JOURNAL OF RESEARCH of the National Bureau of Standards—B. Mathematical Sciences
Vol. 77B, Nos. 3 & 4, July-December 1973

Performance Testing of a FORTRAN Library of Mathematical Function Routines—A Case Study in the Application of Testing Techniques

D. W. Lozier, L. C. Maximon, and W. L. Sadowski

Institute for Basic Standards, National Bureau of Standards, Washington, D.C. 20234

(April 12, 1973)

This paper describes an application of testing methodology and techniques developed by the authors to aid in improving the quality of mathematical software. These techniques differ radically in several aspects from techniques previously used, the most important difference being that the testing is not based exclusively on random arguments. Instead, throughout the range of each function use is made of special arguments that are designed to detect programming errors and to test the performance of an algorithm in different regions. The function values are tested against reference values which are stored on reference tapes generated by a highly authenticated system of subroutines.

Since the effectiveness of such a testing system in discovering errors and performance limitations can be fully ascertained only through actual use, we report the results of employing our system to test an existing FORTRAN library of mathematical function routines. Specific aspects of the numerical accuracy of the library used in this test case are discussed in order to illustrate the effectiveness of a well-designed testing system as an analytic tool for the evaluation of mathematical software.

Since documentation provides information necessary to perform testing and contains specifications that reflect the results of testing, our study includes comments on the documentation.

No information on timing or storage requirements is presented in this case study.

Key words: Automated testing; bit comparison; FORTRAN library; function validation; mathematical functions; performance tests.

1. Introduction

This paper presents the results of a case study of a FORTRAN library of mathematical function routines which serves to demonstrate the effectiveness of a testing program developed by the authors [1][1]. To achieve this purpose we include in the paper a general evaluation of the library and its documentation as well as data on its numerical accuracy. Programming errors discovered by the application of our techniques are pointed out and suggestions for improvement are made. The particular library which served as our test case is the UNIVAC 1108 FORTRAN library of mathematical functions. This library was released by UNIVAC for the EXEC 8 operating system in 1971; the RL 24 version reported on in this paper is a release that contains modifications enabling it to run under EXEC II.

This case study should not be construed as a validation or certification of the manufacturer's software and the authors would like to stress the fact that no endorsement of the manufacturer's library is implied.

AMS Subject Classification: 68A10.

[1] Figures in brackets indicate the literature references at the end of this paper.

2. Testing Techniques

Below is a short discussion of some of the salient features of our testing approach.

1. The library function values were tested by comparing them with reference values. The latter were obtained through the use of a software system containing an arbitrary precision arithmetic package and a set of arbitrary precision algorithms developed by us at the National Bureau of Standards [2]. This eliminates the necessity of recourse to the software on a computer with higher precision to supply the reference values, and permits the testing of software on a computer with any wordlength. To minimize the effect of the local computer environment on the performance of our software testing system, we store the reference function values on tape. The comparison between library function values and the reference values is performed by our Bit Comparison Program designed for highly automated testing [1].

2. The performance of the testing system has been authenticated by a series of rigorous tests that included checks with published tables [1].

3. To avoid conversion errors in the arguments, only exact machine (octal) arguments were used to test the mathematical function subroutines. For the same reason, function values are displayed in octal form. In this way all conversion errors are avoided.

4. Since no theory exists at the present time that would make it possible to construct a set of random arguments that are certain to be representative of the total population of allowable arguments, thus providing a statistical basis for testing, exclusive use of random arguments must currently be considered inadequate for function testing. Special arguments must be used to test certain aspects of performance. It is for this reason that we do not use statistical terms in evaluating the performance of the library. We test functions on a logarithmic scale by supplying arguments in each power of two throughout the entire argument range. Among these are both special and random arguments. Arguments with special bit configurations are designed to probe for weaknesses caused by specific features of the hardware. In view of the testing on a logarithmic scale, the percentages quoted in our breakdown of errors may differ significantly from those given by the manufacturer. Whereas the percentage of full bit accuracy in the various argument ranges is of importance to the tester and the mathematician who does research in algorithms, the user very frequently may not know the range in which his program will generate arguments as a part of a larger calculation. For such a user the most important index of the accuracy of a mathematical function subroutine is its maximum error and this is the index adopted in this study.

