INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE--ETC(U)
MAR 82  L J GALLAHER

UNCLASSIFIED

1 or 2
MAR 82
0 2 0
RADC-TR-82-46
F30602-78-C-0120 ML
INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE FOR DEVELOPING MACHINE INDEPENDENT SOFTWARE

Georgia Institute of Technology

L. J. Gallaher

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441
This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-82-46 has been reviewed and is approved for publication.

APPROVED: Elizabeth S. Kean
ELIZABETH S. KEAN
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER: John P. Huss
JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.
In this investigation of the ability of Ada to support machine independent software, a library package of the elementary mathematical functions (sin, cos, etc.) was implemented and tested on the Ada/ED Compiler Version 11.4. The Ada language constructs proved quite useful and effective in creating the math function package. The programs were written and successfully syntax checked; however, flaws in this version of the compiler prevented a thorough debugging of these routines.
The routines were designed to be machine and accuracy independent. Accuracy independence was obtained using variable length polynomials whose coefficients are computed (at compile time) from Chebyshev series. For increased efficiency, the normally machine dependent operations ("bit picking") are isolated into subroutines that can be optimized for individual installations and hardware.
ABSTRACT

In this investigation of the ability of Ada to support mathematically independent software, a library package of the elementary mathematical functions (sin, cos, etc.) was implemented and tested on the Ada/ED Compiler Version 11.x. The Ada language constructs proved quite useful and effective in creating the main function package. The programs were written and successfully syntax checked; however, flaws in this version of the compiler prevented a thorough debugging of these routines.

The routines themselves were designed to be both machine and accuracy independent; that is, the accuracy can be specified at the time the user integrates the package into their program. Accuracy implementations are obtained using variable length polynomials whose coefficients are computed (at compile time) from Chebyshev series. For increased efficiency the normally cumbersome dependent operations ("bit-pinning") are handled by subroutines that can be optimized for individual installations and hardware.

An appendix lists the library package.
TABLE OF CONTENTS

I. Introduction ........................................................................................................ 1

II. Theory .................................................................................................................. 2
   A. Range Reduction ......................................................................................... 2
   B. Chebychev Approximation ....................................................................... 3
   C. Other Approximations ............................................................................. 9

III. Ada Considerations .......................................................................................... 10

IV. The Routine ....................................................................................................... 11
   A. Introduction ................................................................................................ 11
      1. Variable Accuracy ............................................................................... 11
      2. Questions of Efficiency ........................................................................ 15
      3. Error Tolerances ................................................................................... 19
      4. Error Conditions and Out-of-Range Alarms ........................................ 23

   B. The Functions .............................................................................................. 24
      1. SIN and COS .......................................................................................... 24
      2. TAN ....................................................................................................... 24
      3. ATAN .................................................................................................... 26
      4. ACOS .................................................................................................... 26
      5. ACOSH ................................................................................................. 26
      6. EXP ...................................................................................................... 30
      7. LN .......................................................................................................... 30
      8. SQR ....................................................................................................... 31
      9. SINH and COSH .................................................................................... 31
     10. TANH .................................................................................................... 32
     11. ATANH .................................................................................................. 32
     12. ACOSH .................................................................................................. 32
     13. ATANH .................................................................................................. 32

V. Epilogue ............................................................................................................ 34
   A. What Went Wrong ...................................................................................... 34
   B. What Went Right ....................................................................................... 34
   C. Unsolved Problems ..................................................................................... 40

VI. Results and Conclusions ............................................................................... 64

VII. Recommendations for Further Study ......................................................... 64

VIII. List of References ......................................................................................... 64

Appendix A ............................................................................................................. 7

Appendix B ............................................................................................................. 9
LIST OF TABLES

Table II A-1 .................................................. 6
Table IV A 1-1 .................................................. 13
Table IV A 2-1 .................................................. 14
Table IV A 2-2 .................................................. 17
Table IV A 2-3 .................................................. 20
Table IV A 2-4 .................................................. 25
Table IV C-1 .................................................. 31
Table V A-1 .................................................. 36

LIST OF FIGURES

Figure IV A 4-1 .............................................. 50
Figure V B-1 .................................................. 61
Figure V B-2 .................................................. 62
1. Introduction

The goal of this project is to write a flexible and accurate independent library of elementary functions such as exp, log, etc. in C, and have new functions that should be reasonably close to being as efficient as could be written in assembly language and tailored to a particular machine.

In the past these elementary functions have been written in machine language and incorporated into the highest order language program by using adaptors of the language so far created to furnish and debug and were posted by the compiler. The compiler could find the accuracy of the library for each subject to output of sentences or similar to be available on in general, that a separate function to work in one high order language and have the adaptors written by a machine programmer, and were posted.

The coefficients of the language are then generated, the second accuracy determined the number of times this occurs. The coefficients of the language are then generated. The compiler could find the accuracy of the library for each subject to output of sentences or similar to be available on in general, that a separate function to work in one high order language and have the adaptors written by a machine programmer, and were posted.

For example, consider the language of the program.:

That is no problem if the monomials are only to be evaluated with one output at the time. But the problem is that the compiler could find the accuracy of the library for each subject to output of sentences or similar to be available on in general, that a separate function to work in one high order language and have the adaptors written by a machine programmer, and were posted.

For example, consider the language of the program.:

That is no problem if the monomials are only to be evaluated with one output at the time. But the problem is that the compiler could find the accuracy of the library for each subject to output of sentences or similar to be available on in general, that a separate function to work in one high order language and have the adaptors written by a machine programmer, and were posted.
for the elementary main functions; during the late fifteenth and early sixteenth
it held a unique position in almost every computing center library. As demand
for greater accuracy increased, improved approximations began to appear in
the literature until in 1930 "Approximations for Digital Computers" by Hart,
et al. superseded Hastings as the standard reference. This monumental work
brought together the best techniques then available and supplied tables of
coefficients for several thousand different approximations. The work was
more broad novelty on the work of Hart and company as the only references
indeed, rendered this expedient to exhibit an the philosophy of approach all of
the actual (Chebyshev) coefficients used in these programs were calculated
theoretically and not taken from Hart. This was more necessary by the fact that
the sum of the coefficients in a, or coefficients of the 

polynomials, led to the sum equal here as coefficients of the

Chebyshev series. The same series all of the approx-
mated are approximated to one of the same form given in Hart, as could
choose we and use the corresponding errors and values given there, and these are
 cited where they are expected to produce

Hart and Hastings give a large number of approximations for the various
elementary functions in the form $f(x) = f(x) + g(x)$, where $f$ and $g$ are
respectively. The approximations were used here in conjunction of the same form and
with the notation $h(x) = h(x) + k(x)$, that is, we have used single polynomial
approximations, not the pairs of the combined. There were several reasons
for this. First, it is not at all clear that the $h/k$ approximations are
significantly better (e.g., faster for the same accuracy) than the $f/g$,
spending single polynomial with the same number of total polynomial approx-
lations. Much depends on the status of the finite operation -- much more focus on
that in the floating point finite operation relative to the ordinary
operation. In 1958 it was assumed that the hardware speed ratio for multiply/divide was about 12:1. Today this ratio is more typically 1:2 to 1:4 with multiply gaining in speed relative to divide as newer models appear. In fact some of the array processors have no divide at all. Under these circumstances it seems highly questionable as to whether linear polynomial approximations are significantly faster than the simpler single polynomials.

Other reasons for restricting to a single polynomial is that it is much easier to find the near optimal polynomial for a given accuracy -- the Chebyshev approximation is this -- and the number of constants that need to be stored to them relatively small.

IV. RANGE REDUCTION

There are many ways to sacrifice the efficiency of approximating a function so that one of the three properties of Chebyshev is to reduce the range over which the approximation must be valid. One way is geometric, and the knowledge of symmetry and periodicity are quite effective. One way for example is the range of the function can be reduced to the interval $0 < x < 1$ and the product can be written as

$$ f(x) = f(1) g(1 - x) $$

and finally one can extend the range to $0 < x < 2$ by the relation

$$ f(x) = f(0) + f(1) + f(2) g(2 - x) $$

This then is one way of range reduction which is the basis of some of which the idea is illustrated from the Chebyshev approximation and the cosine from the sine approximation. Further reduction is then possible if $f(x)$ can be continued by making use of symmetry. However the symmetry is usually assuming
by the use of even or odd polynomials, i.e.,

$$\sin(x) = P_e(x^2)$$

$$\cos(x) = P_o(x^2)$$

where $P_e$ and $P_o$ are the sine and cosine polynomial approximations.

Table II.A-1 gives the ranges over which the various functions are approximated by polynomials. The various range reductions relations used are listed in the program. (See Appendix A.)

In doing range reduction for the functions EXP, LN and SQRT, it is assumed that the computer hardware in use is basically binary. This is consistent with both the proposed IEEE standards and HCF or Nubula specifications. For each of the functions LN and SQRT the argument is separated into a characteristic and an exponent of 2, i.e.,

$$x = j \cdot 2^p$$

where $j$ is integer and $0.5 \leq p \leq 0$.

Then for SQRT

$$\sqrt{x} = (\sqrt{2} \cdot \sqrt{j}) \cdot \sqrt{2^p} = a^{2^{p\prime}} \cdot b,$$

and $b$ is approximated by a polynomial.

For LN

$$\ln(x) = \ln(2^p) + \ln(j) = \ln b,$$

and $\ln b$ is approximated by a polynomial.

