AN ALTERNATIVE LIBRARY UNDER
4.2 BSD UNIX ON A VAX 11/780

W. J. Cody
Mathematics and Computer Science Division

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

February 1986
Contents

Abstract .................................................. 1
1. Introduction .......................................... 1
2. Test Results ......................................... 1
3. New Programs ........................................ 6
References ................................................ 8
Appendix .................................................. 9
An Alternative Library under
4.2 BSD UNIX on a VAX 11/780

W. J. Cody*

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439

Abstract

This paper describes an alternative library of elementary functions prepared for use with the standard Fortran compiler under 4.2 BSD UNIX on a VAX 11/780. The library, written in C and based on the book *Software Manual for the Elementary Functions* by Cody and Waite, offers improved accuracy over the standard system library, as well as additional capabilities. Listings and output from the ELEFUNT suite of test programs are included in the appendix.

1. Introduction

This work was motivated by attempts to locate the source of unexpected numerical errors in a program under development. A routine check of the double-precision library of elementary functions delivered with 4.2 BSD UNIX† disclosed that accuracy problems encountered in earlier releases persisted. Thus, it was decided to implement an alternative library using algorithms from the book *Software Manual for the Elementary Functions* by Cody and Waite [1980]. The results of that effort are reported here.

The next section summarizes and analyzes the results of running the ELEFUNT suite of test programs on the standard library, hereafter called the UNIX or system library, and the substitute library. Section 3 discusses the various new programs, and the appendix contains listings for them.

2. Test Results

ELEFUNT is the suite of transportable Fortran test programs from the *Software Manual for the Elementary Functions*. The suite includes MACHAR, a program to dynamically determine important machine constants. Table 1 contains the results from running single- and double-precision versions of MACHAR on the VAX 11/780. The values of most interest to us in interpreting later test results are BETA, the radix for the floating-point system; T, the number of base-BETA digits in a floating-point significand; and XMIN and XMAX, extreme values for positive floating-point numbers. These values show that the VAX double-precision arithmetic is binary and represents numbers between roughly $2^{-128}$ and $2^{127}$ to 56 bits, or about 16 decimal places. Other parameter values tell us that the arithmetic rounds, but that there are no "guard" bits to protect against loss of precision when a prealignment shift is necessary in addition/subtraction. Loosely speaking, this means that $1-\varepsilon$ may differ from $(.5-\varepsilon)+.5$ for small $\varepsilon$.

---

* This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.
† UNIX is a trademark of AT&T Bell Laboratories.
Table 1. Machine Parameters Determined by MACHAR

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Single Precision</th>
<th>Double Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>BETA</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>24</td>
<td>56</td>
</tr>
<tr>
<td>RND</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NGRD</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MACHEP</td>
<td>-24</td>
<td>-56</td>
</tr>
<tr>
<td>NEGEP</td>
<td>-24</td>
<td>-56</td>
</tr>
<tr>
<td>IEXP</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>MINEXP</td>
<td>-128</td>
<td>-128</td>
</tr>
<tr>
<td>MAXEXP</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>EPS</td>
<td>0.5960464477539e-07</td>
<td>0.1387778780781d-16</td>
</tr>
<tr>
<td>EPSNEG</td>
<td>0.5960464477539e-07</td>
<td>0.1387778780781d-16</td>
</tr>
<tr>
<td>XMIN</td>
<td>0.2938735877056e-38</td>
<td>0.2938735877056e-38</td>
</tr>
<tr>
<td>XMAX</td>
<td>0.1701411733193e+39</td>
<td>0.1701411834605d+39</td>
</tr>
</tbody>
</table>

Each of the ELEFUNT test programs exercises one or more of the elementary functions to estimate accuracy, check simple mathematical properties, and assess the response to improper or unusual arguments. The requirement that the test programs be portable has limited the approach to accuracy checking to determining how well the function program tested satisfies certain well-behaved identities. To be effective, such tests must be designed and implemented carefully (see the discussions in the Software Manual). A typical test from ELEFUNT samples 2000 random arguments uniformly distributed across an interval, and reports the maximum relative error (MRE) encountered and the root-mean-square relative error (RMS). To normalize results, these errors are reported in terms of the number of erroneous trailing base-BETA digits in the significand. Table 2 summarizes all accuracy tests run on the double-precision 4.2 BSD UNIX library (Spafford and Flaspohler [1985] report results for similar tests), and on the replacement programs we wrote. The tabulated summaries consist of the number of times out of a possible 2000 that the identities tested reported no error (the column headed 'Exact'), and the corresponding MRE and RMS values. No results are given for single-precision programs because they all return correctly rounded results from their double-precision counterparts.