5. It is very important to have a clear definition of the error measure used in describing the results of any given performance test. If the definition given in the manufacturer's literature (UP–7876, pp. 2–3, 2–4) were applied to the results of their tests it would show the library in a worse light than it deserves. In fact, interpreting their performance results according to our definition of error measure gives a good agreement with our test results. Therefore, comparison between the manufacturer's test data and ours is made on the basis of our definition of the measure of error. We call our measure of error the "mantissa error." It is the difference resulting from a fixed point subtraction of the mantissa of the reference function value from the mantissa of the library function value. Prior to this subtraction the library function value is normalized to the characteristic of the reference value. The difference is expressed in units of the last bit position (ULP) [3], [4]. For example, an error of 3 ULP implies that the last two binary digits of the library value are in error but not the last three. The mantissa error is equal to the relative error to within a factor depending on the normalization of mantissas in the computer used (a factor of 2 in this case, where mantissas are normalized to lie between 1/2 and 1).

3. Comments on Manufacturer's Documentation

The documentation will be commented upon from two points of view, namely from that of the user and that of the tester [5]. The documentation contained in the FORTRAN V Library manual provides information deemed necessary for the user. It is well written and contains a wealth of information on each routine, including the test data. However, in a few instances the documentation con-

tains inaccuracies. For example, the statement $r^2+s^2<2^{128}$ on p. 2–51 (sec. on CLOG, subsection Argument and Function Range) should read $2^{-129} \leq r^2 + s^2 < 2^{127}$. On p. 2–52 the statement that error termination results if $r^2+s^2 > 2^{128}$ or $u<2^{-128}$ is incorrect. It should read $r^2+s^2 \geq 2^{127}$ or $r^2 + s^2 = 0$. No error termination results for the real part of the function less than 2^{-128}.

The task of testing would be facilitated by the the identification of mathematical methods and constants with appropriate references, such as Hart et al., Computer Approximations [8]. Furthermore, in listing the functions referenced by a given library routine, a listing of library routines which call the given function would aid in the cross-referencing and the tracing of the effects that errors in a given routine have on the performance of other routines. Modifications of the source code should be reflected in user documentation more promptly. For example, the September 1972 update of the user manual (UP–7876) does not contain mention of the crossover point for small arguments incorporated into the source code DSINCO at least since January 1971.

The source code and the comments contained therein constitute documentation for the tester. They are concerned with the computer implementation of the mathematical algorithms. Two practices, if adopted uniformly throughout the library, would save time and effort in testing and evaluation: The clear identification of all constants in octal—those used to compute function values as well as those used for logical decisions, examples being crossover points and points at which error tracing begins—and the identification at the appropriate point in the coding of the quantities being tested against these latter constants.

4. General Evaluation of the Library

The present library shows that attention has been paid to the choice of good algorithms based on numerical analysis. The computer implementation of these algorithms shows attention to the loss of significance, particularly in the process of argument reduction.[2] Extended precision coding has generally been supplied to avoid loss of accuracy, which requires special programming to circumvent the limitations of hardware (lack of guard digits, finite wordlength, floating point normalization, etc.) and of instruction repertoire (lack of double precision fixed point instruction for multiply and divide, etc.).

It is appropriate at this point to stress the importance of good testing techniques in developing a mathematical library of high quality. Those subroutines that have been tested with the techniques developed at the Jet Propulsion Laboratory [6], namely the single and double precision subroutines [7], exhibit painstaking attention to detail and have fewer programming errors than the complex function routines.

To minimize the effect of hardware limitations, some constants are "fine-tuned" by slightly modifying their bit representation. For example, in the DSINH routine, the octal machine constant 200140 . . . 00, representing unity on lines 93 and 94 of the source code, was changed to 200140 . . . 01 to decrease the number of two bit errors. This technique could be used to a still greater extent, however, and we occasionally make recommendations to that effect in the body of this report. The effect of these recommendations has been explicitly evaluated only for the routines DSINH and DSINCO.