In the case of the EXP function the argument is first mapped onto a power of 2 by

$$x = a \cdot 2^{p\prime},$$

and $a$ is broken down into an integer and remainder part $k \cdot 2^{p\prime}$ where $k$ is integer and $0.0 \leq 2^{p\prime} \leq 2^k$ so that

$$a = 2^k \cdot 2^{p\prime} = 2^k \cdot 2^{p\prime}. \cdot 2^p$$

where $2^p$ is approximated by a polynomial and if $p \leq 0$ the reciprocal is
TABLE II.A-I

Ranges over which polynomial approximations are used for the various functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN</td>
<td>± π/4,</td>
</tr>
<tr>
<td>COS</td>
<td>± π/4</td>
</tr>
<tr>
<td>TAN</td>
<td>± π/8</td>
</tr>
<tr>
<td>ATAN</td>
<td>±(√2 - 1)</td>
</tr>
<tr>
<td>ASIN</td>
<td>±0.375</td>
</tr>
<tr>
<td>EXP</td>
<td>0, 0.5</td>
</tr>
<tr>
<td>LN</td>
<td>0.5, 1.0</td>
</tr>
<tr>
<td>SQRT</td>
<td>0.5, 1.0</td>
</tr>
<tr>
<td>SINH</td>
<td>±1.0</td>
</tr>
<tr>
<td>COSH</td>
<td>±1.0</td>
</tr>
<tr>
<td>TANH</td>
<td>±0.5</td>
</tr>
<tr>
<td>ASINH</td>
<td>±0.375</td>
</tr>
<tr>
<td>ATANH</td>
<td>±0.25</td>
</tr>
<tr>
<td>ACOS</td>
<td>polynomial not used</td>
</tr>
<tr>
<td>ACOSH</td>
<td>polynomial not used</td>
</tr>
</tbody>
</table>
taken. The operations dealing with the powers of 2 are assumed to be very fast; they will be machine dependent and should be tailored specifically for each machine.

The hyperbolic functions and their inverses are approximated by polynomials in a narrow region around the origin and outside this region are computed from their relationship to EXP, LN and SQRT.

The inverse functions arccos(x) and invcosh(x) have an anomalous behavior in the neighborhood of x = 1; they cannot be approximated by a polynomial in x near x = 1. Instead we have

\[ \text{arccos}(x) \approx \sqrt{2y + y^2/3 + 4y^3/45 + ...} \]

where \( y = 1-x \), and

\[ \text{invcosh}(x) \approx \sqrt{2y - y^2/3 + y^3/45 - ...} \]

where \( y = x-1 \).

The first two terms are used to approximate the functions, and the third term is used to control the range over which the approximation is used in such a way as to maintain the error tolerance. Outside this range the arccosine is computed from the arcsine: the invcosh is computed from LN and SQRT.

For the arcsine, an odd power series is used in the region about the origin; outside this region the arcsine is computed from its relation to arctangent and SQRT function, i.e.,

\[ \text{arcsine}(x) = \arctan(x/\sqrt{1-x^2}) \].

II. B. Chebyshev Approximations

The advantage of the Chebyshev polynomials is that they are the polynomials having the largest number of maxima and minima on the given interval and for which all maxima and minima are equal in magnitude. Thus, if the lowest order term neglected can be considered the error function, it can be seen that the error is more or less uniformly distributed over the interval,
and can be shown to have the smallest maximum error of any polynomial approximation of the same order.

Since accuracy requirements are known at compile time, it is possible in Ada to determine both the number of Chebyshev terms and the values of the coefficients to the power series at that point.

Example: \( \sin(wx) \).

This function can be mapped onto the interval \(-1 \leq x \leq 1\). The function can then be well approximated in this range by the \( n \)-term Chebyshev series

\[
\sin(wx) \approx \sum_{1 \leq k \leq n} B_k T_k(x) \quad -1 \leq x \leq 1
\]

where the Bs can be calculated from

\[
B_k = \frac{2}{\pi} \int_{-1}^{1} T_k(x) \sin(wx) dx / \sqrt{1-x^2}.
\]

(Only the odd numbered Bs will be nonzero.)

Each \( T_k(x) \) is a polynomial of order \( k \)

\[
T_k(x) = \sum_{0 \leq j \leq k} C_{kj} x^j.
\]

The Cs are well known and calculated exactly.
The $n$-term power series approximation for the sine then is

$$\sin(wx) = \sum_{1 \leq k \leq n} \sum_{0 \leq j \leq k} B_k C_{kj} x^j = \sum_{0 \leq j \leq n} A_j x^j$$

where

$$A_j = \sum_{j \leq k \leq n} B_k C_{kj}.$$

We see that adding more Chebyshev terms, say to improve the accuracy, changes all the $A$ coefficients of the power series. Similar considerations apply to approximations for the other functions.

II.C. Other Approximations

In addition to Chebyshev derived polynomials, two other approximations commonly used are Pade' like approximations and continued fraction. The Pade' like method consists of approximating as a function a ratio of two polynomials $P(x)/Q(x)$; it can be converted to an equivalent continue fraction and vice versa. While these methods can sometimes be superior to a simple linear Chebyshev polynomial approximation in the sense that the same or better accuracy can be obtained with fewer computer operations, they all suffer the same drawbacks from the point of view of trying to write machine and accuracy independent programs. This is that for optimum conditions, all coefficients or constants in the method will change if the accuracy is changed. The Pade' and continued fraction methods suffer other problems.

First, it is much more difficult to find the optimum set of coefficients for a given accuracy, while this is relatively easy to do for the Chebyshev series. The Chebyshev series requires storing only a single set of constants.
(10) for each function while for the P(x)/Q(x) method, an entire set of constants would be needed for each accuracy interval.

A second reason for not going with the ratio of polynomials is that it requires a division operation and the trend for hardware today is toward a relatively slow divide operation relative to addition and multiply. Present day hardware suggests we should avoid divisions where practical. (See IV.A.2 for a further discussion of this point.)

In summary, the Chebyshev method was chosen over the P(x)/Q(x), the ratio of two polynomials, because

1) Variable accuracy methods are relatively easier to obtain;
2) Fewer stored constants are required;
3) A division operation is eliminated (or traded for some number of multiplications and additions).

III. Ada Considerations

The principal Ada techniques expected to be most useful in developing machine and accuracy independent functions are the package and generic concepts. The main idea is that the functions will be embedded in a package with a generic parameter, call it REAL, describing the type (number of digits) to be used in the floating point arithmetic. The type REAL or its number of digits is then specified by the user in his main program. Then at the time the package and user program are integrated, the package is instantiated with the appropriate number of digits for REAL.

There are a number of program features that depend on the number of digits in REAL (called in Ada REAL'DIGITS).

First, the number of terms in the Chebyshev series and so the power series is determined by requiring that the Chebyshev terms not used are all smaller than $10^{-\text{(REAL'DIGITS)}}$. Next, this determines the size of the arrays that hold the power series polynomial coefficients. Then, in some cases
(ACOS, ACOSH, and TANH) the range in which a particular approximation is used will be determined by the number of digits in REAL; the computation of these ranges is carried out in the initialization block of the function packages.

Certain assumptions concerning the computer arithmetic have been made in writing these packages. We believe these assumptions are, in general, consistent with the Ada view, the IEEE standards, and the MCF or Nebula choice of floating point arithmetic. These assumptions are as follow:

1. There is a maximum floating point number (called in Ada FLOAT'LARGE).
2. -FLOAT'LARGE exists and is the most negative floating point number.
3. Every floating point number including FLOAT'SMALL, but excepting zero, has a reciprocal; the reciprocal of \(-\text{FLOAT'LARGE}\) may be zero, but need not be.
4. Any positive number can be subtracted from FLOAT'LARGE and any positive number can be added to -FLOAT'LARGE without causing an exception alarm.
5. FLOAT'LARGE can be divided by any number greater than or equal in magnitude to 1.0 or multiplied by any number less than or equal in magnitude to 1.0 without causing an exception.
6. Any two floating point numbers (including \(\pm\text{FLOAT'LARGE}\)) can be compared without causing an overflow or exception.

IV. Routines

IV.A. Introduction

IV.A.1. Variable Accuracy

One of the prime goals of this effort was to construct these subroutines
so as to be able to give variable accuracy. The object is to let the user specify the accuracy to which the computations are to be carried out with the idea that the less accuracy required the less time needed for the computation; the user does not need to pay (timewise) for accuracy not needed. The variable accuracy is achieved here by using variable length polynomials for the approximations. The number of terms in the polynomial is determined at compilation time (or later at "assembly" time) by use of a generic parameter type REAL. REAL is a user specified type and it is the number of digits specified by the user for the type REAL that determines the accuracy and, consequently, the number of terms in the polynomial approximation for each function.

Once the accuracy is determined the polynomial coefficients are computed from the corresponding Chebyshev coefficients. These Chebyshev coefficients have been precalculated and are stored as part of the program or package containing the function. Enough terms in the Chebyshev series are stored to be able to achieve an accuracy (relative error) of $10^{-16}$. It is assumed that this accuracy limitation ($10^{-16}$) will be adequate for the vast majority of the anticipated Ada applications in embedded systems; extension to higher accuracy would be straightforward. Table IV.A.1-1 gives a table of the accuracy versus the number of terms needed for each function. There is, of course, for each function a certain amount of computation necessary to reduce the argument range to within the range of the polynomial, but that is not reflected in this table. (Also not reflected in Table IV.A.1-1 is the fact that certain functions call other functions for some ranges of their values rather than using a polynomial.) Table IV.A.1-11 lists those functions that call others and in what ranges.
Table 1: A 1.1

The order of the polynomial measure for each function is dependent on accuracy required.

