A bit comparison program for algorithm testing

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

Institute for Basic Standards, National Bureau of Standards, Washington DC 20234, USA

In view of the increasingly important role of the computer in scientific calculations, the development of computer algorithms for elementary and special functions has been given a great deal of attention. The development of algorithms cannot be divorced from their evaluation, for a computer algorithm is judged solely on the basis of its performance characteristics. These include storage requirements, speed and accuracy. The present paper will deal only with the accuracy aspect of algorithm testing. The other two aspects must be evaluated in the context in which the algorithm is used. In this paper by an algorithm we mean a computer algorithm, i.e. an implementation of a mathematical algorithm in a specific environment. The environment is taken to include factors that may affect the algorithm, e.g. the operating system under which the program is run and hardware algorithms for arithmetic operations. Whereas in some instances mathematical algorithms have been successfully used to locate hardware malfunctions that were not traceable by normal trouble shooting tests, any malfunctions of the software or hardware will not be considered here to be part of the environment.

(Received November 1971)

1. Testing requirements

Before a test can be designed one must decide what the test is expected to accomplish. We have designed our tests to serve two functions. One is to determine the accuracy of the function values given by an algorithm; the other is to find and diagnose possible errors in the algorithm. The accuracy of the algorithm is tested by supplying argument values and comparing the function values given by the algorithm with reference values. While in principle our procedure does not differ from widely accepted practice (Kuki, 1971), it differs in certain specific aspects which will be discussed later.

1.1. *Decimal vs. machine arguments*

Whereas testing an algorithm with decimal arguments makes it possible to compare the decimal function values returned by it with tabulated values, thus telling the user how accurate the result is, this does not test the algorithm alone. Errors in the function values stem from three different sources, only one of which is traceable directly to the algorithm (Cody, 1969). These are:

1. Decimal-to-binary conversion of the argument and binary-to-decimal conversion of the computed function value via the machine software.
2. Errors in the function value arising from truncating the converted argument to a given computer word length.
3. Errors generated by the algorithm itself. (These might include, for example, either computer round-off errors or inadequacies of the mathematical algorithm for certain argument ranges.)

Since the error produced by the algorithm cannot be separated from the total error in testing with decimal arguments, we test the algorithm exclusively with exact machine arguments and we also display the function values in the machine form, thus avoiding all conversion errors.

1.2. *Machine independent vs. machine dependent testing*

One might think it would be desirable to generate one set of arguments with which an algorithm could be tested on any computer because such a procedure would afford a valid comparison of the algorithm's performance on different machines or because it would provide a guide for evaluating the performance of the machines themselves in the area of mathematical algorithms. However, the performance of a mathematical algorithm is affected by specific machine features (Cody, 1967), such as word length, number representation and hardware algorithms for arithmetic, and a test procedure should probe the influence of these machine dependent features on the performance. For example, a hexadecimal machine may have up to three leading zeros in the fractional part of a floating point number, thus reducing the number of significant bits in the word. A test procedure for a hexadecimal machine should, consequently, include some test arguments with leading zeros. On the other hand, on a binary machine these arguments are of no particular interest. Indeed, the most sensitive tool we have in testing computer algorithms is the possibility of choosing arguments with special bit configurations (i.e. bit patterns). These bit configurations may be chosen such that, although they may correspond to numbers which present no difficulty for the mathematical algorithm, they may be difficult for the computer algorithm. Consequently, in addition to a purely random set of bit configurations covering the range of the function within the limits of a given machine, we supply specific bit configurations that may be difficult for the computer algorithm to handle.

In addition to the specific choice of bit configuration dictated by the specific machine characteristics, there is another factor that makes machine independence impractical. Exact machine arguments should be given to the full number of bits on a given machine, i.e. if a machine has a certain word length then the low-order bits of the word should not be restricted to zeros. Such a restriction, while allowing the same set of arguments to be used on machines of different word length as long as the non-zero part of the word could be accommodated, would eliminate some capabilities of the testing program that we consider essential. First, it is through changes in the last few bits of the argument that one can most accurately check, for example, for the monotonicity of the function being tested and for the behaviour of the function in transition through zeros of the polynomials in terms of which it may be represented. Further, the presence of non-zero bits at the end of the word is a useful diagnostic tool in that any bits that are lost in a right shift through lack of guard digits or incorrect programming will be reflected in the function value returned by the algorithm.

We consider these points of such importance that we forego machine independent features in favour of a testing program that is written to take full advantage of the characteristics specific to the machine on which the algorithm being tested is implemented.

1.3. *Density of arguments*

From the previous discussion it is clear that while the numerical value of the argument is important for testing mathematical algorithms, it is the specific bit configuration in the close vicinity of the desired numerical value that provides important information on the performance of a computer algorithm. This implies, of course, that both random and non-random (special) arguments must be used to test an algorithm.

