RATIONAL FUNCTION IS USED TO EVALUATE THE FUNCTION.
WITH TEXT_IO; USE TEXT_IO;
PACKAGE MY_IO IS NEW FLOAT_IO(FLOAT);
WITH MY_IO; USE MY_IO;
PROCEDURE COSRAT IS
PI,ANS,X1,X2,X3,B,B1,V,X,Z:FLOAT;
FLAG:BOOLEAN;
FUNCTION COMPUTE(Z1:FLOAT)RETURN FLOAT IS SEPARATE;
BEGIN
        PI:=3.1415926;
        FOR J IN 1..15 LOOP
        PUT_LINE('INPUT VALUE OF X FOR COS(X)');
        PUT_LINE('?');
        GET(X);
        FLAG:=TRUE;  --COS POS
        Z:=ABS(X);
        -- THE FOLLOWING WHILE STATEMENTS ARE TO COMPUTE Z MOD 2*PI
        WHILE Z>100.0*PI LOOP
                Z:=Z-100.0*PI;
        END LOOP;
        WHILE Z>20.0*PI LOOP
                Z:=Z-20.0*PI;
        END LOOP;
        WHILE Z>2.0*PI LOOP
                Z:=Z-2.0*PI;
        END LOOP;
        --TEST RANGE OF ARGUMENT AFTER REDUCED IN [0,2*PI]
        IF X1>PI THEN
                X2:=X1-PI;
                FLAG:=FALSE;
        ELSE
                X2:=X1;
        END IF;  --ARGUMENT NOW IN [0,PI]
        IF X2>PI/2.0 THEN
                X3:=PI-X2;
                IF FLAG=FALSE THEN
                        FLAG:=TRUE;
                ELSE
                        FLAG:=FALSE;
                END IF;
        ELSE
                X3:=X2;
        END IF; --ARGUMENT NOW IN [0,PI/2] WITH PROPER SIGN
        IF X3>PI/4.0 THEN   --USE DOUBLE ANGLE FORMULA FOR COS TO REDUCE TO
                            --RANGE [0,PI/4]
                B:=X3/2.0;
                B:=B*(4.0/PI);
                B1:=COMPUTE(B)*COMPUTE(B);   --COS(B)*COS(B)
                V:=2.0*B1;  --2*COS(B)*COS(B)
                ANS:=V-1.0; --(2*COS(B)*COS(B))-1 = COS(2B)
        ELSE
                X3:=X3*(4.0/PI);
                ANS:=COMPUTE(X3);
        END IF;
        IF FLAG=FALSE THEN
                ANS:=-ANS;
```

Routine COSRAT - Continued

```
        END IF;
        PUT('THE COSINE OF '); PUT(X); PUT(' IS '); PUT(ANS);
        NEW_LINE;NEW_LINE;
        END LOOP;
        PUT_LINE('DONE');
END COSRAT;
SEPARATE(COSRAT)
        FUNCTION COMPUTE(Z1:FLOAT) RETURN FLOAT IS
        -- RATIONAL APPROXIMATION TO COS(Z1)
                P:ARRAY(0..2)OF FLOAT;
                Q:ARRAY(0..1)OF FLOAT;
                N,M:INTEGER;
                S,T,VALUE:FLOAT;
                BEGIN
                        N:=2;  --DEG OF P
                        M:=1;  --DEG OF Q
                        -- FOLLOWING CONSTANTS ARE COEFFICIENTS FOR POLYS USED
                        --IN RATIONAL FUNCTION
                        P(0):=47.687292;
                        P(1):=-13.708000;
                        P(2):=0.4478223;
                        Q(0):=47.687290;
                        Q(1):=1.0;
                        S:=P(N); --BEGIN NESTED MULTIPLICATION TO EVALUATE
                                --POLY P
                        FOR I IN REVERSE 0..N-1 LOOP
                                S:=(S*(Z1*Z1))+P(I);
                        END LOOP;
                        T:=Q(M);
                        FOR I IN REVERSE 0..M-1 LOOP
                                T:=(T*(Z1*Z1))+Q(I);
                        END LOOP;
                        VALUE:=S/T;  --P(X**2)/Q(X**2)
                        RETURN VALUE;
                        END COMPUTE;
```

TABLE 2

SAMPLE OUTPUT OF THE SINE APPROXIMATION

Argument	Routine Name				
	SINE	SINEX	SINCHG	SINRAT	TRUE VALUE*
0.25	2.474039×10^{-1}	2.474039×10^{-1}	2.474038×10^{-1}	2.474040×10^{-1}	2.474039×10^{-1}
0.50	4.794255×10^{-1}	4.794255×10^{-1}	4.794254×10^{-1}	4.794256×10^{-1}	4.794255×10^{-1}
0.75	6.816387×10^{-1}	6.816387×10^{-1}	6.816385×10^{-1}	6.816389×10^{-1}	6.816387×10^{-1}
0.7853981	7.071067×10^{-1}	7.071067×10^{-1}	7.071065×10^{-1}	7.071068×10^{-1}	7.071067×10^{-1}
1.0	8.414702×10^{-1}	8.414710×10^{-1}	8.414697×10^{-I}	$8.414721 \times$ 10-1	8.414709×10^{-1}
1.57	9.999997×10^{-1}	9.999997×10^{-1}	9.999992×10^{-1}	1.000000×10^{-1}	9.999997×10^{-1}
3.14	1.592635×10^{-3}	1.592635×10^{-3}	1.592635×10^{-3}	1.592635×10^{-3}	1.592547×10^{-3}
6.28	-3.185266×10^{-3}	-3.185266×10^{-3}	-3.185266×10^{-3}	-3.185267×10^{-3}	-3.185092×10^{-3}
10.0	-5.440215×10^{-1}	-5.440215×10^{-1}	-5.440213×10^{-1}	-5.440216×10^{-1}	-5.440211×10^{-1}
25.0	-1.323496×10^{-1}	-1.323496×10^{-1}	-1.323495×10^{-1}	-1.323496×10^{-1}	-1.323517×10^{-1}
50.0	-2.623679×10^{-1}	-2.623679×10^{-1}	-2.623678×10^{-1}	-2.623679×10^{-1}	-2.623748×10^{-1}
75.0	-3.877800×10^{-1}	-3.877800×10^{-1}	-3.877799×10^{-1}	-3.877801×10^{-1}	-3.877816×10^{-1}
100.0	-5.063607×10^{-1}	-5.063607×10^{-1}	-5.063605×10^{-1}	-5.063607×10^{-1}	-5.063656×10^{-1}
500.0	-4.678073×10^{-1}	-4.678073×10^{-1}	-4.678072×10^{-1}	-4.678075×10^{-1}	-4.677717×10^{-1}
1000.0	8.269249×10^{-1}	8.269259×10^{-1}	8.269244×10^{-1}	8.269269×10^{-1}	8.268795×10^{-1}

*7-digit accuracy values taken from M. Abramowitz and I Stegun, eds., Handbook of Mathematical Functions (New York, 1964).