The specifications for most routines in the library are very good. The general philosophy of accepting all arguments for which the function value can be held in the machine is adhered to for the most part. However, the specifications for certain complex functions (CLOG, CCBRT) are unduly restrictive in that the limit of the argument range is based on an overflow condition for $r^2 + s^2$ rather than $(r^2 + s^2)^{1/2}$.

We found the performance of the subroutines to be generally in conformity with the specifications, although, where the maximum error is concerned, the subroutines do not always come up to specifications. One particular example of this concerns the single precision functions, where the specifications give a maximum error of one ULP. This is indeed verified, with the single ex-

[2] This is in marked contrast to its immediate predecessor (March 23, 1966 update of the model 1107 FORTRAN Library Subroutines manual—UP–3947), which like several libraries of different origin contain certain weaknesses due to inadequate numerical analysis, such as the loss of all significance in the calculation of $\sinh x$ for small x by the use of $\sinh x = 1/2(e^x - e^{-x})$.

TABLE 1. *Performance of library functions*

Entry point	No. of arguments	Max error ULP	Percentage of values with ULP error of					Suggested improvements	Programming errors	Comments
			0	1	2	3	4			
			Percent	*Percent*	*Percent*	*Percent*	*Percent*			
SIN	2888	1	98	2				x		
DSIN	8140	2	34	63	3			x		
CSIN	2874							x	x	4–ULP errors due to COS routine.
real		1	97	3						
imag		4	76	20	3	ϵ	ϵ			
COS	2888	4	99	ϵ	ϵ		ϵ	x		Max. error exceeds specs.
DCOS	8140	2	83	12	5			x		
CCOS	2877							x	x	4–ULP errors due to COS routine.
real		4	97	1	ϵ	ϵ	ϵ			
imag		2	78	19		ϵ	4			
TAN	5693	1	95	5					x	
DTAN	8174	4	78	12	8	2	ϵ			
CTAN	2881								x	Error traceable to TANCOT routine.
real		1	96	4						
imag		1	99	1						
COTAN	5678	1	96	4					x	Fails strict monotonicity in range 2^{-26} to 2^{-14}.
DCOTAN	8159	5	57	37	5	1	ϵ		x	Max. error exceeds specs.
ASIN	1839	1	99.7	ϵ						
DASIN	9037	6	54	39	6	1	ϵ			Max. error exceeds specs.
ACOS	1839	1	99.5	ϵ						
DACOS	9037	4	74	23	3	ϵ	ϵ			Max. error exceeds specs.
ATAN	3870	1	96	4						
DATAN	11353	5	68	27	5	ϵ	ϵ			Max. error exceeds specs.
ATAN2(X_1, X_2)	699	1	98	2						The arguments were heavily weighted towards X_1/X_2 very small or very large to test the management function of the entry point.
DATAN2(X_1, X_2)									x	See section on Suggested Improvements, Errors and Specific Comments.
SINH	1384	1	99.7	ϵ						
DSINH	5906	3	8	91	ϵ	ϵ		x		
CSINH	2874							x	x	Max. error exceeds specs.
real		4	76	20	3	ϵ	ϵ			4–ULP errors due to COS routine.
imag		1	97	ϵ						
COSH	1384	1	99.6	ϵ						
DCOSH	5906	2	93	7	ϵ					
CCOSH	2877							x	x	Max. error exceeds specs.
real		4	98.6	1	ϵ	ϵ	ϵ			4–ULP errors due to COS routine.
imag		2	78	19	3					
TANH	1426	1	99	1						
DTANH	5906	1	99	1						