Even a cursory examination of these results shows that many of the 4.2 BSD UNIX library programs have not been programmed as carefully as we expect of library programs. For example, while the first test of DSIN shows that the program is reasonably accurate when little or no argument
Table 2. ELEFUNT Test Results on a VAX 11/780

<table>
<thead>
<tr>
<th>Test</th>
<th>Interval</th>
<th>Library</th>
<th>Exact</th>
<th>MRE</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DASIN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{asin}(x) ) vs ( \text{Taylor Series} )</td>
<td>((-1/8, 1/8))</td>
<td>System 1108</td>
<td>1.95</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New 1988</td>
<td>0.59</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>(3/4, 1)</td>
<td>System 1312</td>
<td>2.68</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New 1519</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>DACOS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{acos}(x) ) vs ( \text{Taylor Series} )</td>
<td>((-1/8, 1/8))</td>
<td>System 1581</td>
<td>0.47</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New 1910</td>
<td>0.47</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>(3/4, 1)</td>
<td>System 725</td>
<td>10.69</td>
<td>5.27</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New 1975</td>
<td>0.95</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>(-1, -3/4)</td>
<td>System 1674</td>
<td>2.67</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New 1775</td>
<td>0.73</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>DATAN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{atan}(x) ) vs ( \text{Taylor Series} )</td>
<td>((-1/16, 1/16))</td>
<td>System 1324</td>
<td>1.19</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>( \text{atan}(x) ) vs ( \text{atan}(1/16)+\text{atan}\left(\frac{x-1/16}{1+x/16}\right) )</td>
<td>((1/16, 2-\sqrt{3}))</td>
<td>System 1138</td>
<td>1.98</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>( 2 \text{ atan}(x) ) vs ( \text{atan}\left(\frac{2x}{1-x^2}\right) )</td>
<td>((2-\sqrt{3}, \sqrt{2}-1))</td>
<td>System 790</td>
<td>2.19</td>
<td>0.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DEXP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \exp(x-1/16) ) vs ( \frac{\exp(x)}{\exp(1/16)} )</td>
<td>((1/16-\ln(2)/2, \ln(2)/2))</td>
<td>System 816</td>
<td>2.33</td>
<td>0.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New 1422</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>( \exp(x-45/16) ) vs ( \frac{\exp(x)}{\exp(45/16)} )</td>
<td>((-5 \ln(2), \ln(2^{256 \times \text{min}})))</td>
<td>System 21</td>
<td>5.97</td>
<td>4.35</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New 1361</td>
<td>1.55</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>((10 \ln(2), \ln(10 \times \text{max})))</td>
<td>System 2</td>
<td>5.98</td>
<td>5.01</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Table 2. ELEFUNT Test Results, Continued

<table>
<thead>
<tr>
<th>Test</th>
<th>Interval</th>
<th>Library</th>
<th>Exact</th>
<th>MRE</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLOG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ln(x)$ vs Taylor Series</td>
<td>$(1=0,1+\varepsilon)$</td>
<td>System 138</td>
<td>2.57</td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 1208</td>
<td>1.55</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$\ln(x)$ vs $\ln(17x/16) - \ln(17/16)$</td>
<td>$(1/\sqrt{2}, 15/16)$</td>
<td>System 135</td>
<td>7.26</td>
<td>5.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 1249</td>
<td>1.68</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$\ln(x \times x)$ vs $2 \ln(x)$</td>
<td>$(16, 240)$</td>
<td>System 497</td>
<td>4.82</td>
<td>2.73</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 1928</td>
<td>0.96</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>DLOG10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\log(x)$ vs $\log(11x/10) - \log(11/10)$</td>
<td>$(1/\sqrt{10}, .9)$</td>
<td>System 179</td>
<td>8.19</td>
<td>6.56</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 815</td>
<td>2.40</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td>POWER</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x$ vs $x^1$</td>
<td>$(1/2, 1)$</td>
<td>System 166</td>
<td>6.45</td>
<td>4.67</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 1607</td>
<td>0.88</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$(x \times x)^{1/3}$ vs $(x \times x) \times x$</td>
<td>$(1/2, 1)$</td>
<td>System 256</td>
<td>7.02</td>
<td>5.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 1355</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>$x^2$ vs $(x \times x)^{2/3}$</td>
<td>$(1, x_{\text{max}}^{1/3})$</td>
<td>System 9</td>
<td>8.43</td>
<td>6.72</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 1388</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>DSIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sin(x)$ vs $3\sin(x/3) - 4\sin(x/3)^3$</td>
<td>$(0, \pi/2)$</td>
<td>System 510</td>
<td>2.81</td>
<td>1.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 1241</td>
<td>1.37</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\cos(x)$ vs $4\cos(x/3)^3 - 3\cos(x/3)$</td>
<td>$(7\pi, 7.5\pi)$</td>
<td>System 201</td>
<td>14.01</td>
<td>9.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New 1192</td>
<td>1.92</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>
Table 2. ELEFUNT Test Results, Continued