<table>
<thead>
<tr>
<th>Function</th>
<th>10/6</th>
<th>10/9</th>
<th>10/30</th>
<th>10/15</th>
<th>10/75</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN</td>
<td>0.9</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>COS</td>
<td>0.9</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>TAN</td>
<td>0.9</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>ATAN</td>
<td>0.9</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>ASIN</td>
<td>0.9</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>ACOS</td>
<td>0.9</td>
<td>2</td>
<td>6</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>ACOSH</td>
<td>polynomial not used</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ACOS: polynomial not used
<table>
<thead>
<tr>
<th>Function</th>
<th>Signature Call</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>FB (R, +)</td>
<td>R260 fi move</td>
<td>m m - 1.2</td>
</tr>
<tr>
<td>SCHD (R)</td>
<td>R260 fi move</td>
<td>m m - 4.1</td>
</tr>
<tr>
<td>C49H (R)</td>
<td>R260 fi move</td>
<td>m m - 2.4</td>
</tr>
<tr>
<td>189K (R)</td>
<td>R260 fi move</td>
<td>m m - 1.2</td>
</tr>
<tr>
<td>R26H (R)</td>
<td>R260 fi move</td>
<td>m m - 4.1</td>
</tr>
<tr>
<td>R66H (R)</td>
<td>R260 fi move</td>
<td>m m - 2.4</td>
</tr>
<tr>
<td>R16H (R)</td>
<td>R260 fi move</td>
<td>m m - 1.2</td>
</tr>
<tr>
<td>R66H (R)</td>
<td>R260 fi move</td>
<td>m m - 4.1</td>
</tr>
<tr>
<td>189K (R)</td>
<td>R260 fi move</td>
<td>m m - 2.4</td>
</tr>
</tbody>
</table>
5.1.2 Questions of efficiency

There are two types of efficiency: to be understood when existing processes
are considered, again and time calculations, one could even start to
put an effort to generate and scale up as little effort as possible. In a
sensible context, these two components, again and time, can be taken
into account. Figure 1 shows a comparison and in addition of the effort of
those tools in question, of which some requirement can be stated that
could be of great use. Of the comparison of the two graphs, which
is shown from the angle the tool was tested, the effort can be much
more complex. This figure above shows the impact of process
exhaustion in terms of effort and time, but when this is
understood. The effort and time above has been in content of

Figure 1: An important part of the time to come a result of
the existing processes. This figure shows the
time consumption of Python, the time consumption
of R, the time consumption of C, and the time
consumption of Java. The large difference in
time consumption can be attributed to the
existence of Python, which has lower times.

It is important to note that the effort and time of
these tools vary significantly. For Python, the
time consumption is significantly lower than for
R, C, and Java. However, the time consumption
of Java is significantly higher than for Python,
R, and C. This figure shows the importance of
understanding the time consumption of these
tools in order to make informed decisions in
the selection of tools for specific applications.

In conclusion, it is important to understand
the effort and time consumption of each of
these tools. This figure shows that Python has
demonstrated to be the most efficient tool in
terms of time consumption, followed by R and
C, and finally Java. The selection of the
tool depends on the specific requirements of
the application.
In calculating the average number of starts for the normal functions, it is to
be assumed that either the value of the largest number of the weight factor or
the function itself is equal to one. Since it is given that the normal for
the maximum number and the number of starts is the height of a weight
interpolated at zero.

<table>
<thead>
<tr>
<th>Function</th>
<th>Starts</th>
<th>Weight</th>
<th>Interpolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>225</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>226</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>236</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25:0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26:2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27:0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43:0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6:0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
example, Floating Point Systems' AP-120B, a very high speed auxiliary array processor, doesn't even have a divide; addition and multiplication have the same speed, giving ratios of 1:1:1 for its arithmetical speeds. Again this means that algorithms involving division will be at a significant disadvantage.

In any event, this suggests that one needs to make some kind of assumption concerning relative speeds of arithmetical operations in order to compare different algorithms. As a first approximation the speed ratios in Table 12 are used in comparing the efficiency of algorithms.

It is also clear from the observation of how slow divide is relative to multiplication that one should be looking for algorithms that use as few divides as possible. This is one of the justifications for avoiding the P(x)/Q(x), the ratio of two polynomials, algorithm and using the simpler Chebyshev polynomial instead.

4.4 Error Tolerance

One of the main goals here is to achieve an error tolerance consistent with the accuracy specified by the user. The user specifies the number of digits to be used for his particular type of floating point numbers or reals and then determines the acceptable error. Thus, for example, if the user specifies

```
.. afo and
```

the acceptable relative error tolerance will be taken as 10^{-5}. This implies then that the power series approximation will be composed of all Chebyshev terms whose absolute value is greater than or equal to 10^{-5}. This computation of the power series coefficients from the Chebyshev coefficients is done in the sequence initialization. It is also at this point that the size of the power series arrays is determined.
### Table 11.5.11

#### Normal Relative Speeds for the Arithmetic Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Relative Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add two real numbers</td>
<td>1</td>
</tr>
<tr>
<td>Multiply two reals or integers</td>
<td>1.2</td>
</tr>
<tr>
<td>Divide two reals or integers</td>
<td>2</td>
</tr>
<tr>
<td>Integer add or compare</td>
<td>4</td>
</tr>
<tr>
<td>Integer operations</td>
<td>4</td>
</tr>
<tr>
<td>Multiply, add, and divide by 1</td>
<td>4</td>
</tr>
<tr>
<td>Compare two real numbers</td>
<td>4</td>
</tr>
</tbody>
</table>
There is one exception to the above description of how the error is controlled and that is for the SQRT function. In the square root function a single Newton-Raphson or Heron's iteration is performed; i.e.,

\[ x = \frac{(y/x + x)}{2} \]

where \( x = \sqrt{y} \). This operation squares the relative error when the error is small. For SQRT it is only the first approximation that is obtained from the Chebyshev polynomials, the final value is obtained from the above iteration. Thus the accuracy of the power series need be only as good as the square root of the relative error tolerance; i.e., if the relative error tolerance is \( 10^{-5} \), the polynomial need only be as accurate as \( 10^{-2.5} = 3.16^{-5} \) -- the Heron iteration will improve this back to \( 10^{-5} \).

The Chebyshev approximations normally give absolute not relative error limits. When dealing with floating point arithmetic it is relative error that one wishes to maintain more or less uniform over some interval. For the even functions this is not a problem since in the interval of approximation the functions do not change greatly and so the relative error is very nearly equal to the absolute error (the functions being approximately 1 at the origin). To maintain the relative error more or less constant for the odd functions, say \( f(x) \), we use a Chebyshev approximation to \( f(x)/x \). This gives nearly uniform relative error for \( f(x) \) on the interval around \( x = 0 \), and again the absolute error is approximately equal to the relative since for these odd functions \( \lim_{x \to 0} (x + 1) f(x)/x = 1 \). So for example, the Chebyshev coefficients for the SIN function are actually those for \( \sin(x)/x \), and those for \( \tan(x)/x \), etc., for all the odd functions.
IV.A.4. Error Conditions and Out of Range Alarms

In nearly all computer arithmetic systems there is a limit to the size of a floating point number; there is also a smallest-in-magnitude nonzero number. These numbers are machine dependent but referable in Ada as F'LARGE and F'SMALL, where F refers to the (floating point) type. Because of these limits there are certain functions such as EXP, SINH, COSH, etc. for which only a relatively restricted range of inputs are valid. Thus for example, if we refer to our floating point type as REAL, input values to the EXP function greater than \( \ln(\text{REAL'LARGE}) \) cannot be computed since they would result in values greater than REAL'LARGE. There are other functions such as LN and SQRT that have invalid ranges because the result would not be in the real number system (but would be complex). For example, numbers less than or equal to zero are invalid inputs for LN; \( X < 0 \) is invalid for SQRT(X).

The problem of what to do about invalid inputs is complicated by the fact that all computer arithmetic with floating point numbers involves rounding; floating point arithmetic is seldom exact. When rounding takes place in the neighborhood of the boundary between the valid and invalid region of the function, it is difficult to decide what is the best action to take. Thus for example, if one wants to evaluate:

\[
\text{SQRT}(A + B + C - \text{SIN}(D))
\]

and the exact value of \( A + B + C - \text{SIN}(D) \) is 0 but due to rounding (or truncation in SIN) the computer results turns out to be \(-1.0 \times 10^{-9}\), what is the proper action? How should the SQRT subroutine respond to small negative inputs, and what is small? One action would be to trip a numeric exception alarm and let (force) the user handle the problem. Some systems halt the computation at this point.
Another solution is to assume that there is a small negative region such that input from that region ought really be treated as input of zero and not trip the exception alarm for these inputs, simply return zero; however it is difficult to determine how large or small this boundary interval should be.

The solution to the problem of the invalid input here is different yet from those mentioned above. We take the point of view that there should be no truly invalid input region and redefine our functions so all inputs are legitimate and will return some value and no exceptions are ever raised. For example, SQRT is defined so that

$$\text{SQRT}(X) = \begin{cases} \sqrt{X} & X \geq 0, \\ 0 & X < 0 \end{cases}$$

The square root function given here always returns a value, with no exception raised for input values from -REAL'LARGE to REAL'LARGE. Thus, small rounding errors in the vicinity of zero will cause no difficulties; however, it is the users responsibility to trap any input values to SQRT that the user feels ought to be considered illegitimate. Table IV.A.4-I gives the extended definition of the elementary functions showing how the usually considered invalid regions are treated. As an example, Figure IV.A.4-I shows how the arcsine function is extended beyond the normal input range of ± 1.0.