We believe that every bicade (power of 2) in the characteristic should be tested over the entire range of arguments on any given machine. The characteristic is a non-random component of the bit configuration. This is testing on a logarithmic scale.

The density of arguments is dictated by the following considerations. Since we believe that the majority of arguments supplied to an algorithm will range over a few bicades around unity, the density should be the highest there. In addition to this general consideration, specific functions should be tested with a large number of arguments in ranges where accuracy is difficult to achieve. Finally, a given program that uses several algorithms to span the argument range should be tested with a large number of arguments straddling the cross-over points. The logarithmic testing proposed here de-emphasises the statistical nature of random argument testing which by itself, as we have discussed, does not provide a complete evaluation of a computer algorithm.

The importance of displaying information cannot be overemphasised. A testing program should be set up in such a way as to allow for an easy scan for trends that an algorithm may exhibit. The arguments should be arranged in ascending order and the information printed out in an easily-readable form. For example, in testing a particular algorithm for the calculation of double precision SINH, it became apparent that the function value was consistently high by one bit for small arguments. Examination of the algorithm showed that a constant was entered with one bit in the 60th place whereas the true constant (Hart, 1968) has a bit in the 63rd position, i.e. the 60th bit should have been zero. Changing the constant resulted in correct function values to all bits for small arguments. This change, however, adversely affected the important argument range around 1/2 where it increased the number of 2 bit deviations significantly. Thus the 'wrong' constant was not an error but a design feature of the algorithm. This brings us to the last consideration we want to mention. An algorithm should return function values over the entire range that give the best fit and minimise the errors. In terms of the above example, many one-bit errors with a small number of two-bit

X	SINH (X) IN DECIMAL TO 60 AND 50 PLACES, AND THE DIFFERENCE	
2**(10)	+.26092 72717 18371 71005 60604 76684 90084 50586 99396 52155 47629 70533	(445)
	+.26092 72717 18371 71005 60604 76684 90084 50586 99396 52155	(445)
	0	
2**(9)	+.11422 06793 26987 83201 89375 75856 12017 11448 40509 39029 49054 12145	(223)
	+.11422 06793 26987 83201 89375 75856 12017 11448 40509 39029	(223)
	0	
2**(8)	+.75571 38325 02051 77126 00448 32853 64325 37531 20449 14356 03581 58175	(111)
	+.75571 38325 02051 77126 00448 32853 64325 37531 20449 14355	(111)
	1	
2**(7)	+.19438 54202 99729 75461 11336 84417 87390 36364 08753 15414 99442 88648	(56)
	+.19438 54202 99729 75461 11336 84417 87390 36364 08753 15414	(56)
	1	
2**(6)	+.31175 74540 40580 84414 54619 35446 42348 72415 69592 31178 99957 11410	(28)
	+.31175 74540 40580 84414 54619 35446 42348 72415 69592 31178	(28)
	1	
2**(5)	+.39481 48009 13403 47580 48901 13112 22029 33543 10096 96115 71324 95575	(14)
	+.39481 48009 13403 47580 48901 13112 22029 33543 10096 96116	(14)
	0	
2**(4)	+.44430 55260 25388 00507 94152 24083 34682 87811 82984 73325 00449 49167	(7)
	+.44430 55260 25388 00507 94152 24083 34682 87811 82984 73325	(7)
	0	
2**(3)	+.14904 78825 78955 01861 15876 63903 18814 46447 47431 41163 50990 94480	(4)
	+.14904 78825 78955 01861 15876 63903 18814 46447 47431 41164	(4)
	0	
2**(2)	+.27289 91719 71277 52448 90827 15907 93818 58028 94124 85530 29656 55284	(2)
	+.27289 91719 71277 52448 90827 15907 93818 58028 94124 85531	(2)
	1	
2**(1)	+.36268 60407 84701 87676 68213 98280 12617 04886 34201 23211 35721 30949	(1)
	+.36268 60407 84701 87676 68213 98280 12617 04886 34201 23212	(1)
	1	

Table 1 Reference function values for sinh (x) calculated to 60 and 50 decimal places. Numbers in parentheses on the right indicate powers of ten. Integers 0, 0, 1, etc., indicate deviation of the 50-digit number from the 60-digit number.

errors were deemed preferable to no one-bit errors at the expense of an increased number of two-bit errors.

2. Reference values and reference tape

The heart of a scheme to test algorithms is a package that permits the calculation of reference function values to a higher degree of precision than that of the test algorithm. The higher precision is usually obtained by computing reference values to a greater number of figures than are returned by the test algorithm. Since the reference algorithm, whatever its other characteristics, is assumed to return function values which have a greater number of correct significant figures than does the algorithm to be tested, it is important to make sure that this assumption is correct. Below we describe a hierarchy of tests which we have used to check this assumption.