<table>
<thead>
<tr>
<th>Test</th>
<th>Interval</th>
<th>Library</th>
<th>Exact</th>
<th>MRE</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSINH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sinh(x) vs Taylor Series</td>
<td>(0, 1/2)</td>
<td>System 1079</td>
<td>1.90</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>sinh(x) vs (\frac{\sinh(x+1)+\sinh(x-1)}{2\cosh(1)})</td>
<td>(3, ln(xmax)-1/2)</td>
<td>System 146</td>
<td>6.03</td>
<td>4.76</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>1.98</td>
<td>0.28</td>
<td></td>
</tr>
<tr>
<td>DCOSH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cosh(x) vs Taylor Series</td>
<td>(0, 1/2)</td>
<td>System 57</td>
<td>2.54</td>
<td>1.34</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>cosh(x) vs (\frac{\cosh(x+1)+\cosh(x-1)}{2\cosh(1)})</td>
<td>(3, ln(xmax)-1/2)</td>
<td>System 157</td>
<td>6.02</td>
<td>4.74</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>1.94</td>
<td>0.31</td>
<td></td>
</tr>
<tr>
<td>DSQRT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x vs (\sqrt{x})</td>
<td>(1/\sqrt{2}, 1)</td>
<td>System 1833</td>
<td>0.50</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1, \sqrt{2})</td>
<td>System 2000</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DTAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tan(x) vs (\frac{2\tan(x/2)}{1-\tan(x/2)^2})</td>
<td>(0, \pi/4)</td>
<td>System 683</td>
<td>2.44</td>
<td>0.69</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>2.50</td>
<td>0.65</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(7\pi/8, 9\pi/8)</td>
<td>System 778</td>
<td>2.56</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>2.00</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(6\pi, 6.25\pi)</td>
<td>System 663</td>
<td>2.55</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>2.23</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td>DCOTAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cot(x) vs (\frac{\cot(x/2)^2-1}{2\cot(x/2)})</td>
<td>(6\pi, 6.25\pi)</td>
<td>System --</td>
<td>--</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>1.92</td>
<td>0.29</td>
<td></td>
</tr>
</tbody>
</table>
Table 2. ELEFUNT Test Results, Continued

<table>
<thead>
<tr>
<th>Test Interval</th>
<th>Library</th>
<th>Exact</th>
<th>MRE</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tanh(x)) vs (\frac{\tanh(1/8) + \tanh(x-1/8)}{1 + \tanh(x+1/8)\tanh(1/8)})</td>
<td>(1/8, \ln[3]/2)</td>
<td>System</td>
<td>412</td>
<td>2.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>1072</td>
<td>1.91</td>
</tr>
<tr>
<td></td>
<td>(1/8+\ln[3]/2, 59 \ln[2]/2)</td>
<td>System</td>
<td>617</td>
<td>2.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New</td>
<td>827</td>
<td>1.65</td>
</tr>
</tbody>
</table>

reduction is required, the second test highlights a weakness in argument reduction by reporting a maximum detected loss of over 15 bits in evaluating the particular identity used. This is equivalent to a loss of about 5 decimal places of accuracy, out of 16 available, in the worst case. That such a loss is not intrinsic to the computation is evidenced by the corresponding test results for the replacement program, which loses about 1.5 bits, or half a decimal place, in the worst case.

Of the system library programs tested, only two were judged to be acceptably accurate and reliable: DATAN and DSQRT. Measured errors for DATAN were generally within the ranges we had observed in the past for good programs on binary machines. We therefore decided not to replace that program. Similarly, the measured accuracy of the DSQRT program was such that we could not reasonably expect to improve on it. For all other functions, at least one of the accuracy tests reported errors large enough to warrant preparation of a substitute program.

Some of the library programs are secondary routines; i.e., they rely on other function programs for part of their computation. In particular, the system library versions of DSINH and DCOSH use DEXP extensively, the power function calculates \(x^2\) as DEXP(Y*DLOG(X)), and DTANH uses DSINH and DCOSH. More accurate versions of DEXP and DLOG might, therefore, improve the accuracy of at least some of these secondary programs to the extent that they would be acceptable. Retesting with the replacement versions of DEXP and DLOG verified that that was not the case, however, and it was ultimately necessary to provide substitutes for all system library programs except DATAN and DSQRT.

3. New Programs

The replacement programs are written in C to enhance performance. Because they are among the first programs we have written in that language, no attempt has been made to be clever in programming. Tighter, more efficient code could be written by more experienced C programmers, if that were important. Nor has any attempt been made to modify the algorithms used; the programs are "vanilla" implementations of the algorithms in the book, even preserving the variable names in most cases.

The replacement library augments the system library with four programs not previously available:
1. ADX(X,N), a double-precision function routine that returns the double-precision floating-point argument X scaled by 2 raised to the Nth power.
2. INTXP(X), an integer-valued function routine that returns as a signed integer the floating-point exponent of the double-precision argument X.
Table 3. Default Response for $X^{**}Y$

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>System Routine</th>
<th>Replacement Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7.1</td>
<td>0.0</td>
<td>1.0</td>
<td>-1.7d+38</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-1.7d+38</td>
</tr>
<tr>
<td>-7.1</td>
<td>-7.1</td>
<td>0.0</td>
<td>-1.7d+38</td>
</tr>
<tr>
<td>0.0</td>
<td>-7.1</td>
<td>0.0</td>
<td>-1.7d+38</td>
</tr>
</tbody>
</table>

3. SETXP(X,N), a double-precision function routine that returns the double-precision argument $X$ with its exponent replaced by $N$.