As mentioned above the main virtue of extending the definitions of the functions into their normally invalid regions is to avoid extraneous error signals when the argument drifts into the invalid region by a small amount of rounding or truncation error. This solution may not be the best resolution to the problem nor the solution desired by a particular user. It does however have the virtue of allowing the user to determine what he would like to see as
Figure IV.A.4-F
The extended definition of the arcsine function.
Table IV.A.4-I

A list of the functions and the way in which their inputs were extended to
the normally invalid regions, together with the output values in these regions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sin(x) )</td>
<td>( \sin(x) )</td>
<td>all ( x )</td>
</tr>
<tr>
<td>( \cos(x) )</td>
<td>( \cos(x) )</td>
<td>all ( x )</td>
</tr>
<tr>
<td>( \tan(x) )</td>
<td>( \tan(x) )</td>
<td>all ( x )</td>
</tr>
<tr>
<td>( \tan(x) )</td>
<td>returns ( \pm \text{REAL'\text{LARGE}} ) for ( x = \pm \frac{\pi}{2}, \pm \frac{3\pi}{2}, \pm \frac{5\pi}{2} \ldots )</td>
<td></td>
</tr>
<tr>
<td>( \arcsin(x) )</td>
<td>( \arcsin(x) ) (</td>
<td>x</td>
</tr>
<tr>
<td>( \arccos(x) )</td>
<td>( \arccos(x) ) (</td>
<td>x</td>
</tr>
<tr>
<td>( \arctan(x) )</td>
<td>( \arctan(x) )</td>
<td>all ( x )</td>
</tr>
<tr>
<td>( e^x )</td>
<td>( e^x ) (</td>
<td>x</td>
</tr>
<tr>
<td>( \ln(x) )</td>
<td>( \ln(x) ) ( x &gt; 0 ) ( -\text{REAL'\text{LARGE}} ) ( x &lt; 0 )</td>
<td></td>
</tr>
<tr>
<td>( \sqrt{x} )</td>
<td>( \sqrt{x} ) ( x &gt; 0 ) ( 0 ) ( x &lt; 0 )</td>
<td></td>
</tr>
<tr>
<td>( \sinh(x) )</td>
<td>( \sinh(x) ) (</td>
<td>x</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Domain</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>TANH(x)</td>
<td>$\tanh(x)$</td>
<td>all $x$</td>
</tr>
<tr>
<td>ASINH(x)</td>
<td>$\text{invsinh}(x)$</td>
<td>all $x$</td>
</tr>
</tbody>
</table>
| ACOSH(x)   | \[
|            | $\text{invcosh}(x)$               | $x > 1.0$|
|            | 0                                  | $x < 1.0$|
| ATANH(x)   | \[
|            | $\text{invtanh}(x)$               | $|x| < 1.0$|
|            | REAL'\text{LARGE}                  | $x > 1.0$|
|            | -REAL'\text{LARGE}                | $x < -1.0$|
the valid regions and allows (requires) him to set his own error traps for
what that user considers illegitimate inputs.

IV.B.1. **SIN and COS**

Input Value Range: -REAL’LARGE to REAL’LARGE (in radians)

Output Value Range: -1.0 to +1.0

The argument, \( x \), is first mapped onto the interval \(-\frac{\pi}{2}\) to \(\frac{\pi}{2}\).
If \( x \leq \frac{\pi}{4} \) the sine or cosine polynomial is computed for the respective
function; if \( |x| \) is greater than \( \frac{\pi}{4} \), SIN calls the cosine polynomial and COS
calls the sine polynomial with the appropriate sign and phase adjustment.

There are no invalid input or output ranges for SIN and COS. However
because of truncation and/or rounding errors it is possible for results to be
returned which are slightly larger in magnitude than 1.0.

IV.B.2. **TAN**

Input Value Range: -REAL’LARGE to REAL’LARGE (in radians)

Output Range: -REAL’LARGE to REAL’LARGE

The argument \( x \), is first mapped onto the interval \(-\frac{\pi}{8}\) to \(\frac{\pi}{8}\). If \( |x| \)
then is less than \( \frac{\pi}{8} \) the tangent polynomial is called; if \( |x| \) is in the
range \( \frac{\pi}{8} \) to \( 3\frac{\pi}{8} \) the tangent polynomial for \( |x| = \frac{\pi}{4} \) is computed (as Q)
and TAN is evaluated as \((Q+1)/(1-Q)\), with appropriate sign*. If \( |x| > 3 \frac{\pi}{8} \),
TAN is computed from the reciprocal of the tangent polynomial with appropriate
sign and phase adjustment.

The only invalid input arguments are \( |x| = m \frac{\pi}{2}, m=1, 3, 5, \ldots \) For
these angles the value ± REAL’LARGE is returned. No error halt or exceptions
are generated.

* This is an application of the relation:
\[
\tan x = \frac{\tan(x-b) + \tan b}{1 - \tan(x-b)}
\]
with \( b = \pm \frac{\pi}{4} \)
IV.B.3. ATAN

Input Value Range: -REAL*LARGE to REAL*LARGE

Output Range: -π/2 to π/2 in radians

If the argument, e. i. e. in the range \( 0 < \theta < 1.0 \), the arctan polynomial approximation is used. If \( \theta = 1.0 \), then arctan polynomial is used with \( 1/\theta \) as argument with appropriate sign and phase adjustment. In the intermediate range, that is \( \theta = 0 \), (sec \( \theta = 1.0 \)) the arctan polynomial is used with argument \( \theta = 1.0 \) (\( \sec \theta = 1.0 \)) and appropriate phase and sign adjustment.

There is no invalid input range.

IV.B.4. ASIN

Valid Input Value Range: -1.0 to +1.0

Output Range: -π/2 to π/2 in radians

If the argument, e. i. e. in the region \( \theta < 0.175 \) then the arcsine polynomial approximation is used; if \( \theta \) lies outside this region then the relation:

\[
\text{arcsine } \theta = \text{arctan}(\sqrt{1.0 - \theta^2})
\]

is used. Thus the arcsine routine uses both the arctan and the square root routine.

If the input is outside the valid region, that is outside -1.0 to 1.0, the value ±π/2 is returned depending on the sign of the argument. Note: no error halo nor exception is generated for input values outside the valid region -1.0 to 1.0 (see Section IV.A.4 above).
IV.3. **ACOS**

**Input** Input range: \(-\infty \leq x \leq 1\)

**Output** Range: \([-\pi / 2, \pi / 2]\)

If the argument, \(x\), is in the argument range \([-1, 1]\), then a special approximation is used as follows:

\[
0.5 \cdot \sqrt{4 + 4 \cdot \tan^2 \left(\frac{x}{2}\right)}
\]

where \(\theta\) is the angle between \(a\) and \(p + i q\). By using the final formula in this power series for small \(\theta\), the error in \(\theta\) is approximately half the third term. The relative error is then approximately \(\frac{2}{3} \cdot \frac{\cos \theta}{\theta}\) where \(\theta = \frac{x}{2}\) has been used. This relative error comes to be half below \(0.0005\pi\) when \(x = 0\) as we need

\[
\left|\frac{2}{3}\right| \leq 0.0005\pi
\]

The range of \(x\) then is roughly \(-1 \leq x \leq 0.0005\pi\) (more: \(x = \sqrt{\frac{3}{2}}\) and \(x = 0\) - \(\sqrt{\frac{3}{2}}\)).

Outside the above range the argument is calculated from \(\frac{\pi}{2} - \arg\) if the argument is outside the valid range, that is outside \([-\pi / 2, \pi / 2]\), the value 0 or \(\pi\) is returned depending on the sign of the argument. No error call nor exception is generated for input values outside the valid range (see Section IV.4.4 above).
The statement is true for all integers $n$ because the result of raising $n$ to an even power is a positive integer. If $n$ is an even integer, then $n^2$ is a positive integer. If $n$ is an odd integer, then $n^2$ is also a positive integer. Therefore, the statement holds true for all integers $n$.
If the argument $t$ to sine or cosine is larger than the value REAL-LARGE in the same unit as real, and is equal to or greater than
an argument not smaller than 0.0001

IV.B.4. SINH and COSH

Valid Input Value Range: -REAL-LARGE to -0.0001, REAL-LARGE

Output Range: +REAL-LARGE to REAL-LARGE

Also, as a result of the above statement, the value in SIGN is always positive.
Note that all inputs of $a$ are valid. In error cases or exceptions are

generated by $f(a)$. 

If $a$ is in the domain $A = \{ x \mid a \in A \}$, then $f(a)$ is computed from

$$f(a) = \frac{1}{a^2}.$$ 

If $a$ is in the domain $B = \{ x \mid a \in B \}$, then $f(a)$ is computed from

$$f(a) = \frac{1}{a^2}.$$ 

With appropriate sign of $f(a)$. 

Note that all inputs are valid and will return valid output. However, if $a$ is in the domain $C = \{ x \mid a \in C \}$ for which

$$f(a) = \frac{1}{a^2}$$ 

and the denominator will be quite large. Also, if $a$ is in the domain

$$f(a) = \frac{1}{a^2}.$$ 

If $a$ is in the domain $D = \{ x \mid a \in D \}$, then $f(a)$ is computed from

$$f(a) = \frac{1}{a^2}.$$ 

If the argument $a$ is in the neighborhood of 0, then a special 
approximation is used, as follows:
where $\theta$ is the inverse hyperbolic cosine and $y = x - 1.0$. By using the first two terms in this power series, for small $x$ the error in $\theta$ is approximately half the third term, the relative error is then approximately

$$\frac{\sqrt{y^2 - 1}}{\sqrt{y^2 + 1}}$$

where $\sqrt{y}$ has been used. This relative error needs to be less than 1000 * REAL_LARGE, so we need

$$\sqrt{y^2 - 1} \leq 1000 * \text{REAL_LARGE}$$

in the range of $y$ then $\theta$ is found by

$$\theta = \ln \left( y + \sqrt{y^2 + 1} \right)$$

With $\theta$ in the above range the approximation is divided into the regions:

For $\theta < \text{REAL_LARGE}/2$, the approximation $\ln(y + \sqrt{y^2 + 1})$ is used and above this limit $\text{ACOSH}$ is used for $\text{ACOSH}$

For real inputs, that is for $x < 0$, the value zero is returned, no error flag or exception is generated (see Section 13.4.4 above).

There will be values of $x$ is $\text{REAL_LARGE}/2$ for which the relation $\text{ACOSH}(x) > x$ will fail and the discrepancy will be quite large. Also the relation

$$\text{ACOSH}(x) = y$$

will fail badly for $y$ values such that

$$y > \text{REAL_LARGE}/2$$
IV.B.13 ATANH

Valid Input Value Range: -1.0 to 1.0

Output range: -REAL'LARGE to REAL'LARGE

If the argument, \( x \), is in the interval -0.25 to 0.25 a polynomial approximation is used for ATANH. In the interval 0.25 < \(|x|\) < 1.0 the relation

\[
\sinh^{-1}(x) = \ln\left(x + \sqrt{1 + x^2}\right)
\]

is used.

For the invalid input region \(|x| > 1.0\), the value \( \pm \) REAL'LARGE \( \pm \) returned and no error halt or exception is raised; this point is discussed in Section IV.A.4.

IV.C. Verification of the Function Package

Short of proving that each function given here is correct, the best that can be done is to check the correctness of the value computed for a large number of arguments for each function. Since at this point we have no way of obtaining check values internally in Ada, it appears that to check them, we must write them out and verify each value by hand or by another computer program in a different language such as FORTRAN.