The function values are computed in extended precision arithmetic, using Maximon's package (Maximon, 1971). Two of the features that make the package particularly valuable in algorithm work are that it will calculate to any desired number of figures and it will calculate in any desired number base. This makes it possible to use this package in the base of the particular computer and allows for the computation of all function values of exact machine representable arguments. These two features lead to the following hierarchy of tests:

1. The reference algorithm is run in the decimal mode with decimal arguments. Reference values are computed to two different numbers of figures. Usually 50 and 60 decimal figures are used. Comparison of the shorter with the longer number gives a check on truncation and roundoff errors. **Table 1** presents a sample of this kind of check, performed on the hyperbolic sine function. In no case does the shorter number differ from the longer number by more than nine units in the last place. The comparison is necessary to establish a bound on the truncation and roundoff errors across the entire range of definition of the function, but it does not establish the correctness of the function values.

x	Sinh (x) to 50 places
0·1	0·10016 67500 19844 02582 37293 83521 90502 35149 20916 87855
	0·10016 67500 19844 02582 37293 83521 90502 35149 20916 87855
0·2	0·20133 60025 41093 98762 55682 43010 31737 29744 94842 62574
	0·20133 60025 41093 98762 55682 43010 31737 29744 94842 62574
0·3	0·30452 02934 47142 61895 84352 67005 09522 90980 24232 68017
	0·30452 02934 47142 61895 84352 67005 09522 90980 24232 68018
0·4	0·41075 23258 02815 50854 02100 13844 69810 43531 50924 36328
	0·41075 23258 02815 50854 02100 13844 69810 43531 50924 36330
0·5	0·52109 53054 93747 36162 24256 26411 49155 91059 28982 61145
	0·52109 53054 93747 36162 24256 26411 49155 91059 28982 61148
0·6	0·63665 35821 48241 27112 34543 75465 14831 90249 63425 92786
	0·63665 35821 48241 27112 34543 75465 14831 90249 63425 92790
0·7	0·75858 37018 39533 50345 98746 47592 76815 41549 37614 21696
	0·75858 37018 39533 50345 98746 47592 76815 41549 37614 21702
0·8	0·88810 59821 87623 00657 47175 73189 75698 05597 09596 88808
	0·88810 59821 87623 00657 47175 73189 75698 05597 09596 88815
0·9	1·02651 67257 08175 27595 83361 61978 42235 37940 34465 51347
	1·02651 67257 08175 27595 83361 61978 42235 37940 34465 51348

Table 2 Sinh (x) calculated to 50 places by Maximon's extended precision routine working in the decimal mode (upper lines) and sinh (x) calculated by hand (lower lines) on the basis of 62 place values of exp (x) and exp ($-x$). The tabular values have been truncated to 50 places.

2. The correctness of the decimal values is established by comparison with tabulated values. See **Table 2,** in which decimal function values for the hyperbolic sine are compared with the hand calculated values of the same function, based on Van Orstrand's tables of the exponential function (Van Orstrand, 1921).
3. The same algorithm is then run in the octal mode with octal arguments and the truncation and roundoff errors are checked by comparing values with two different numbers of octal figures. **Table 3** shows this comparison where the number of octal figures corresponds to the number of decimal figures in 1.
4. Decimal function values obtained for exactly machine representable decimal arguments are compared with the function values for the same arguments in the octal mode. For example, if 50 or 60 decimal figures are used when running in the decimal mode, 55 and 65 figures are used when running in the octal mode. A conversion routine was used to express the values in the same number base. (See **Table 4**).

If the reference algorithm passes all of the above four tests it is considered to be correct and is used to generate reference values in the number base of the machine.

The reference function values, together with the exact machine representable arguments, are put on tape. The tape is then used as reference for the bit comparison package. The tape is a desirable feature for the following reasons. It allows a further degree of independence from the testing environment as it eliminates compilation of the arbitrary precision arithmetic routines, the arbitrary precision function routines and the actual calculation by these (compiled) routines of the function values. A tape also constitutes a convenient and relatively safe medium for the distribution of large numbers of reference values.

3. The driver program

The use of reference tapes separates the job of algorithm testing into two parts. The first is the selection of a set of arguments, generation of the reference function values and writing the tape. The second part consists of reading arguments from the tape, obtaining function values from the algorithm to be tested and comparing them with reference values read from tape. We call the program to accomplish this second part the bit comparison, or driver, program.