4. DCOTAN(X), a double-precision function routine that returns the cotangent of the double-precision argument $X$, where $X$ is given in radians.

The first three of these routines are used internally by the new library, but are also accessible by direct call from Fortran. Although each of these four programs has been prepared in double precision, they work equally well in single precision.

Aside from the generally improved accuracy, the main differences between the behavior of replacement routines and the corresponding system routines are slightly modified domains and slightly different responses to bad arguments. For example, the system program DASIN returns 0.0, an unremarkable result, as the default response for arguments larger than 1.0 in magnitude, while the replacement routine returns the largest positive floating-point number. The philosophy in this case is to return a value so outrageous that the user is bound to notice it.

Similarly, the default responses for improper arguments to the power (exponentiation) routine differ, as shown in Table 3. The Fortran standard forbids negative-valued floating-point numbers raised to floating-point powers, and zero raised to a zero power. Clearly, zero cannot be raised to a negative power, either. Thus, the replacement routine supplies an outrageous value in these cases, in contrast with the innocuous values returned by the system library routine. On the other hand, the replacement program is deficient in supplying 0.0 as the result for $2.0^{-128}$, whereas the system program supplies a number close to the correct value, the smallest positive floating-point number.

The philosophy behind handling large arguments in the DSIN, DCOS, and DTAN programs differs between the system and replacement libraries. For sufficiently large arguments, argument reduction in these routines destroys all significance in the argument. The system routine takes no special action for large arguments, returning less and less significance in the computed function values until the only choice is the result 0.0. This occurs earlier for DTAN than for the other two programs. As a further indication of faulty argument reduction, the programs return 0.0 for certain non-zero arguments, even though integer multiples of $\pi/2$ are not exactly representable in the machine. The replacement routines, on the other hand, return 0.0 whenever the argument is so large that half of the significance is lost in argument reduction, i.e., whenever the argument exceeds about $8.4e8$ in magnitude, and do not return 0.0 results for any other non-zero arguments.

The domains for the replacement SINH and COSH programs have been extended over those for the system routines to include arguments for which the function values are representable even though
the value of DEXP is not. The default value returned for arguments that are too large is the largest floating-point number, as opposed to half of that number returned by the system routines.

The one remaining deficiency in the system library is the default value for ATAN2(0.0,0.0), which is π/2. A better choice, in our opinion, would have been either 0.0 or the largest floating-point number. We did not feel strongly enough about this to replace the ATAN2 program, however.

References


APPENDIX
/*
adx returns arg ** n computed by adding n to the floating-point exponent of arg (see the book Software Manual for the Elementary Functions by Cody and Waite.

translated from a Fortran program written by C. Moler.

*/

double adx(double arg, long int n)
{
    union u_tag {
        long int lx[1];
        double xx;
    } uval;
    uval.xx = arg;
    if (arg < 0.0e0) uval.xx = -uval.xx;
    /*
    the exponent field is in the following strange place
    ........ xx ........
    0000ff80 00000000
    ..lx(0). ..lx(1).
    
    shift n left 7 bits and add to high order word
    */
    uval.lx[0] = uval.lx[0] + 128*n;
    if (arg < 0.0e0) uval.xx = -uval.xx;
    if (arg == 0.0e0) uval.xx = 0.0e0;
    return (uval.xx);
}

adx_ provides access to adx from Fortran


*/

double adx_(double *x, long int *nn)
{
    double adx();
    return( adx(*x,*nn) );
}
/*
  asin(arg) and acos(arg) return the arcsin, arccos, respectively of their arguments.

  The program implements the algorithm from the book Software Manual for the Elementary Functions by Cody and Waite

  W. J. Cody, October 22, 1985
*/

#include <errno.h>
#include <math.h>
int errno;
double sqrt();
double asnwjc();

static double p1 = -27.368494524164255994;
static double p2 = 57.208227877891731407;
static double p3 = -39.688662997504877339;
static double p4 = 10.152522233806463645;
static double p5 = -0.6967457344735064641;
static double q0 = -164.21096714498560795;
static double q1 = 417.14430248260412556;
static double q2 = 381.86303361750149284;
static double q3 = 150.95270841030604719;
static double q4 = -23.823859153670238830;
static double eps = 1.0e-9;
static double pio2 = 1.57079632679489661923;
static double pio4 = 0.78539816339744830962;

double asin(double arg)
double arg;
{
  double asinjc();
  return (asinjc(arg,0));
}
double acos(double arg)
double arg;
{
  double asinjc();
  return (asinjc(arg,1));
}

static double asinjc(double arg, int flag)
double arg;
int flag;