TABLE 3

SAMPLE OUTPUT OF THE SINE APPROXIMATION
FOR ARGUMENTS IN $\pi/4$, $\pi/2$

Argument	Routine Name			
	SINE	SINEX	SINRAT	TRUE VALUE*
0.79	7.103509×10^{-1}	7.103532×10^{-1}	7.103555×10^{-1}	7.103532×10^{-1}
0.80	7.173538×10^{-1}	7.173561×10^{-1}	7.173583×10^{-1}	7.173560×10^{-1}
0.85	7.512768×10^{-1}	7.512804×10^{-1}	7.512822×10^{-1}	7.512804×10^{-1}
0.90	7.833255×10^{-1}	7.833269×10^{-1}	7.833282×10^{-1}	7.833269×10^{-1}
0.95	8.134144×10^{-1}	8.134155×10^{-1}	8.134166×10^{-1}	8.134155×10^{-1}

*See note in Table 2.

FOOTNOTES

[1]United States Department of Defense, Reference Manual for the Ada Programming Language (July 1980, foreward).

[2]Ibid.

[3]Ibid.,pp. 1-2.

[4]Ibid.,pp. 1-2.

[5]Ibid., pp. 1-3.

[6]Clinton F. Goss, et. al., NYU Ada Project (Courant Institute, New York University, 1981), overview.

[7]Ibid.

[8]Ibid.

[9]Lyle R. Langdon, "Approximating Functions for Digital Computers," Indust. Math., 6 (1965), 79.

[10]James S. Vandergraft, Introduction to Numerical Computations (New York, 1978), p. 2.

[11]C. T. Fike, Computer Evaluation of Mathematical Functions (Englewood Cliff, 1968), pp. 2-4.

[12]Ibid., p. 5.

[13]Ibid., pp. 5-6.

[14]Ibid., p. 6

[15]Anthony Ralston, A First Course in Numerical Analysis (New York, 1965), pp. 4-5.

[16]John F. Hart, et. al., Computer Approximations (New York, 1968), pp. 7-8.

[17]Fike, op. cit., pp. 10-11.

BIBLIOGRAPHY

Abramowitz, M. and Stegun, I., eds. Handbook of Mathematical Functions with Formulas, Graphs, and Mathemaical Tables. New York: Academic Press, 1964.

Clenshaw, C.W. "Polynomial Approximations to Elementary Functions." MTAC, 5, 1954.

Duijvestijn, A. J. W. and Dekkers, A. J. "Chebyshev Approximations of Some Transcendental Functions for Use in Digital Computing." Phillips Research Reports, 16, 1961.

Fike, C. T. Computer Evaluation of Mathematical Functions. Englewood Cliff: Prentice-Hall, Inc., 1968.

Hart, John F.; et. al. Computer Approximations. New York: John Wiley and Sons, Inc., 1968.

Hastings, C.; et. al. Approximations for Digital Computers. Princeton: Princeton University Press, 1955.

Knuth, Donald E. The Art of Computer Programming, Vol. 2: Seminumerical Algorithms. Reading, Massachusetts: Addison-Wesley, 1969.

Kogbetliantz, E. G. "Computation of Sin N. Cos N and Mth Root of N Using An Electronic Computer." IBM Journal of Research and Development, 3, 1959.

Langdon, L. R. "Approximating Functions for Digitata Compu Computers." Indust. Math., 6, 1955.

Lyusternik, L.A.; Chervonekis, O.A.; and Yanpol'skii, A. R. Handbook for Computing Elementary Functions. Translated from Russian by G. J. Tee. Translation edited by K. L. Stewart. New York: Pergamon Press, 1965.

Ralston, Anthony. A First Course in Numerical Analysis. New York: McGraw-Hill, Inc., 1965.

Ralston, Anthony, ed. and Meek, C., asst. ed. Encyclopedia of Computer Science. New York: Petrocelli/Charter, 1976.