We have written a driver program to help us clarify our ideas on algorithm testing. It has been used to test the single and double precision exponential, hyperbolic and trigonometric functions in the NBS FORTRAN Library. We have also used it as a tool to help design Bessel function sub-routines. The driver program consists of two types of subroutines—those written in FORTRAN and those that deal with bit manipulation and hence lend themselves naturally to assembly language. Each of the assembly language subroutines consists of a few simple instructions that perform operations such as double precision fixed point add and subtract and left and right shifts. The driver program described below was used to test the Univac 1108 library and consequently contains assembly language subroutines pertaining to that computer. When the driver program is used on a different computer these assembly language subroutines, and certain other references to binary operation and octal display, must be modified appropriately. We have done this for the IBM System 360 series, a hexadecimal computer.

3.1. *Reference tape*

The reference tape is written in a form most convenient for use with the Univac 1108. The data is written in blocks of 50 arguments and 50 extended precision function values, together with

X	SINH (X) IN OCTAL TO 65 AND 55 PLACES, AND THE DIFFERENCE	
2**(10)	+.11770 13220 34724 32064 23103 41566 55320 63305 51010 03130 46233 25424 12343	(493)
	+.11770 13220 34724 32064 23103 41566 55320 63305 51010 03130 46232	(493)
	1	
2**(9)	+.62435 45011 35205 13466 01162 05317 40034 26463 11021 55155 03243 67234 21650	(246)
	+.62435 45011 35205 13466 01162 05317 40034 26463 11021 55155 03240	(246)
	4	
2**(8)	+.50161 72420 12276 56407 06214 60036 73575 51454 12444 33036 36511 25655 22554	(123)
	+.50161 72420 12276 56407 06214 60036 73575 51454 12444 33036 36510	(123)
	1	
2**(7)	+.14536 25142 73520 41357 72477 23251 03167 61071 21671 43334 55161 31067 74672	(62)
	+.14536 25142 73520 41357 72477 23251 03167 61071 21671 43334 55161	(62)
	0	
2**(6)	+.24113 14054 75313 71510 05634 36533 23555 40305 65300 52275 12737 06452 62715	(31)
	+.24113 14054 75313 71510 05634 36533 23555 40305 65300 52275 12737	(31)
	0	
2**(5)	+.10764 17746 11331 42617 54107 42040 01177 76230 46074 15534 33707 60026 00132	(16)
	+.10764 17746 11331 42617 54107 42040 01177 76230 46074 15534 33707	(16)
	1	
2**(4)	+.20745 65720 51777 74367 46531 11637 63550 02211 60376 52226 32767 74713 46037	(8)
	+.20745 65720 51777 74367 46531 11637 63550 02211 60376 52226 32770	(8)
	0	
2**(3)	+.27223 65121 23545 14564 06724 73210 53772 15477 40460 03542 73176 72272 37456	(4)
	+.27223 65121 23545 14564 06724 73210 53772 15477 40460 03542 73177	(4)
	0	
2**(2)	+.33224 34003 34033 06062 17716 01141 66227 12616 50420 32755 71066 60075 73340	(2)
	+.33224 34003 34033 06062 17716 01141 66227 12616 50420 32755 71067	(2)
	0	
2**(1)	+.35007 47543 55560 63733 63231 00054 45420 01134 16751 33561 25567 14775 55220	(1)
	+.35007 47543 55560 63733 63231 00054 45420 01134 16751 33561 25567	(1)
	0	

Table 3 Reference function values for sinh (x) calculated to 65 and 55 octal places. Numbers in parentheses on the right indicate powers of eight. Integers 1, 4, 1, etc. indicate deviation of the 55-digit number from the 65-digit number.

an identifying block number. The arguments are in double precision 1108 format and the extended precision values are in double precision 1108 format plus an additional 36 bit word of extended precision. Thus the block length is 251 words. This is an efficient blocking because the 1108 Exec II operating system writes output on tape in blocks of 256 words.

We hasten to add that, although the driver program has been written as an implementation of our philosophy of testing with a reference tape, it is not restricted to using a tape to store the reference table. Essentially, what we do is generate a table of reference values which conceptually is stored on magnetic tape in an appropriate format, but it may in fact be stored on any auxiliary storage device, such as disc or drum.

For single precision testing we simply use the corresponding double precision subroutines, when available, to generate the reference table in the format described above, assuming they have been previously checked out. The third word of extended precision is simply carried as zero. In this case we do not actually mount a tape that has been prepared in advance, but simply generate the reference table internally and store it on drum. This work is done in an initialisation subroutine, called INIT, where other critical parameters for the driver program are also stored.