{ double g, r, y, adx(), sqrt();

    int i;

    y = arg;
    if (y < 0.0)
        y = -y;
    if (y > .5) {
        i = 1-flag;
        if (y > 1.) {
            errno = EDOM;
            return(HUGE);
        }
        g = adx((0.5-y)+0.5,-1);
        y = -sqrt(g);
        y = y + y;
    } else {
        i = flag;
        g = 0.0;
        if (y > eps)
            g = y*y;
    }

    r = (((((p5*g+p4)*g+p3)*g+p2)*g+p1)*g; 
    r = r/((((g+q4)*g+q3)*g+q2)*g+q1)*g+q0); 
    r = y + y*r;
    if (flag == 0) {
        if (i == 1)
            r = (pio4+r)+pio4;
        if (arg < 0)
            r = -r;
    } else {
        if (arg < 0) {
            if (i == 1)
                r = (pio4+r)+pio4;
            else
                r = (pio2+r)+pio2;
        } else {
            if (i == 1)
                r = (pio4-r)+pio4;
            else
                r = -r;
        }
    }

    return (r);
}
/* 
exp returns the exponential function of its 
floating-point argument.

The program implements the algorithm from the 
book Software Manual for the Elementary Functions 
y by Cody and Waite

W. J. Cody, November 9, 1984 */

#include <errno.h>
#include <math.h>

int errno;
static double p0 = .24999999999999993e0;
static double p1 = .694360001511792852e-2;
static double p2 = .165203300268279130e-4;
static double q0 = .5;
static double q1 = .555538666969001188e-1;
static double q2 = .495862884905441291e-3;
static double bigx = 88.0296919;
static double smallx = -88.722;
static double eps = 6.0e-18;
static double log2e = 1.4426950408889634073599247;
static double c1 = .693359375;
static double c2 = -2.1219444005469058277e-4;

double exp(arg) double arg;
{ double x2,g,z,t1,t2,r,adx();
    int ent,en;
    if(abs(arg) < eps)
        return(1.);
    if(arg < smallx)
        return(0.);
    if(arg > bigx) {
        errno = ERANGE;
        return(HUGE);
    }
    ent = floor(arg*log2e+q0);
    en = arg;
    x2 = arg - en;
    g = ((en-ent*c1)+x2)-ent*c2;
    z = g*g;
    t1 = ((p2*z+p1)*z+p0)*g;
    t2 = (q2*z+q1)*z + q0;
    r = t1/(t2-t1)+q0;
    return (adx(r,ent+1));
}
//
// intxp returns as a signed integer the floating-point exponent of arg (see the book Software Manual for the Elementary Functions by Cody and Waite.


*/

static long int mask1 = 0xffff;

long int intxp(arg)
    double arg;
{
    union u_tag {
        long int lx[1];
        double xx;
    } uval;
    long int n;
    uval.xx = arg;
    if (arg < 0.0e0) uval.xx = -uval.xx;
    /*
    the exponent field is in the following strange place
    ........ xx ........
    0000ff80 00000000
    ..lx(0). ..lx(1).
    
    mask off bits above exponent field, shift right, and subtract bias.
    */
    uval.lx[0] = uval.lx[0] & mask1;
    n = uval.lx[0]/128-128;
    return (n);
}

long int intxp_(x)
    double *x;
{
    long int intxp();
    return( intxp(*x) );
}
Listing for log.c

The program implements the algorithm from the book Software Manual for the Elementary Functions by Cody and Waite

W. J. Cody, November 6, 1985

#include <errno.h>
#include <math.h>

int errno;

double frexp();

static double lge = 0.43429448190325182765e0;
static double half = 0.5e0;
static double c0 = 0.70710678118654752440e0;
static double c1 = 0.693359375e0;
static double c2 = -2.1219440054690582768e-4;
static double a0 = -0.64124943423745581147e2;
static double a1 = 0.16383943563021534222e2;
static double a2 = -0.78956112887491257267e0;
static double b0 = -0.76949932108494879777e3;
static double b1 = 0.31203222091924532844e3;
static double b2 = -0.35667977739034646171e2;

double log(arg)

double arg;
{
    double f, xn, r, w, z, zden, znum, setxp();
    long int n, intxp();

    if (arg <= 0.) {
        errno = EDOM;
        return (-HUGE);
    }
    n = intxp(arg);
    f = setxp(arg, 0);
    if (f > c0) {
        znum = (f - half) - half;
        zden = f*half + half;
    }
    else {
        n = n - 1;
        znum = f - half;
        zden = znum*half + half;
    }
    z = znum/zden;
    w = z*z;
    r = w*((a2*w + a1)*w + a0);
    r = r/(((w + b2)*w + b1)*w + b0);
Listing for log.c

    r = z+z*r;
    xn = (double) n;
    return ((xn*c2+r)+xn*c1);
}

double
log10(arg)
double arg;
{
    return(log(arg)*lge);
}
/ * computes a\(^b\).