There is, however, another good method of verifying the correctness of a large number of values of a function and that is by checking certain addition theorems for these functions. For example the sine and cosine should obey the relation

\[
\sin^2 x + \cos^2 x = 1
\]

for all \( x \). This, however, is not a very good check since the relative error of the smaller of these could be very large but not be observed. A better check are the "triple relations". For example

\[
\sin 3x = 3 \sin x - 4 \sin^3 x
\]

can be used to verify the sine function. The triple relations have the advantage of requiring just one of the functions at a time and allows it to be
checked against itself. Table IV.C-I gives a set of triple relations for the functions considered here. A very good verification procedure would be to check the triple relation of each function for several thousand (random) points uniformly distributed over a range that would ensure the exercising of all branches of the function code. Let us emphasize again that this has not been done yet due to lack of time and not having a suitable Ada compiler. Only the exponential function has been executed and partially verified; Figures V.B-I and II give triple relation error curves of EXP for a limited sample of arguments.

V. Critique

V.A. What Went Wrong

The most serious difficulty encountered by this project was the lack of an adequate Ada compiler. In fact not until 30 days before the final termination date of the project (60 days after the original termination date, a three month no cost extension was requested and granted), did we obtain the Ada/ED compiler. Ada/ED (Version 11.4) was obtained through NTIS and is considered a preliminary unvalidated version intended for education and experimental use only. And while it was extremely complete, it did have a variety of bugs or errors; those we uncovered are listed in Table V.A-I. These errors, while not serious once they were understood, contributed significantly to our confusion. We were, in fact, learning about Ada and the difficulties of untangling our mistakes and misapprehensions about Ada from errors in Version 11.4 proved a real strain and lead to the consumption of vast amounts of computer time. (Ada/ED is an experimental version and not designed for production use — it is very slow — by a factor of about $10^4$ to
The "Triple Relations" for Verifying the Accuracy of the Elementary Functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sin 3x )</td>
<td>( 3 \sin x - 4 \sin^3 x )</td>
</tr>
<tr>
<td>( \cos 3x )</td>
<td>( -3 \cos x + 4 \cos^3 x )</td>
</tr>
<tr>
<td>( \tan 3x )</td>
<td>( (3 \tan x - \tan^3 x)/(1 - 3 \tan^2 x) )</td>
</tr>
<tr>
<td>( e^{3x} )</td>
<td>( (e^x)^3 )</td>
</tr>
<tr>
<td>( \ln 3x )</td>
<td>( \ln 3 + \ln x ) or ( 3 \ln x )</td>
</tr>
<tr>
<td>( \sqrt[3]{x^3} )</td>
<td>( \sqrt[3]{3} \sqrt[3]{x} ) or ( \sqrt[3]{x^3} = (\sqrt[3]{x})^3 )</td>
</tr>
<tr>
<td>( \sinh 3x )</td>
<td>( 3 \sinh x + 4 \sinh^3 x )</td>
</tr>
<tr>
<td>( \cosh 3x )</td>
<td>( -3 \cosh x + 4 \cosh^3 x )</td>
</tr>
<tr>
<td>( \tanh 3x )</td>
<td>( (3 \tanh x + \tanh^3 x)/(1 + 3 \tanh^2 x) )</td>
</tr>
</tbody>
</table>

\[\begin{align*}
3 \sin^{-1} x &= \sin^{-1}(3x - 4x^3) \\
3 \cos^{-1} x &= \cos^{-1}(-3x + 4x^3) \\
3 \tan^{-1} x &= \tan^{-1}((3x - x^3)/(1 - 3x^2)) \\
3 \sinh^{-1} x &= \sinh^{-1}(3x + 4x^3) \\
3 \cosh^{-1} x &= \cosh^{-1}(-3x + 4x^3) \\
3 \tanh^{-1} x &= \tanh^{-1}((3x + x^3)/(1 + 3x^2))
\end{align*}\]
Table V.A-I

A list of known flaws (as of 10/26/81) in the Ada/ED Compiler Version 11.4

Compile Time:
1. Generic and actual parameters may not have the same name.
2. LONG_FLOAT not implemented.
3. Won't handle exponentiation of a universal constant.

Run Time:
1. Won't multiply by a floating point number if its value is 0.0.
2. INTEGER truncates instead of rounds (sometimes?).
3. Universal constants as parameters of functions or procedures cause a
   RUN time error halt, not a compile time error.
The net result of not obtaining a compiler until so late in the program and then to have it be somewhat flawed has been that the functions and packages here are not very well debugged. In fact we will have to include the disclaimer that "this package of functions is an unvalidated early version and is intended for experimental use only". We would like to be able to claim at this point that each of the functions has been thoroughly debugged and can be certified correct; but this is not at all the case. The best we can claim is that they have been syntax checked; except for the EXP function, none of the functions given here have been executed. Preliminary debugging of the EXP function has been accomplished and error curves obtained showing that EXP does meet the error tolerances requested; however, even here more checking should be done before it is released for incorporation into a "live" system.

V.B. What Went Right

The main bright note of this project is that most of the ideas about how to use Ada to build library packages seem to have checked out. To summarize these briefly:

1) Packages; the packaging feature of Ada works nicely and is just what is needed for writing library subroutines.

2) Generics; also work well and provide a mechanism for constructing variable accuracy routines where the error tolerance is effectively supplied by the user at the time the package is integrated into his total program. This has been demonstrated explicitly for the EXP function where the error curves show how the accuracy depends on the users specification of the number of digits in his REAL type variables; see Figures V.B-I and II.
3) Overloading; this was a useful and convenient device but was not really crucial to setting up the library routines. The main advantage of overloading was that it allowed mixed mode arithmetic (most of the arithmetic operators were overloaded for various combinations of INTEGER, REAL, FLOAT and LONG_FLOAT).

4) Function and procedures; these constructs also worked well and of course were indispensable in setting up an elementary math function library.

5) Dynamic Arrays; worked well and were especially useful for library routines in allocating exactly the needed storage (for the polynomial coefficients for example).

6) Isolation of machine dependent operation; most of the operations expected to be machine dependent were isolated into small, short functions. These are mainly the bit-picking operations and are to be tailored by each installation to their particular hardware. By using the pragma INLINE, these operations can be made run time efficient. However, since the INLINE pragma is not implemented in the Ada/ED version 11.4, it was not possible to demonstrate explicitly here that this works well. We were able to demonstrate that this is not a difficult or unnatural way to proceed.

As mentioned earlier we were able to demonstrate that the elementary function package is syntactically correct and have been able to execute a reduced package containing just the EXP function. Figures V.B-I and II show error curves for the EXP function. (These error curves were generated by comparing \( e^{3x} \) with \((e^x)^3\) for 50 values of \( x \). Note that \( \Delta = e^{3x}/(e^x)^3 - 1 \) can be up to four times the error tolerance since cubing \( e^x \) will triple the relative error and the error in \( e^{3x} \) can be in the opposite direction.) It would be desirable to run several more such curves at different (higher) accuracy requests before certifying the correctness of the EXP
Figure V.8-I
An error curve for the EXP function.

The measure of the relative error, \( \delta = \frac{\text{EXP}(x)}{\text{EXP}(x)^{1/3}} - 1 \),
is plotted here vs. X. REALDIGITS = 5 here giving an error
tolerance of \( 10^{-5} \).
An error curve for the EXP function.

The measure of the relative error, $\delta = \exp(32)/\exp(50) - 1$, is plotted here vs $x$. REAL*8 DIGITS = 6 here giving an error tolerance of $10^{-6}$.
function; and of course similar tests must be run on the rest of the library functions before they can be used with any confidence.

V.C. Unresolved Problems

There is one major unsolved problem associated with the particular method of constructing function packages as given here. This has to do with memory space and the fact that Ada does not seem to have anything similar to the OVERLAY feature of FORTRAN. In setting up the elementary functions SIN, COS, EXP, etc., a set of Chebyshev constants are introduced for each function; these are used to calculate the coefficients of the power series or polynomial approximation. These new constants need to be retained for the remainder of the execution of the program, but the Chebyshev constants need not be retained; the space allocated to the Chebyshev coefficients could be abandoned or reused for something else, except that Ada has no mechanism for doing this. Also there are program segments that are needed only at initialization time; these too could be abandoned once initialization has been completed and the space reused, but again there is no mechanism for doing this.

In a paging environment on a large computer such as the MAC, this is no problem -- segments that are no longer needed are eventually paged out and never brought back into main memory again. For the anticipated Ada applications however, say for executing an Ada program on a microprocessor with only 4K bytes, space is crucial and it is imperative that no longer needed data or program areas in memory be reusable. We do not know yet how in Ada to reuse either data or program memory area no longer needed, unless the system has some kind of automatic paging arrangement.
The principal results of this effort show that the building of library components is well supported by the various EBCDIC constructs such as FORTRAN, general-purpose editors and word processors. However, the present state of the art is such that each aspect of the project will be tested in practice. One would like to see that the machine and accuracy independent techniques used here produce elementary main function routines that can be efficiently handled as fully as other assembly language routines. Such comparisons cannot be made at this time and will depend sensitively on the optimization effectiveness of the particular quality high level compiler. Assuming that the optimization procedures to be used are reasonably effective, the family of machine and accuracy independent routines described here should be as efficient as hand-coded assembly language routines.
III. Recommendations for Further Study

There are several issues that need yet to be completed before a final

stable ANSI library of elementary functions can be distributed for general use

in a production environment.

1. The existing routines of the package presented here need to be further

thoroughly exercised and benchmarked. This was not done here because of the

quality of the the compiler available and because of time limitations.

2. The packages here need to be compiled with a production quality Ada

compiler and the resulting machine code examined and compared to hand written

assembly language routines for efficiency. This would be a measure both of the

effectiveness of the Ada compiler and optimizer and of the assembly language

routines themselves.

3. The routines written here are meant for floating point operations.

Similar routines for integer and fixed point operations need to be written

and examined in detail for efficiency.