3.2. *The initialisation subroutine*

We have designed the bit-comparison program to be useful in testing whatever function subroutines the user desires, providing he has either a reference tape supplied by us or a way of generating his own. Certain critical parameters concerning the tape are set up in the initialisation subroutine INIT. These parameters (all of which are of integer type) are passed via the labelled COMMON block

/INITBK/ITN, NPRE, NFN, NCODE(50), NLAST(50), NLOC(50)

ITN is the FORTRAN reference number of whatever auxiliary storage unit the reference table is stored on, be it tape, disc or drum. NPRE is the precision indicator; it is stored in Hollerith (left adjusted) as 'SNGL' for single-precision testing or 'DBLE' for double-precision testing. NFN is the number of functions that are stored on the reference tape, presently limited to 50. NCODE is a table of Hollerith codes of up to four characters which are set by the user to identify the functions on the reference tape. There must be one entry in the NCODE table for each function on the tape, i.e., NFN entries. For example, if the exponential and sine functions are on the tape, the user may wish to have the codes 'EXP' and 'SIN'. NLAST is a table giving the number of reference values stored for each function on the tape. Finally NLOC is a table giving the starting block number on tape of each function.

All these parameters except ITN will be specified for reference tapes supplied by us. But if the user writes his own reference

X	SINH (X) TO 50 DECIMAL AND 55 OCTAL PLACES, OCTAL CONVERTED TO DECIMAL, AND THE DIFFERENCE	
2**(10)	+.26092 72717 18371 71005 60604 76684 90084 50586 99396 52155	(445)
	+.26092 72717 18371 71005 60604 76684 90084 50586 99396 52150	(445)
	5	
2**(9)	+.11422 06793 26987 83201 89375 75856 12017 11448 40509 39029	(223)
	+.11422 06793 26987 83201 89375 75856 12017 11448 40509 39028	(223)
	1	
2**(8)	+.75571 38325 02051 77126 00448 32853 64325 37531 20449 14355	(111)
	+.75571 38325 02051 77126 00448 32853 64325 37531 20449 14352	(111)
	3	
2**(7)	+.19438 54202 99729 75461 11336 84417 87390 36364 08753 15414	(56)
	+.19438 54202 99729 75461 11336 84417 87390 36364 08753 15414	(56)
	0	
2**(6)	+.31175 74540 40580 84414 54619 35446 42348 72415 69592 31178	(28)
	+.31175 74540 40580 84414 54619 35446 42348 72415 69592 31178	(28)
	0	
2**(5)	+.39481 48009 13403 47580 48901 13112 22029 33543 10096 96116	(14)
	+.39481 48009 13403 47580 48901 13112 22029 33543 10096 96111	(14)
	5	
2**(4)	+.44430 55260 25388 00507 94152 24083 34682 87811 82984 73325.	(7)
	+.44430 55260 25388 00507 94152 24083 34682 87811 82984 73325	(7)
	0	
2**(3)	+.14904 78825 78955 01861 15876 63903 18814 46447 47431 41164	(4)
	+.14904 78825 78955 01861 15876 63903 18814 46447 47431 41163	(4)
	1	
2**(2)	+.27289 91719 71277 52448 90827 15907 93818 58028 94124 85531	(2)
	+.27289 91719 71277 52448 90827 15907 93818 58028 94124 85530	(2)
	1	
2**(1)	+.36268 60407 84701 87676 68213 98280 12617 04886 34201 23212	(1)
	+.36268 60407 84701 87676 68213 98280 12617 04886 34201 23211	(1)
	1	

Table 4 Reference function values for sinh (x) calculated to 50 decimal places (upper lines) and 55 octal places converted to 50 decimal places (lower lines). Numbers in parentheses indicate powers of ten. Integers 5, 1, 3, etc. indicate differences.

tape, he must supply his own parameters. Note that the reference table can actually be prepared as part of the INIT subroutine. This is the procedure we use for testing single-precision library functions against double-precision library values.

3.3. *Input and output*

The user controls the driver program by supplying input data on cards that cause the program to perform various functions. They start with two header cards. These supply a heading of 131 symbols (80 from the first card, 51 from the second) which is printed at the top of every page. Following these is a control card that is punched, starting in column 1, with one of the control words CHECK, LIST or TEST. The effect of these options is described below. Next a function card is punched, starting in column 1, with a valid function code. Valid function codes of from one to four alphanumeric characters are stored in the NCODE table of the previously described INIT subroutine. Following the function card is an argument card which must have ARG punched in the first three columns. It is used to specify the sequence numbers of the arguments one wishes to test, for example,

ARG N1 TO N2 IN STEPS OF N3

Here N1, N2 and N3 are read as free format decimal integers and need only be separated by either non-numeric characters or blanks. The program has default options in case N1, N2, or N3 are not supplied. Specifically, if N3 is missing, it is set equal to 1. If N2 is missing, it is set equal to the highest sequence number for that function. If N1 is missing, it is set equal to 1. Hence, if one wants all arguments on the tape only ARG need be punched. The user can supply as many ARG cards as he wants, one after another. In this way he can sample selected arguments or argument ranges. Following all the ARG cards there must be an END card, again punched in the first three columns. This denotes the end of that particular data set. Subsequent data sets having the same form may follow the END card, viz. Header, Control, Function, Argument and END. After the last data set one may supply a card punched ENDRUN (starting in column 1) which denotes the end of all data sets and causes the program to exit.