TABLE 1. *Performance of library functions*—Continued

Entry point	No. of arguments	Max error ULP	Percentage of values with ULP error of					Suggested improvements	Programming errors	Comments
			0	1	2	3	4			
			Percent	*Percent*	*Percent*	*Percent*	*Percent*			
CTANH	2881								x	Max. error exceeds specs.
real		1	99	1						4–ULP errors due to COS routine.
imag		1	96	4						
EXP	2810	1	99	ϵ					x	
DEXP	11812	2	97	3	ϵ					
CEXP	2875							x	x	Max. error exceeds specs.
real		4	98	1	ϵ	ϵ	ϵ			4–ULP errors due to COS routine.
imag		2	97	3	ϵ					
ALOG	1973	1	99.9	ϵ						
DLOG	14706	3	98	1.5	ϵ	ϵ				
CLOG	2101								x	Argument range unnecessarily restrictive.
real		1	85	ϵ						
imag		2	97	3	ϵ					
ALOG10	1973	1	99.3	ϵ						
DLOG10	14706	3	71	28	ϵ	ϵ				The disparity in 1 ULP errors between DLOG and DLOG10 is due to the double precision floating point multiply to convert from base e to base 10.
SQRT	1281	1	90	10						See section on Suggested Improvements, Errors and Specific Comments.
DSQRT	10242	0	100							
CSQRT	3715								x	
real		7+	84	6	ϵ	ϵ	ϵ			
imag		7+	84	5	ϵ	ϵ	ϵ			
CBRT	1537	1	99	1						
DCBRT	12290	2	69	31	ϵ				x	
CCBRT	2106								x	Max. error arbitrarily large. Argument range unnecessarily restrictive.
real			14	23	18	13	7			
imag			16	22	19	13	7			
CABS	3726	1	99.6	ϵ						

ception of the subroutine COS, where a misplaced crossover point results in a maximum error of 4 ULP in an extremely narrow range of the argument. In view of the narrowness of the range it is not surprising that tests based exclusively on random arguments missed this region.

In view of the fact that the operating system in almost every installation is modified to some extent, we would like to point out that such practices as branching on an overflow condition to return a special value of the function (such as is done in DATAN2) may lead to the printing of overflow messages and termination of execution, in a great many computer centers, whether or not such an error tracing capability is supplied by the manufacturer.

5. Tabulation of Results

In this section the results of our testing are presented in tabular form for quick reference. The functions are listed under the entry point name and are in the sequence in which they appear in the manufacturer's manual. A cross-reference table of entry points versus element names follows the tables. The comments on suggested improvements and errors contained in the section entitled Suggested Improvements, Errors and Specific Comments refer primarily to assembly listings and are therefore arranged by element name; the cross-reference table is given to facilitate reference to this later section. The suggested improvements describe for the most part modifications that can be made to the existing algorithms with very little effort and that often lead to substantial improvement in accuracy. They are included because they have come to light as a result of exhaustive testing based on our specialized testing techniques and may be of interest to anyone working on further refinement of these subroutines.

TABLE 2. *Cross-reference table of entry points versus element name*

Entry point	Element name	Entry point	Element name
SIN	SINCO	EXP	EXP
DSIN	DSINCO	DEXP	DEXP
CSIN	CSINCO	CEXP	CEXP
COS	SINCO	ALOG	ALOG
DCOS	DSINCO	DLOG	DLOG
CCOS	CSINCO	CLOG	CLOG
TAN	TANCOT	ALOG10	ALOG
DTAN	DTANCO	DLOG10	DLOG10
CTAN	CTNTNH	SQRT	SQRT
COTAN	TANCOT	DSQRT	DSQRT
DCOTAN	DTANCO	CSQRT	CSQRT
ASIN	ASINCO	CBRT	CBRT
DASIN	DASNCO	DCBRT	DCBRT
ACOS	ASINCO	CCBRT	CCBRT
DACOS	DASNCO	CABS	CABS
ATAN	ATAN		
DATAN	DATAN		
ATAN2(X_1, X_2)	ATAN		
DATAN2(X_1, X_2)	DATAN		
SINH	SINHCO		
DSINH	DSINH		
CSINH	CSINCO		
COSH	SINHCO		
DCOSH	DCOSH		
CCOSH	CSINCO		
TANH	TANH		
DTANH	DTANH		
CTANH	CTNTNH		

6. Suggested Improvements, Errors and Specific Comments

SINCO

Suggested improvements: Relocate crossover point from 2^{-12} to 2^{-13} (line 139 of assembly listing).

Comments: The large maximum error in COS is due to the misplaced crossover point mentioned above.

DSINCO

Suggested improvements:

1. The crossover point at 2^{-944} on line 210 of the assembly listing should be moved to 2^{-30}.
2. The constant on lines 224 and 225 should be changed from 2001400 . . . 0 to 2000777 . .7.
3. The constant on line 229 should be changed from 544067210334 to 544067210335.