4. The routines presented here are written primarily for rounding time,

accuracy, and speed. Additional time requirements are in addition to the accuracy

of the library itself. The environment under which the routines are used should

also be considered.

5. Finally, in an ideal environment, the breadth of the function

library needs to be considerably expanded and should also include, for

example, functions of complex variables and values.
References


**Package**

1. **Package**
2. **Function**
3. **Function**
4. **Function**
5. **Function**
6. **Function**
7. **Function**
8. **Function**
9. **Function**
10. **Function**
11. **Function**
12. **Function**
13. **Function**
14. **Function**
15. **Function**
16. **Function**
17. **Function**
18. **Function**
19. **Function**
20. **Function**
21. **Function**
22. **Function**
23. **Function**
24. **Function**
25. **Function**
26. **Function**
27. **Function**
28. **Function**
29. **Function**
30. **Function**
31. **Function**
32. **Function**
33. **Function**
34. **Function**
35. **Function**
36. **Function**
37. **Function**
38. **Function**
39. **Function**
40. **Function**
41. **Function**
42. **Function**
43. **Function**
44. **Function**
45. **Function**
46. **Function**
47. **Function**
48. **Function**
49. **Function**
50. **Function**
51. **Function**
52. **Function**
function ABS(REAL) return REAL
function ASIN(REAL) return REAL
function ACOS(REAL) return REAL
function ATAN(REAL) return REAL
function COSH(REAL) return REAL
function TANH(REAL) return REAL
function SINC(REAL) return REAL
function SIN(REAL) return REAL
function TAN(REAL) return REAL
function COS(REAL) return REAL

end ALL_UTIL

function * (REAL, REAL) return REAL is
  begin
    return A * B;
  end *

function */ (REAL, REAL) return REAL is
  begin
    return A / B;
  end */

function */ (REAL, REAL) return REAL is
  begin
    return A / (B * C);
  end */

function */ (REAL, REAL) return REAL is
  begin
    return A / (B * C * D);
  end */

function */ (REAL, REAL) return REAL is
  begin
    return A / (B * C * D * E);
  end */

function */ (REAL, REAL) return REAL is
  begin
    return A / (B * C * D * E * F);
  end */
function */(X1,Y2,FL0AT 1)lNTERGE1) return LONG_FLOAT is
   begin
      return A/W*MC_F(L0AT(1));
   end */;

function ASK_4T((X1:REAL) RETURN INTEGER is
   begin
   if X < 1.0 then
      return 1;
   end if;

function TestConstant 1 :: 1 is // := named because of nun in GNU compiler vers

function TO_0LINT(1) RETURN LONG_INT is
   begin

function TO_FLOA T(1) RETURN FLOAT is
   begin

function CP-INT(1) RETURN INTEGER is
   begin

function CP_0L0NT(1) RETURN LONG lNT is
   begin

function CP FLOAT(1 RETURN FLOAT is
   begin

function CP_REAL(1) RETURN REAL is
   begin

function CP_FIX(1) RETURN INTEGER is
   begin

function CP_0LFIX(1 RETURN LONG lNT is
   begin

function CP_REAL(1 RETURN REAL is
   begin

--------------------------

A-3
---

function C: REAL
129 begin if x then return x;
130     else return y;
131     end if;
132 end C;

134 function C: REAL
135 begin if x then return x;
136     else return y;
137     end if;
138 end C;

146 function U001: REAL) return REAL is --
147    -- (prema INLINLI)
148    -- fewer in assembly code.
149    begin return 1 mod 2 /= 0;
150    end U001;

156 function ROUND(A: ARRAY) return INTEGER is --
157    -- (prema INLINLI)
158    -- allows version 11.4 of compiler THUCATPS TOWARDS ZERO
159    -- instead of rounding for INT32.
160    begin if x > 0.0 then return INTEGER(x + 0.5);
161        else return INTEGER(x - 0.5);
162        end if;
163 end ROUND;

169 function "rem"(A, B: ARRAY) return ARRAY is --
170    -- (prema INLINLI)
171    -- fewer in assembly if there is a machine
172    -- code instruction for remainder divide of FLOATs.
173    begin return(A - iNFALT(ROUND(A/B)))
174    end "rem";

178 function SIGN(Z, X: ARRAY) return ARRAY is --
179    -- (prema INLINLI)
180    begin if x < 0.0 then return -x;
181 end SIGN;

---
function POLY(X:REAL; A:ARRAY REAL) return REAL is
  -- pragma INLINE; -- ?
  begin
    TEMP:REAL := A(A'LAST);
    for J in reverse 0..A'LAST-1 loop
      TEMP := A(J) + X*TEMP;
    end loop;
    return TEMP;
  end POLY;
end;

function JUO-POLY(X:REAL; A:ARRAY REAL) return REAL is
  -- pragma INLINE;
  begin
    return X*POLY(X*X, A);
  end JUO-POLY;
end;

function VFKA-POLY(X:REAL; A:ARRAY REAL) return REAL is
  -- pragma INLINE;
  begin
    return POLY(X*X, A);
  end VFKA-POLY;
end;

-- The Elementary Functions in ADA

-- In the past the elementary functions have been written in
-- assembly code and incorporated into the higher order language
-- or in part by being intrinsics of the language and were handled as
-- part of the compiler. The accuracy was fixed at the accuracy of
-- the machine for which the compilation was done. In Ada, however,
-- neither the machine nor the accuracy is known until compile time.
-- In most cases these functions are mapped onto some reduced range
-- and then approximated over that range by a Chebyshev polynomial
-- which are Chebyshev polynomials. (The advantage of the Chebyshev polynomials
-- are that they give a smaller maximum error over the approximation
-- range than any other polynomial of the same degree.)
-- The Chebyshev polynomials themselves are not used but the sums
-- of their coefficients of like powers of the variable are used as
-- the constants in the power series. Since the Chebyshev polynomials
-- contain all powers up to the order of the polynomial, adding one
-- more term (to get more accuracy for example) changes all the terms
-- in the power series. This means that the number and values of the
-- coefficients must be calculated at compile time (or later) when
-- required accuracy is known.
subtype INT_1..2 is INTEGER range 1..2;
subtype INT_0..1 is INTEGER range 0..1;
KMAX:constant := 10;
type A_LONG_FLOAT is Array(INTEGER range <>) of LONG_FLOAT;

PI :constant := 3.14159_26535_89793_23846_26433;
SORT_2 :constant := 1.41421_35026_23881_89793_23846_26433;
LN_2 :constant := 0.9314_71805_59945_30342_17233;

-------------------------

--- CHEBSHEV EXPANSION

--- Let
--- f(x) = \sum_{k=0}^{KMAX}(\frac{C(k,1)\times(x/a)^{1/2}}{2})
--- (0 \leq x \leq 1)
--- where
--- T(k,x/a) = \sum_{j=0}^{KMAX}(C(k,1)\times(x/a)^{1/2})
--- (0 \leq j \leq KMAX)
--- n(k) = (2/\Pi)\times\text{Integral}(f(x)*1(k,x/a)/sqrt(1-(x/a)^{2})), k>0
--- (-1 \leq x \leq 1)
--- = (2/\Pi)\times\text{Integral}(f(x)*cos(y))\timescos(k*y))
--- (0 \leq y \leq \Pi)
--- then
--- t(x) = \sum_{k=0}^{KMAX}(\frac{C(k,1)\times(x/a)^{1/2}}{2})
--- (0 \leq x \leq 1)
--- where
--- A(1) = (1/a)^{1/2}\times\sum_{k=0}^{KMAX}(C(k,1)\times(x/a)^{1/2})
--- (0 \leq x \leq 1)
--- -------------------------

package FFLT.TXT is new FFLT.LUT(REAL);
use FFLT.LUT;

procedure ICH_SHI

--- About A_LONG_FLOAT;
--- \#A_LONG_FLOAT_LUT;
--- L11-T1.1_LUT;
--- L11-T1.2_LUT;
--- L11-T1.3_LUT
--- [L11-T1.4_LUT] is --
--- \# l = 2, l = 0 gives even lower series,
--- \# l = 2, l = 1 gives odd lower series,
--- \# l = 1, l = 0 gives even lower and odd terms,
-- L = 1, I = 1 is undefined.
-- The CHEBYSHEV constants:
function C(N, J:INTEGER) return INTEGER is --
-- This is not efficient, time wise, but no matter;
-- only used to initialize package constants,
-- never used in real time,
-- This IS efficient space wise, since it eliminates
-- the need of storing the sparse matrix C(N, J).
begin
if H < 0 or else J < 0 or else J > N
or else JDD(N+J) then return 0;
elsif (J=0 or else (J=1 and J=1) then return 1;
else return 2*C(N-1,J-1) - C(N-2, J);
end if;
end C;

begin == TCH_SUM
for J in 0..A'LAST
loop
TEMP := 0.0;
for K in J..A'LAST
loop
CTEMP := C(K+1, L*J+1);
if CTEMP /= U then
TEMP := TEMP + CTEMP*(H);
end if;
end loop;
A(J) := KEAL(TEMP*(-1)**(-L*J-1));
PUT(A(J),<10TH X> 15);
end loop;
end TCH_SUM;

function LIM(H:LONG_FLOAT; WFA1, I := 10.0) return INTEGER is --
WLENGTH := 1;
REAL DIGITS := INTEGER := INTEGER(FRAIL*DIGITS);
begin
PUT(w, WLENGTH := 15); new_line;
PUT(W, WLENGTH := 15); new_line;
1000
if H <= FRAIL(H(K)) < W**REAL_DIGITS then
return H := 1;
elsif K <= A'LAST then return K;
end if;
K := H + 1;
end loop;
end LIM;

-- Note W.MAX(-1) = maximum of range of variable in function XXX;
-- 1
-- W_MIN(-1) = 01/4 W_CUS(-1) = 2/1
-- W_MAX(-1) = 01/3 W_TA1(-1) = 01/3
-- W_SQR(-1) = 0.25 W_EXP(-1) = 0.25
-- W_ATAN(-1) = 0.25 W_MIN(-1) = 0.25
-- m ASINH(-1) = 0.375