The control card CHECK causes the program to read selected data from the tape and compare it with stored data in the program for the purpose of verifying the tape reading and program operation. We include this program check so that a user who receives a deck and tape may assure himself that it runs properly on his computer. Since the CHECK subroutine verifies internal consistency of the tape and program, no function or ARG cards need follow the CHECK control card. It supplies its own sequence of arguments. The CHECK control card should be followed directly by an END card.

	ARGUMENT	STANDARD VALUE OF SINH	LIBRARY VALUE OF SINH	BIT DEVIATION
5220	177777777777 777777777777	2000412⁶3200 376307345545	200041263200 376307345523	000000000000 0000000000??
5221	200040000000 000000000000	2000412⁶3200 376307345545	200041263200 376307345523	000000000000 0000000000??
5222	200040000000 000000000001	2000412⁶3200 376307345546	200041263200 376307345546	000000000000 0000000000?0
5223	200040001763 171524647100	2000412⁶5364 636776727720	200041265364 636776727716	000000000000 0000000000??
5224	200040316010 221047455526	200041633714 277324016315	200041633714 277324016312	000000000000 0000000000??
5225	200040442044 355522340007	200041773010 200676335235	200041773010 200676335232	000000000000 0000000000??
5226	200040761546 212777731465	2000423463⁵4 144122262012	200042346354 144122262001	000000000000 0000000000??
5227	200041107167 067640477472	2000425⁰7521 214756357724	200042507521 214756357711	000000000000 0000000000??
5228	200041235145 647534046750	200042651360 654456163047	200042651360 654456163036	000000000000 0000000000??
5229	200041432405 761353152001	200043070212 265031171551	200043070212 265031171534	000000000000 0000000000??
5230	200041561430 611673322707	200043233466 725014404703	200043233466 725014404670	000000000000 0000000000??
5231	200041711015 262533706055	200043377446 753310613175	200043377446 753310613170	000000000000 0000000000??
5232	200042241067 660013241317	2000437⁶7044 161166515767	200043767044 161166515762	000000000000 0000000000??
5233	200042372077 710011650575	200044135204 441241404301	200044135204 441241404277	000000000000 0000000000??
5234	200042726003 430124221065	200044532210 345556441061	200044532210 345556441050	000000000000 0000000000??
5235	200043060454 720037263706	2000447⁰2572 127334341063	200044702572 127334341053	000000000000 0000000000?0
5236	200043213501 173412513636	200045053705 260045071524	200045053705 260045071512	000000000000 0000000000??
5237	200043420253 731145467530	200045305327 215634305271	200045305327 215634305254	000000000000 0000000000??
5238	200043554404 536257626041	2000454601⁷4 616167571130	200045460174 616167571110	000000000000 0000000000?0
5239	200043711114 432221140420	200045633605 754153176716	200045633605 754153176707	000000000000 0000000000??
5240	200044255770 271055624204	200046246006 060663557537	200046246006 060663557525	000000000000 0000000000??
5241	200044414202 126676766575	2000464⁰3761 375267200271	200046423761 375267200263	000000000000 0000000000??
5242	200044553000 413030172632	200046602516 310774666476	200046602516 310774666471	000000000000 0000000000??
5243	200044765071 740664216550	2000470⁴4224 616657255543	200047044224 616657255532	000000000000 0000000000??
5244	200045125024 263431307671	2000472⁰4610 051124066163	200047224610 051124066152	000000000000 0000000000??
5245	200045265347 466000316174	2000474⁰5770 060633623422	200047405770 060633623411	000000000000 0000000000??
5246	200045501774 230224236230	200047653254 535015306360	200047653254 535015306343	000000000000 0000000000?5
5247	200045643466 166753403175	200050036317 101205570230	200050036317 101205570220	000000000000 0000000000?0
5248	200046005555 305346734723	2000502⁰2171 757471247424	200050222171 757471247420	000000000000 0000000000?4
5249	200046370033 664056376261	200050661121 455631535751	200050661121 455631535743	000000000000 0000000000?6
5250	200046533706 123277461224	2000510⁴7540 477536741063	200051047540 477536741056	000000000000 0000000000?5
5251	200046700163 346264662551	2000512⁰7007 663124735647	200051237007 663124735631	000000000000 0000000000?6
5252	200047122372 122271427613	200051515240 200036654104	200051515240 200036654073	000000000000 0000000000?1
5253	200047270047 120433263065	2000517⁰6476 460407630146	200051706476 460407630134	000000000000 0000000000?2
5254	200047436133 535561042766	200052100601 761272453564	200052100601 761272453543	000000000000 0000000000?1
5255	200050032306 414666374476	200052557254 504467175705	200052557254 504467175701	000000000000 0000000000?4
5256	200050202222 127060002743	200052754271 524552362574	200052754271 524552362567	000000000000 0000000000?5
5257	200050602733 277521267471	2000534⁴2112 030456431016	200053442112 030456431005	000000000000 0000000000?1
5258	200050754516 230633420554	200053642120 147031370530	200053642120 147031370515	000000000000 0000000000?3
5259	200051126723 367026272302	2000540⁴3032 724174627312	200054043032 724174627301	000000000000 0000000000?1
5260	200051361635 620324640541	200054337275 024166224433	200054337275 024166224422	000000000000 0000000000?1
5261	200051535310 333614737460	200054542355 662520041336	200054542355 662520041313	000000000000 0000000000?3
5262	200051711412 121061057675	2000547⁴6361 767552652412	200054746361 767552652403	000000000000 0000000000?7
5263	200052324471 724225154645	200055455502 200762012157	200055455502 200762012147	000000000000 0000000000?0
5264	200052502511 003336340577	200055664664 130625125011	200055664664 130625125000	000000000000 0000000000?1
5265	200053122335 072724207403	2000564⁰4004 644661020407	200056404004 644661020373	000000000000 0000000000?4
5266	200053302312 536007403625	200056616433 163137173107	200056616433 163137173072	000000000000 0000000000?5
5267	200053462732 545341525430	2000570⁴2051 220624672331	200057032051 220624672304	000000000000 0000000000?5
5268	200053726755 325440322131	200057345765 120754445041	200057345765 120754445026	000000000000 0000000000?3
5269	200054110712 700636512623	200057563751 143001751347	200057563751 143001751337	000000000000 0000000000?0