The program implements the algorithm from the book Software Manual for the Elementary Functions by Cody and Waite

W. J. Cody, November 7, 1985

*/
#include <errno.h>
#include <math.h>
int errno;
static long int kk[34] = { 0X4080L, 0X0L ,
    0X257d4075L, 0X86cc1524L,
    0Xc0c6406aL, 0X2439e7ddL,
    0Xdce4060L, 0X94e1ec2aL,
    0X44f4057L, 0X9d6bcad6L,
    0X248c404eL, 0X8481151fL,
    0X672a4045L, 0X06db1155L,
    0X08a3403dL, 0X0c379f58L,
    0X04f34035L, 0Xde6433f9L,
    0X583e402dL, 0Xa14bea42L,
    0Xfed64025L, 0X5139a9b1L,
    0X532401eL, 0Xa126091L,
    0X37f04018L, 0Xb8a9518dL,
    0Xc3d34011L, 0X11c373abL,
    0X95c1400bL, 0X8bd7e3eaL,
    0Xaac34005L, 0X487b67ccL,
    0X4000L, 0X0L } ;
static double bigx = 127.01e0;
static double smallx = -126.0e0;
static double xk = 4.42695040888963407e-1;
static double a2[8] = { 0.241142095034203e-17,
    0.922915669372431e-18,
    -0.152419152311223e-17,
    -0.354218502822747e-17,
    -0.312862152451415e-17,
    0.293069995707897e-17,
    0.112608510409335e-17 } ;
static double p1 = 8.333333333412136e-2;
static double p2 = 1.24994796500608e-2;
static double p3 = 2.233824352815418e-3;
static double q1 = 6.931471805599378e-1;
static double q2 = 2.402265069567775e-1;
static double q3 = 5.550410842475687e-2;
static double q4 = 9.618117691387241e-3;
static double q5 = 1.333081011340821e-3;
static double q6 = 1.5c7740617881424e-4;

double pow(arg1, arg2)
  double arg1, arg2;
Listing for pow.c

```c
{  
    double adx(),*a1,g,r,redwjc(),setxp(),sz,temp,u1,u2,v;
    double w,w1,w2,x,y,y1,y2,z;
    long intxp(),irl,m1,na,np,np1,n1;

    if(argl < 0.) return(-HUGE);
    if(argl == 0.)
        if(arg2 <= 0.)
            return(-HUGE);
        else return(0.0);
    al = (double *) kk;

    /*
    determine m, g and np
    */
    m = intxp(arg1);
    g = setxp(arg1,0);
    np = 0;
    if (g <= *(a1+8))
        np = 8;
    if (g <= *(a1+np+4))
        np = np+4;
    if (g <= *(a1+np+2))
        np = np+2;
    na = (np+1)/2;

    /*
    Compute sz - 2s = (g-a)/(g/2 + a/2), subtracting a
    in 2 steps to preserve accuracy. Then compute
    logbase2((1+s)/(1-s)), and reduce result.
    */
    sz = (((g-a1[np+1])...}}}
/*
 * Check size of w1, then adjust w1 and w2 so w2 < 0.
 */
if (wl >= bigx) return (-HUGE);
if (wl <= smallx) return (0.0e0);
if (w2 > 0.0e0) {
    w2 = w2 - 0.0625e0;
    w1 = w1 + 0.0625e0;
}

/*
 * Determine p', m' and r'.
 */

n1 = (long) (wl*16.0e0);
irl = 1;
if (n1 < 0) irl = 0;
m1 = n1/16+irl;
np1 = 16*m1-n1;

/*
 * find 2**(-w2) as a polynomial in w2
 */

z = (((((q6*w2+q5)*w2+q4)*w2+q3)*w2+q2)*w2+q1)*w2;

/*
 * multiply by 2**(p'/16) and fix constant term 1.
 */

z = z*al[np1]+al[np1];

/*
 * add m' to biased exponent of result and exit
 */
return (adx(z,m1));

double redwjc(double arg)
{  long int i;
    i = (long) (arg*16.0e0);
    return(((double) i)*.0625e0);
}
Listing for setxp.c

/*
    setxp replaces the floating-point exponent of arg with the exponent n. See the book
    Software Manual for the Elementary Functions by Cody and Waite.

*/

double setxp(arg,n)
double arg;
long int n;
{
    double xx, adx();
    long int nn, intp();
    xx = arg;
    nn = n-intxp(xx);
    return (adx(xx,nn));
}

/*
    setxp_ provides access to setxp from Fortran

*/

double setxp_(x,nn)
double *x;
long int *nn;
{
    double setxp();
    return( setxp(*x,*nn) );
}
Listing for sin.c

#include <errno.h>
#include <math.h>

int errno;

static double r1 = -1.666666666666656052e-1;
static double r2 = 8.3333333333333331650314e-3;
static double r3 = -1.9841269841201840457e-4;
static double r4 = 2.7557319210152756119e-6;
static double r5 = -2.5052106798274584544e-8;
static double r6 = 1.6058936490371589114e-10;
static double r7 = -7.6429178068910467734e-13;
static double r8 = 2.7204790957888846175e-15;
static double eps = 1.0e-9;
static double pi02 = 1.5707963267948961923;
static double pi1 = 3.14159265358979323846;
static double pi2 = -8.5089102067615373566e-6;
static double wunopi = 0.31830988618379067154;
static double ymax = 8.433148e8;

double sin(double arg) {
    double sinwjc();
    double y, sgn;
    y = arg;
    sgn = 1.0;
    if (arg < 0) {
        y = -y;
        sgn = -sgn;
    }
    return (sinwjc(arg, y, sgn));
}

double cos(double arg) {
    double sinwjc();
    double y, sgn;
    return (sinwjc());
}
y = arg;
sgn = 1.0;
if (arg<0)
    y = -y;
y = y+pio2;

return (sinwjc(arg, y, sgn));

static double
sinwjc(arg, y, sgn)
double arg, y, sgn;
{
    double xn, f, g, r, d_abs();
    long int n;

    if (y > ymax) return (0);
    xn = y*wunopi+0.5;
    n = (long int) xn;
    xn = (double) n;
    if (n%2 != 0)
        sgn = - sgn;
    g = d_abs(&arg);
    if (g != y)
        xn = xn-0.5;
    f = (g - xn*pi) - xn*pi2;
    r = f;
    if (d_abs(&f) > eps) {
        g = f*f;
        r = (((((r8*g+r7)*g+r6)*g+r5)*g+r4)*g+r3)*g+r2)*g+r1;
        r = (r*g)*f+f;
    }
    r = sgn*r;

    return (r);
*\*  
\sinh(\text{arg}) returns the sinh of its argument.  

The program implements the algorithm from the book *Software Manual for the Elementary Functions* by Cody and Waite  

W. J. Cody, December 19, 1985  
*\*