-- These constants are believed to be accurate to the 24th decimal digit.

\[ \text{ASINH(-1) = 0.375} \]

\[ \text{ASINH(-2) = 1.04113703040149298,} \]

\[ \text{ASINH(-3) = 1.4705891052683385,} \]

\[ \text{ASINH(-4) = 1.8635229011614972,} \]

\[ \text{ASINH(-5) = 2.223143388837291,} \]

\[ \text{ASINH(-6) = 2.5596374836507260,} \]

\[ \text{ASINH(-7) = 2.8664979065778954,} \]

\[ \text{ASINH(-8) = 3.1462642980377497,} \]

\[ \text{ASINH(-9) = 3.395351522418500,} \]

\[ \text{ASINH(-10) = 3.6176177879573984,} \]

\[ \text{ASINH(-11) = 3.8208575750171414,} \]

\[ \text{ASINH(-12) = 4.0049745088208303,} \]

\[ \text{ASINH(-13) = 4.1707466677763689,} \]

\[ \text{ASINH(-14) = 4.3200517563071739,} \]

\[ \text{ASINH(-15) = 4.4540070391010320,} \]

\[ \text{ASINH(-16) = 4.5742805499823643,} \]

\[ \text{ASINH(-17) = 4.6817084732167592,} \]

\[ \text{ASINH(-18) = 4.7770312757968168,} \]

\[ \text{ASINH(-19) = 4.8606761436004172,} \]

\[ \text{ASINH(-20) = 4.9330098216341743,} \]

\[ \text{ASINH(-21) = 4.9945869837153044,} \]

\[ \text{ASINH(-22) = 5.0462147714206453,} \]

\[ \text{ASINH(-23) = 5.0880688875255400,} \]

\[ \text{ASINH(-24) = 5.1195897860867682,} \]

\[ \text{ASINH(-25) = 5.1412331352050394,} \]

\[ \text{ASINH(-26) = 5.1533249075576341,} \]

\[ \text{ASINH(-27) = 5.1659852603241103,} \]

\[ \text{ASINH(-28) = 5.1792346486703961,} \]

\[ \text{ASINH(-29) = 5.1921854719370384,} \]

\[ \text{ASINH(-30) = 5.2058482333460274,} \]

\[ \text{ASINH(-31) = 5.2192219199813093,} \]

\[ \text{ASINH(-32) = 5.2323125636033227,} \]

\[ \text{ASINH(-33) = 5.2451221109633716,} \]

\[ \text{ASINH(-34) = 5.2576526119123525,} \]

\[ \text{ASINH(-35) = 5.2699130775556940,} \]

\[ \text{ASINH(-36) = 5.2818945196020908,} \]

\[ \text{ASINH(-37) = 5.2936070553809021,} \]

\[ \text{ASINH(-38) = 5.3050516858859794,} \]

\[ \text{ASINH(-39) = 5.3162373942235460,} \]

\[ \text{ASINH(-40) = 5.3271641890007498,} \]

\[ \text{ASINH(-41) = 5.3378320962086154,} \]

\[ \text{ASINH(-42) = 5.3482410928000021,} \]

\[ \text{ASINH(-43) = 5.3583910888530150,} \]

\[ \text{ASINH(-44) = 5.3683720844820224,} \]

\[ \text{ASINH(-45) = 5.3781830797945983,} \]

\[ \text{ASINH(-46) = 5.3878240758091710,} \]

\[ \text{ASINH(-47) = 5.3972940715275220,} \]

\[ \text{ASINH(-48) = 5.4065930669507432,} \]

\[ \text{ASINH(-49) = 5.4157220621786665,} \]

\[ \text{ASINH(-50) = 5.4246800571213897,} \]
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{y2} & \textbf{y1} & \textbf{y0} & \textbf{y} & \textbf{y} \\
\hline
452 & 1.0000 & 0.0000 & 0.0000 & 0.0000 \\
453 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
454 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
455 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
456 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
457 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
458 & 1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
459 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
460 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
461 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
462 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\
463 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
464 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
465 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
466 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
467 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
468 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
469 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
470 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
471 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
472 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
473 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
474 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
475 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
476 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
477 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
478 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
479 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
480 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
481 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
482 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
483 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
484 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
485 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
486 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
487 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
488 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
489 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
490 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
491 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
492 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
493 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
494 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
495 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
496 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
497 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
498 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
499 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
500 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
501 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
502 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
503 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
504 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
505 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
506 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
507 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
508 & \textbf{y2} \textbf{y1} \textbf{y0} \textbf{y} \textbf{y} \\
\hline
\end{tabular}
509  H_ATAN  =  CONSTANT REAL  IF( H_ATAN/-1)
The hyperbolic

The hyperbolic functions

The hyperbolic functions
FUNCTION SING(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN LOOP ---
    IF AMS(Z) <= 3*PI/4 THEN RETURN SING.SMALL(Z); 
    ELSE IF AMS(Z) <= PI THEN Z := SING(Z, PI - AMS(Z)); 
    END IF;
  END LOOP;
END SING;---

FUNCTION COS(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN IF AMS(Z) <= 3*PI/4 THEN RETURN COS.SMALL(Z); 
  END IF;
  BEGIN LOOP ---
    RETURN SING(P) - AMS(Z)); 
  END LOOP;
END COS;---

FUNCTION TAN(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN LOOP ---
    IF AMS(Z) <= 5*PI/8 THEN RETURN TAN.SMALL(Z); 
    ELSE IF AMS(Z) <= PI THEN Z := SING(Z, PI - AMS(Z)); 
    END IF;
  END LOOP;
END TAN;---

FUNCTION COT(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN LOOP ---
    IF AMS(Z) <= PI/4 THEN RETURN COT.SMALL(Z); 
    ELSE IF AMS(Z) <= PI/2 THEN Z := SING(Z, PI - AMS(Z)); 
    END IF;
  END LOOP;
END COT;---

FUNCTION SEC(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN LOOP ---
    IF AMS(Z) <= PI/2 THEN RETURN SEC.SMALL(Z); 
    ELSE IF AMS(Z) <= 3*PI/4 THEN Z := SING(Z, PI - AMS(Z)); 
    END IF;
  END LOOP;
END SEC;---

FUNCTION CSC(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN LOOP ---
    IF AMS(Z) <= PI/2 THEN RETURN CSC.SMALL(Z); 
    ELSE IF AMS(Z) <= PI/4 THEN Z := SING(Z, PI - AMS(Z)); 
    END IF;
  END LOOP;
END CSC;---

FUNCTION ACOS(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN LOOP ---
    IF AMS(Z) <= 3*PI/4 THEN RETURN ACOS.SMALL(Z); 
    ELSE IF AMS(Z) <= PI THEN Z := SING(Z, PI - AMS(Z)); 
    END IF;
  END LOOP;
END ACOS;---

FUNCTION ASIN(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN LOOP ---
    IF AMS(Z) <= 1*PI/4 THEN RETURN ASIN.SMALL(Z); 
    ELSE IF AMS(Z) <= PI/2 THEN Z := SING(Z, PI - AMS(Z)); 
    END IF;
  END LOOP;
END ASIN;---

FUNCTION ATAN(Z:REAL) RETURN REAL IS ---
  "Rounding"
  ZREAL := Z;
  BEGIN LOOP ---
    IF AMS(Z) <= 3*PI/8 THEN RETURN ATAN.SMALL(Z); 
    ELSE IF AMS(Z) <= PI/4 THEN Z := SING(Z, PI - AMS(Z)); 
    END IF;
  END LOOP;
END ATAN;---
function ASIN(1:MEAN) return REAL is
begin
  -- Preine unable --
  if ABS(Y) < 2*ASIN(RAND) then RETURN ODDOM(Y, P_ASIN);
  elsif ABS(Y) > 1.0 then RETURN SIGN(Y, PI/2);
  end if;
  RETURN ASIN((1.0 + Y)*(1.0 - Y));
end ASIN;
function ACSH(X:REAL) return REAL is --
  pragma INTEGRAL;
  Y:REAL := 1.0 - X;
  begin -- assert X:INTEG <= 1.0;
  if Y < ACSH_CONST then return SQR(T*2.0 + Y/3.0);
  end if;
  return Y/2 + ACSH(Y);
end ACSH;

function SN4(X:REAL) return REAL is --
  pragma INTEGRAL -- ?
  E:REAL := ABS(X);
  begin if E < M_SINH_RAN then return UNP_BOLY(E, F_SINH);
  end if;
  E := EXP(E);
  if E < 0.0 then return -REAL(LARGE)/2;
  end if;
  X := 1.0/E/2;
end SINH;

-- Hart, page 104 (range[0.0,5]):

<table>
<thead>
<tr>
<th>X</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>err</td>
<td>1.7</td>
<td>4.2</td>
<td>7.0</td>
<td>10.1</td>
<td>13.3</td>
<td>16.7</td>
<td>20.3</td>
<td>23.9</td>
</tr>
<tr>
<td>X</td>
<td>3.0</td>
<td>5.2</td>
<td>7.7</td>
<td>10.3</td>
<td>13.1</td>
<td>16.0</td>
<td>19.1</td>
<td></td>
</tr>
</tbody>
</table>
function \text{COSH}(x: \text{REAL}) \text{ return } \text{REAL} \text{ is } 
\begin{align*}
\text{begin} & \quad \text{if } \text{AN}(x) \leq \text{H\_COSH\_RA} \text{ then return } \text{EVEN\_POLY}(x, \text{P\_COSH}) \; \text{end if;}
\text{let } \epsilon = \text{EXP}(x); 
\text{if } \epsilon = 0.0 \text{ then return } \text{REAL\_LARGE}/2; 
\text{end if;}
\text{return } \epsilon + 1.0/\epsilon)/2; 
\text{end COSH;}
\end{align*}

---

\text{Range}(0, 1.0): 
\begin{align*}
n &= 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7
\end{align*}