Table 5 **Sample page of output from the bit comparison program showing comparison of NBS Univac 1108 FORTRAN library values of DSINH (X) with reference values for a selection of arguments in the range from 0·5 to 0·7. The output is in Univac 1108 machine format (octal). The bit deviation is in units of the 20th octal position of the 'standard' value.**

The control card LIST causes the program to read arguments and function values from the tape and print them according to the function and argument cards that follow it. The program prints the specified sequence numbers in decimal and the arguments and function values in octal. The arguments are displayed in double-precision machine format and the function values in double-precision machine format plus an additional word of extended precision. For single-precision function testing, where the reference table is merely double-precision, the double-precision arguments are zero past the single-precision length and the third word of extended-precision in the function value, being superfluous, is not printed.

The third control card, TEST, causes the program to perform the bit comparison between the reference values from the tape, properly rounded to 1108 double-precision format, with the function values from the algorithm being tested. The algorithm to be tested must be accessible in FORTRAN from a subroutine named TSTVAL(DX, DY). Here DX is the double-precision argument read from tape and DY is the double-precision test function value. Since the program is designed primarily to do bit-comparison testing, it being anticipated that listing or checking will be done only occasionally, the control card designating TEST may be omitted, in which case the program will automatically do the testing as indicated on the Function and ARG cards which directly follow the Header cards.

The testing part of the program takes arguments and function values from the tape in accordance with the Function and ARG cards. It then rounds each extended precision function value to double precision 1108 format. Next it obtains the double-precision value of the function from the test algorithm. The bit deviation of the test value from the reference value is now obtained by the following procedure. The test value is normalised to the reference value by right or left shifting the mantissa until the characteristics of the test value and the reference value are the same. The difference of the mantissas is then obtained by a fixed-point subtraction. This difference is the deviation of the approximate number from the true number expressed in units of the 60th bit position of the true number. It therefore is closely related to the relative error in the approximate number. For example, if the true number is $a.2^b$ and the approximate number is $\alpha.2^\beta = \tilde{a}.2^b$, where $\frac{1}{2} \leqq a < 1$, $\frac{1}{2} \leqq \alpha < 1$, then the bit deviation is $\tilde{a} - a$, which is equal to the relative error times a. Note that the bit deviation is not more than a factor of two different from the relative error. If the bit deviation is more than 60 places, only the difference of the characteristics is printed, since in this case the numbers are far apart. If the test value is more than 60 powers of two below the true value, the bit difference is the reference value, by definition.