```c
#include <errno.h>
#include <math.h>

int errno;

double exp();

static double ybar = 88.0296919e+0;
static double wmax = 88.0265308893e+0;
static double lnv = 0.69316101074218750000e+0;
static double vo2 = 0.52820835025874852469e-4;
static double eps = 3.725290298e-9;
static double p0 = -0.35181283430177117881e+6;
static double p1 = -0.11563521196851768270e+5;
static double p2 = -0.16375798202630751372e+3;
static double p3 = -0.78966127417357099479e+0;
static double q0 = -0.21108770058106271242e+7;
static double q1 = 0.36162723109421836460e+5;
static double q2 = -0.27773523119650701667e+3;

double sinh(double arg)
{
    double g, r, w, y, d_abs();
    y = d_abs(&arg);
    if (y < eps)
        return(arg);
    else if (y < 1.0) {
        g = y*y;
        r = (((p3*g+p2)*g+p1)*g+p0)*g)/(((g+q2)*g+q1)*g+q0);
        r = y+y*r;
    } else {
        if (y > ybar) {
            w = y-lnv;
            if (w > wmax) {
                errno = ERANGE;
                return(HUGE);
            }
            r = exp(w);
            r = r+r*vo2;
        } else {
            w = exp(y);
            r = (w - 1.0/w)*0.5;
        }
    }
}
```

Listing for sinh.c

if (arg < 0.0) r = -r;
return(r);
}

double d_sinh(x)
double *x;
{
    double sinh();
    return( sinh(*x) );
}

/*
cosh(arg) returns the cosh of its argument.

The program implements the algorithm from the book Software Manual for the Elementary Functions by Cody and Waite

W. J. Cody, December 19, 1985
*/

#include <errno.h>
#include <math.h>
int errno;
double exp();

static double ybar = 88.0296919e+0;
static double wmax = 88.0265308893e+0;
static double lnu = 0.69316101074218750000e+0;
static double vo2 = 0.52820835025874852469e-4;

double cosh(arg)
double arg;
{
    double r,w,y,d_abs();

    y = d_abs(&arg);
    if (y > ybar) {
        w = y-lnu;
        if (w > wmax) {
            errno = ERANGE;
            return(HUGE);
        }
        r = exp(w);
        r = r+r*vo2;
    } else {
        w = exp(y);
        r = (w + 1.0/w)*0.5;
    }
    return(r);
}
double d_cosh() {
double *x;
{
double cosh();
return( cosh(*x) );
}
Listing for tan.c

/*
  tan(arg) and cotan(arg) return the tan, cotan, respectively of their arguments.

  The program implements the algorithm from the book Software Manual for the Elementary Functions by Cody and Waite

  W. J. Cody, November 11, 1985
*/

#include <errno.h>
#include <math.h>

int errno;
static double p1 = -0.13338350006421960681e+0;
static double p2 = 0.34248878235890589960e-2;
static double p3 = -0.17861707342254426711e-4;
static double q1 = -0.4667168339755294240e+0;
static double q2 = 0.2566383228940112864e-1;
static double q3 = -0.31181531907010027307e-3;
static double q4 = 0.49819433993786512270e-6;
static double eps = 1.0e-9;
static double eps1 = 6.0e-39;
static double c1 = 1.57080078125000000000e0;
static double c2 = 0.63661977236758134308;
static double twoopi = 8.433148e8;
static double ymax = 1.0e-9;
static double s = 0.34248878235890589960e-2;
static double c = 1.57080078125000000000e0;

double tan(double arg)
double arg;
{
  double y, tanwjc(), d_abs();
  int iflag;

  y = d_abs(&arg);