The crossover point at 2^{-944} is set too low. This results in computing the function value instead of simply returning the argument for arguments between 2^{-944} and 2^{-30}, leading to increase in execution time and loss of accuracy. The suggested change of the crossover constant 121 to 1743 on line 210 of the assembly listing leads to the following statistics:

	DSIN	DCOS
0 ULP error	85.7 % (6976)	82.7 % (6730)
1 ULP error	12.2 % (995)	12.1 % (986)
2 ULP error	2.1 % (169)	5.2 % (424)

Fine tuning the constant 200140 . . . 0 to the value 200077 . . . 7 is to eliminate under certain conditions the renormalization in the calculation of the cosine. The polynomial representation used above the crossover point to calculate the cosine is of the form $1-(.308\ldots)x^2+\ldots$ [8]. When unity is represented by 200140 . . . 0, the result of subtracting the next term in the polynomial must be renormalized no matter how small x is. This brings in a zero in the last bit. Fine tuning avoids this problem for all x. (Note that the reduced argument is less than unity.) The statistics with this change, in addition to the first change, are:

	DSIN	DCOS
0 ULP error	87.6 % (7132)	87.7 % (7136)
1 ULP error	12.2 % (995)	12.2 % (995)
2 ULP error	0.2 % (13)	0.1 % (9)

Changing the constant 544067210334 to 544067210335 is a further step in fine tuning. Changing the constant that represents unity, to avoid renormalization, results in some negative one- and two-ULP errors that can be avoided by decreasing by one bit the magnitude of the coefficient of x^2 in the polynomial representation. The final statistics are as follows:

	DSIN	DCOS
0 ULP error	89.1 % (7254)	89.2 % (7258)
1 ULP error	10.7 % (874)	10.7 % (874)
2 ULP error	0.1 % (12)	0.1 % (8)

CSINCO

Suggested Improvements: Improvement suggested in SINCO will improve the performance of CSINCO.

Errors:

1. Whenever the real or the imaginary part of the function value is in the first bicade (characteristic 000), a zero is returned in place of the correct value for that part of the function. This is caused by testing only the characteristic when checking for underflow.

Comments: The 4–ULP errors come from the SINCO routine.

TANCOT

Errors:

1. The routine returns the incorrect sign for function values whose arguments are in the first seven bicades (characteristics 000 through 006) in TAN.

2. The routine fails to reject the argument 002400000000 for which the correct function value overflows in COTAN. An unnormalized result is returned.

Comments: Routine regularly fails to exhibit strict monotonicity in COTAN in the argument range 2^{-26} through 2^{-14}

DTANCO

Errors:

1. The routine fails to trace overflow values of the function for arguments $0 < x \leq 2^{-1023}$ in DCOTAN.

CTNTNH

Errors:

1. Wrong sign is returned in real part of answer in the first seven bicades (characteristics 000 through 006) in CTAN. The error is traceable to TANCOT routine.

2. Whenever the real or imaginary part of the function value is in the first bicade (characteristic 000), a zero is returned in CTANH in place of the proper value for that part of the function. This error is caused by testing only characteristics when checking for underflow.

DATAN

Errors:

1. The entry point DATAN2(X_1, X_2) has serious programming errors. Whenever $X_2 = 0$ (and $X_1 \neq 0$) the run is terminated due to an illegal operation instead of returning $\pm\frac{1}{2}\pi$.

DSINH

Suggested improvements: The function value should be set equal to the argument whenever the characteristic of the argument is less than or equal to 2^{-30} (characteristic 1743). This would decrease execution time and eliminate most of the 1–ULP errors in this range.

EXP

Errors:

1. The crossover constant to return zero (see lines 71 and 72 of the assembly listing) is placed incorrectly, neglecting to take account of the asymmetry in the storage of positive and negative characteristics. That is, the smallest normalized floating point number is 2^{-129} (octal 000400000000) and the largest is $2^{127}\ (1 - 2^{-27})$, (octal 377777777777).

CEXP

Suggested Improvements: Improvement suggested in SINCO will improve the performance of CEXP.

Errors:

1. Zero is returned for some non-zero function values.

Comments: The 4–ULP errors come from the SINCO routine.