The output format for the TEST option consists of printing the sequence number in decimal and the argument, reference value, test value and bit deviation in octal. The output is displayed in a highly readable form allowing for rapid visual

inspection for trends. (See **Table 5**). After each data set a page of statistics is printed, giving the number of test function values with bit deviations of 0, 1, . . . 7 units in the 60th bit position. (Thus for example, a three bit deviation in the 60th place would actually affect the 60th and 59th bit positions, since it has the binary representation 11.) The number of function values with more than 7 bits deviation from that data set is also printed. Finally, the number of function values tested is printed.

Acknowledgements

We would like to express our gratitude to the following people. To R. J. Arms, Computer Services Division, NBS, and I. A. Stegun, Applied Mathematics Division, NBS, for numerous helpful discussions in the course of the development of the program; to John Milazzo, Instruction and Research Support Group of the Computing Center at the State University of New York, Stony Brook and H. J. Oser, Applied Mathematics Division, NBS, for a critical reading of the manuscript resulting in valuable suggestions; to D. J. Sookne, for supplying certain assembly language subroutines; and to L. E. Sutton, Computer Services Division, NBS, for assistance in debugging parts of the program.

References

CODY, W. J. (1967). *The Influence of Machine Design on Numerical Algorithms*, Proc. Spring Joint Computer Conference, AFIPS Press, Montvale, N.J., pp. 305-309.

CODY, W. J. (1969). *Performance Testing of Function Subroutines*, Proc. Spring Joint Computer Conference, AFIPS Press, Montvale, N.J., pp. 759-763.

HART, J. F., *et al.* (1968). *Computer Approximations*, John Wiley and Sons, New York.

KUKI, H. (1971). *Mathematical Function Subprograms for Basic System Libraries-Objectives, Constraints and Trade-Off*, Mathematical Software, Academic Press, New York and London, pp. 187-199.

MAXIMON, L. C. (1971). *FORTRAN Program for Arbitrary Precision Arithmetic*, NBS Technical Report 10563, April 1, 1971.

VAN ORSTRAND, C. E. (1921). *Tables of the Exponential Function and of the Circular Sine and Cosine to Radian Arguments*, Memoirs of the National Academy of Sciences, 14, Fifth Memoir, U.S. Government Printing Office, Washington, D.C.

Book review

Computers and Automata, edited by Jerome Fox, 1971; 653 pages. (*Polytechnic Press of the Polytechnic Institute of Brooklyn*, £9·85)

This book contains 28 papers presented at a Microwave Research Institute Symposium held at the Polytechnic Institute of Brooklyn in April 1971. Two introductory papers have been added. The main aim of the symposium was to promote stronger links between practical computing and automata theory. As well as a table of contents the book includes the programme of the symposium, which divides the papers into the main areas of Programming Languages, Operating Systems, Computation Complexity, Logical Design and Computer Models. The titles of some papers in the contents are different to the corresponding ones in the programme and the latter lists a paper by Somalvico under Computation Complexity and a paper by Lazarev under Logical Design which have not been included.

It is with considerable pleasure that one picks up a book with the title *Computers and Automata*, because the relationships between the two have not been given nearly enough attention in the past. Computing is now complex enough to need a coherent way of stating and investigating the principles being used and it is unfortunate, as A. E. Laemmel says in his introduction, that 'Automata theory . . . has grown into an independent discipline whose connections with practical problems is at times quite tenuous'. The situation being what it is this book is inevitably something of a disappointment in that the problems have not been solved. Signs of promise however can be seen.

'Toward a mathematical semantics for computer languages' by Scott and Strachey is one of the most interesting papers in *Computers and Automata* to aim at a general theory. Ultimately the authors' approach should enable us to define the semantics of computer languages without ambiguity and without having to state how particular features should be implemented. The paper by Dennis, 'On the design and specification of a common base language', is an example of an alternative approach that regards the base language as a specification of the functional operation of a computer system and so as a suitable vehicle for the definition of computer languages. Many papers describe algorithms for the solution of computing problems, a few discuss computing systems based upon theoretical work.

In the section on Programming Languages, Lewis and Rosenkrantz describe a 2-pass ALGOL 60 compiler that uses a finite-state machine for lexical analysis and a deterministic pushdown machine for syntactical analysis. Other topics covered are machine language design, a theory to help in program design and fuzzy programs that, in difficult situations, 'do the best that they can'. The section on Operating Systems mainly consists of papers giving new or improved methods for solving system design problems. Computation Complexity covers a variety of subjects, from measures for computation complexity to problems in artificial intelligence. The section on Logical Design consists mainly of extensions to the theory that already exists. Computer Models includes a discussion by Zeigler on criteria for determining when one system simulates another and theoretical work on cellular computers that is claimed to be applicable to parallel computers.

Altogether, this is a valuable collection that should help us to focus our attention on an important area.

E. A. EDMONDS (Leicester)