  iflag = 0;

  return(tanwjc(arg, y, iflag));
}

double cotan(double arg)

double arg;
{
  double y, tanwjc(), d_abs();
  int iflag;

  y = d_abs(&arg);
  if (y < eps1) {
    errno = EDOM;
    if (arg >= 0)
      return (HUGE);
else return(-HUGE);
}
iflag = 1;
return(tanwjc(arg,y,iflag));
}

static double

tanwjc(arg,y,iflag)
double arg, y;
int iflag;
{

double xn,f,xnum,xden,g,d_abs();
long int n;

if (y > ymax) {
    errno = ERANGE;
    return (0);
}

xn = y*twoopi+0.5;
if (arg < 0.0) xn = -xn;
n = (long int) xn;
xn = (double) n;
f = (arg -xn*c1)-xn*c2;
xnum = f;
xden = 1.0;
if (d_abs(&f) > eps) {
    g = f*f;
    xnum = (((p3*g+p2)*g+p1)*g*f+f;
    xden = (((q4*g+q3)*g+q2)*g+q1)*g+0.5)+0.5;
}

if (iflag == 0) {
    if (2*(n/2) == n)
        return(xnum/xden);
    else
        return(xden/(-xnum));
}

else {
    if (2*(n/2) == n)
        return(xden/xnum);
    else
        return(-xnum/xden);
}

double d_tan(x)
double *x;
{
    double tan();
    return( tan(*x ) );
}

double r_tan(x)
float *x;
{
double tan();
    return( tan(*x) );
}
double d_cotan(x)
double *x;
{
    double cotan();
    return( cotan(*x) );
}
double r_cotan(x)
float *x;
{
    double cotan();
    return( cotan(*x) );
}
double d_cotan_(x)
double *x;
{
    double cotan();
    return( cotan(*x) );
}
double c_cotan_(x)
float *x;
{
    double cotan();
    return( cotan(*x) );
}
Listing for tanh.c

/*
 * tanh(arg) computes the hyperbolic tangent of its floating-point argument.

The program implements the algorithm from the book Software Manual for the Elementary Functions by Cody and Waite

W. J. Cody, December 10, 1985
*/

double exp();

static double xbig = 2.0101268236e+1;
static double c = 0.54930614433405484570e0;
static double eps = 3.725290298e-9;
static double p0 = -0.1613411902399628053e+4;
static double p1 = -0.99225929672236083313e+2;
static double p2 = -0.96437492777225469787e+0;
static double q0 = 0.48402357071988688686e+4;
static double q1 = 0.22337720718962312926e+4;
static double q2 = 0.11274474380534949335e+3;

double tanh(arg)
    double arg;
{
    double f, g, r, d_abs();

    f = d_abs(&arg);
    if (f > xbig)
        r = 1.0;
    else if (f > c) {
        r = 0.5*1.0/(exp(f+f)+1.0);
        r = r+r;
    } else if (f < eps)
        r = f;
    else {
        g = f*f;
        r = (((p2*g+p1)*g+p0)*g)/(((g+q2)*g+q1)*g+q0);
        r = f+f*r;
    }
    if (arg < 0) r = -r;
    return(r);
}

double dtanh(x)
    double *x;
{
    double tanh();
    return(tanh(*x));
}
Distribution for ANL-86-10

Internal:

W. J. Cody (40)
K. L. Kliewer
A. B. Krisciunas
P. C. Messina
G. W. Pieper (90)
D. M. Pool
J. M. Beumer (2)

ANL Patent Department
ANL Contract File
ANL Libraries
TIS Files (5)

External:

DOE-TIC, for distribution per UC-32 (168)
Manager, Chicago Operations Office, DOE
Mathematics and Computer Science Division Review Committee:
  J. L. Bona, U. Chicago
  T. L. Brown, U. of Illinois, Urbana
  P. Concus, LBL
  S. Gerhart, MCC, Austin, TX
  G. H. Golub, Stanford U.
  W. C. Lynch, Xerox Corp., Palo Alto
  J. A. Nohel, U. of Wisconsin, Madison
  D. Austin, ER-DOE
  J. Greenberg, ER-DOE
  G. Michael, LLL