CLOG

Errors:

1. Failure to check for underflow in converting $r^2 + s^2$ from double precision to single precision results in returning erroneous values for the real part of the function.

2. Poor handling of arguments $r^2 + s^2 = 1 + \epsilon$ with ϵ small, prior to passing them to ALOGC$, leads to an arbitrarily large loss of significance in the real part of the function value. These argu-

ments are represented by 34 bits, thus providing 7 guard bits instead of the 127 that are required by the method used here to provide enough significance to comply with the manufacturer's specifications after unity has been subtracted.

SQRT

Comments: The difference between our statistics, which show 90 percent full length accuracy, and those given in the FORTRAN V Library manual (UP−7876), which shows 100 percent full length accuracy, arises from certain of our nonrandom arguments. For example, for arguments of the form $2^{2N}(1 + 2^{-26})$ (where N is an arbitrary integer), i.e., for arguments having the mantissa 400000001, the square root of the argument, when written in the form corresponding to the normalization used, is $2^{N+1}(\frac{1}{2} + 2^{-28} - 2^{-56} + - \ . \ . \ .)$. In view of the term $- 2^{-56}$, the properly rounded mantissa of the square root is 400000000 rather than 400000001, as given by the library function. However, since our decision on how to round is based on the knowledge of the value in the 56th bit position, we do not consider that the expense of carrying the extra precision is worthwhile.

CSQRT

Errors:

1. For the arguments $(-0, 0)$ and $(0, -0)$ a result of large magnitude is returned with no indication of error.[3] The argument -0 exists because the computer used in the case study is a one's complement machine. The octal representation of $+0$ is 000000000000. The octal representation of -0 is 777777777777.

2. Erroneous results are returned for some arguments with a small characteristic. For example,

Argument		Reference value		Library value	
r:	200434545172	u:	200575625447	u:	200575625447
s:	000777777777	v:	000527235230	v:	377527235230
r:	−020400000001	u:	210400000000	u:	207552023632
s:	220400000001	v:	210400000000	v:	210552023632
r:	375777777777	u:	277552023631	u:	277552023651
s:	−100400000000	v:	−000552023632	v:	−377552023632

DCBRT

Errors:

1. Whenever the characteristic of the argument is 0000, the routine returns wrong results. This is due to failure to check characteristic for zero before subtracting 1 on line 70 of assembly listing.

This work was supported in part by the Institute for Computer Science and Technology of the National Bureau of Standards.

We want to express our deep appreciation for the contribution to this project by A. Liao who wrote several of the extended precision algorithms used in this work and designed tests for several of the functions. We would also like to express our appreciation to David Sookne for advice in some phases of this work.

[3] A similar result was obtained by Jet Propulsion Laboratory (C. L. Lawson, private communication, manufacturer notified).

7. References

[1] Lozier, D. W., Maximon, L. C., and Sadowski, W. L., A bit comparison program for algorithm testing, The Computer Journal **16,** No. 2, 111–117 (May 1973).

[2] Maximon, L. C., Fortran program for arbitrary precision arithmetic, unpublished data.

[3] Clark, N. A., Cody, W. J., Hillstrom, K. E., and Thieleker, E. A., Performance statistics of the Fortran IV (H) library for the IBM System/360, Argonne National Laboratory Report ANL–7321 (May 1967).

[4] Kahan, W., A survey of error analysis, Information Processing 71, Proceedings of the IFIP Congress 71, **2,** 1214–1239 (1971).

[5] Lozier, D. W., Maximon, L. C., and Sadowski, W. L., Documentation of mathematical function routines, presented at the joint SIAM–SIGNUM meeting in Austin, Texas, Oct. 16–18, 1972.

[6] Devine, Jr., C. J., and Lawson, C. L., Accuracy of single precision UNIVAC 1108 subroutine library functions, JPL Space Programs Summary 37–56, **II,** 115–121 (Jan.-Feb. 1969).

[7] Cox, M. W., UNIVAC library, SIGNUM Newsletter, **6,** No. 3, 9 (Nov. 1971).

[8] Hart, J. F., et al., Computer Approximations (John Wiley and Sons, Inc., New York, 1968).

(Paper 77B3&4–384)