---

\text{TANH}\_\text{COSH}\_\text{REAL;}

\begin{align*}
\text{function } \text{TANH}(x: \text{REAL}) \text{ return } \text{REAL} \text{ is } 
\text{begin} & \quad \text{if } \text{AN}(x) \leq \text{H\_TANH\_RA} \text{ then return } \text{ODD\_POLY}(x, \text{P\_TANH}); 
\text{elsif } \text{AN}(x) >= \text{TANH\_CONST} \text{ then return } \text{SIGN}(x, 1.0); 
\text{end if;}
\text{let } \epsilon = \text{EXP}(x); 
\text{return } (\epsilon x + 1.0)/(\epsilon x + 1.0); 
\text{end TANH;}
\end{align*}

---

\text{TANH}\_\text{CONST}\_\text{REAL;}

\begin{align*}
\text{function } \text{TANH}(x: \text{REAL}) \text{ return } \text{REAL} \text{ is } 
\text{begin} & \quad \text{if } \text{AN}(x) \leq \text{H\_TANH\_RA} \text{ then return } \text{ODD\_POLY}(x, \text{P\_TANH}); 
\text{elsif } \text{AN}(x) >= \text{TANH\_CONST} \text{ then return } \text{SIGN}(x, 1.0); 
\text{end if;}
\text{let } \epsilon = \text{EXP}(x); 
\text{return } (\epsilon x + 1.0)/(\epsilon x + 1.0); 
\text{end TANH;}
\end{align*}

---

\text{ATANH;}

\begin{align*}
\text{function } \text{ATANH}(x: \text{REAL}) \text{ return } \text{REAL} \text{ is } 
\text{begin} & \quad \text{assert } \text{AN}(x) < 1.0; 
\text{if } \text{AN}(x) \leq \text{H\_ATANH\_RA} \text{ then return } \text{ODD\_POLY}(x, \text{P\_ATANH}); 
\text{elsif } \text{AN}(x) >= 1.0 \text{ then return } \text{SIGN}(x, \text{LN\_REAL\_LARGE}/2); 
\text{end if;}
\text{return } \text{LN}((1.0 + x)/(1.0 - x))/2; 
\text{end ATANH;}
\end{align*}

---

\text{Range}\{0, 1.0\}: 
\begin{align*}
n &= 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7
\end{align*}
function ASINH(X:REAL) return REAL is
  -- pragma INLINE; -- ?
begin
  if ABS(X) <= P_Asinh_Ran then return ODD_POLY(X, P_Asinh); 
  elsif ABS(X) >= SQRT_REAL_LARGE/2 then
    return SIGN(X, LN(Abs(X)) + LN_2);
  end if;
  return SIGN(X, LN(Abs(X) + SQRT(1.0 + X*X)));
end ASINH;

Range[0, 0.375];

function ACOSH(U:REAL) return REAL is
  -- assert U) > 1.0;
begin
  if U <= 1.0 then return 0.0;
  elsif U <= ACOSH_CONST then return SQRT(Y*(2.6 - Y/3));
  elsif U >= SQRT_REAL_LARGE/2 then return LN(Abs(U)) + LN_2;
  end if;
  return LN(SQRT(Y*(1.0 + U)) + U);
end ACOSH;

TWO_P_FIV(REAL); 

TWO_P_FIV: constant REAL := 2.5; -- needed by compiler version 11.4.

begin
  -- Initialization of package THC_FUN.
TCH_SUM(P_SIN, 3_SIN, 2, 0);
TCH_SUM(P_COS, 3_COS, 2, 0);
TCH_SUM(P_TAN, 3_TAN, 2, 0);
TCH_SUM(P_SINH, 3_SINH, 2, 0);
TCH_SUM(P_COSH, 3_COSH, 2, 0);
TCH_SUM(P_TANH, 3_TANH, 2, 0);
TCH_SUM(P_EXP, U_EXP, 1, 0);
TCH_SUM(P_SINH, 3_SINH, 2, 0);
TCH_SUM(P_COSH, 3_COSH, 2, 0);
TCH_SUM(P_TANH, 3_TANH, 2, 0);
TCH_SUM(P_ARINH, 3_ARINH, 2, 0);
TCH_SUM(P_ATANH, U_ATANH, 1, 0);
TCH_SUM(P_LN, 3_LN, 2, 0);
PROCEDURE Intro is
type REL is digits 5;

type LONG_FLOAT is digits 7;

package VEN_55_EFUN is new ELE_FUN(REL, LONG_FLOAT);

use VEN_55_EFUN;

begin  -- TEST
  PUT(" TEST  OK ");
  new_line;
end TEST;

no parse errors detected
Parsing time: 742 seconds

no semantic errors detected
Translation time: 2236 seconds
APPENDIX B

A Listing of the Programs and Subroutines for Computing the Chebyshev Coefficients for the Various Elementary Math Functions

These programs are in FORTRAN but for documentation purposes, the FLECS listings are first given. The numbers in the right-hand column of the FORTRAN listings show the line to which it corresponds in the FLECS listing.
THE PROGRAM TRIGOE CALCULATES THE CHEBYSHEV AND POLYNOMIAL
COEFFICIENTS FOR ODD OR EVEN (MOSTLY TRIG) FUNCTIONS.
ALL COMPUTATIONS ARE IN DOUBLE PRECISION EXCEPT FOR THE
ERROR TOLERANCE WHICH IS IN SINGLE.
THE ACTUAL COMPUTATIONS ARE CARRIED OUT IN SUBROUTINE POE
THE MAIN PROGRAM ONLY DOES THE INTERACTIVE DIALOGUE TO
determine what is to be computed and how accurately.

THE RANGE OF THE APPROXIMATION TO THE FUNCTION IS FROM
THE user and if the user specifies A > 1 or less then
A DEFAULT AN > 1 IS USED.

PROGRAM TRIGOE INPUT/OUTPUT
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
COMMON * AB * I
EXTERNAL FANCY, FIN

IF A > 141.4 OR A < -141.4 THEN ERROR SIGNAL
IF A IS NOT AN INTEGER THEN ERROR SIGNAL
IF A IS NOT A POSITIVE NUMBER THEN ERROR SIGNAL

DATA PI,3.141592653589793238462643383279502884197

WHILE TRUE
PRINT * , "OBTAIN EVEN-TEGRAL A";
READ * , A
PRINT * , "OBTAIN FANCY";
 Call FANCY
Print * , "END";
END
00059  IF .EQ. 10COSH
00060       CALL FOE(FMIN)
00061       FIN
00062 C     IF .EQ. 10HSUAL
00063 C     IF .LT. 0.000 AN 0.500
00064 C     CALL FOE(FSUAL), 30A
00065 C     FIN
00066 C     IF .EQ. 10SIN
00067 C    IF .LT. 2.000 AND .LT. 1.000
00068 C     CALL FOE(FSIN), 31A
00069 C     FIN
00070 C     IF .EQ. 10TAN
00071 C     IF .LT. 2.000 AND .LT. 1.000
00072 C     CALL FUL(TAN), 32A
00073 C     FIN
00074 C     IF .EQ. 10ATAN
00075 C    IF .LT. 2.000 AND .LT. 1.000
00076 C     CALL FUL(ARCTAN)
00077 C     FIN
00078 C     IF .EQ. 10RATAN
00079 C     IF .LT. 2.000 AND .LT. 1.000
00080 C     CALL FUL(RATAN), 33A
00081 C     FIN
00082 C     IF .EQ. 10ZET
00083 C    IF .LT. 2.000 AND .LT. 1.000
00084 C     CALL FUL(ZET), 34A
00085 C     FIN
00086 C     IF .EQ. 10FMIN
00087 C     IF .LT. 2.000 AND .LT. 1.000
00088 C     CALL FUL(FM), 35A
00089 C     FIN
00090 C     IF .EQ. 10FM
00091 C    IF .LT. 2.000 AND .LT. 1.000
00092 C     CALL FUL(FMIN), 36A
00093 C     FIN
00094 C     IF .EQ. 10STN
00095 C    IF .LT. 2.000 AND .LT. 1.000
00096 C     CALL FUL(STN), 37A
00097 C     FIN
00098 C     IF .EQ. 10STN
00099 C    IF .LT. 2.000 AND .LT. 1.000
00100 C     CALL FUL(FM), 36A
00101 C     FIN
00102 C     IF .EQ. 10STN
00103 C    IF .LT. 2.000 AND .LT. 1.000
00104 C     CALL FUL(FMIN), 35A
00105 C     FIN
00106 C     FIN
AS IS THE ARRAY OF COEFFICIENTS FOR THE POLYNOMIAL
APPROXIMATION OF $S_5(x)$, COMPUTED IN ACME.

THE OBJ. FUNCTION FOR $S(x)$ AS $S_5(x)$, $S_6(x)$ ARE COMPUTED.
THE OBJ. FUNCTION $S_7(x)$ FOR $S_5(x)$ AS $S_7(x)$, $S_8(x)$ ARE.

ARAN IS IN ORDER THAT THE RELATIVE ERROR BE UNIFORMLY
DISTRIBUTED IN THE COEFFICIENTS APPROXIMATION.

THE ALGORITHM'S THEOREM APPROXIMATION IS VERIFIED.
IMPLEMENTATION CAN THEREFORE FOR THE INTERVAL $[0, 1]$.

SUBTRACT THE POLYNOMIALS.
IMPLEMENTATION FOR THE INTERVAL $[0, 1]$.
REFERENCE VALUES.
DIMENSION VALUES.
END OF ROUTINE.
Program both calculates Chebyshev and polynomial approximations
that use both odd and even terms in their approximations of a
function (exp, log, sort). All computations are in double
precision except for the error tolerance which is in single.
The actual computations are carried out in the subroutine Pboth,
the main program only does the interactive dialog to determine
what is to be computed and how accurately.

The range of the approximation to the function is from AL to AR;
there are no default values for AL or AR.

Program both: input output
Implicit double precision (A-H,I-Z)
Common A, AR, I, AL
Common consts. F1* REAL
External +G1* ALOG
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2
F1  F2

WHILE *.TRUE.
   PRINT * 'FUNCTION',
   READ *, I
   PRINT * 'G1', I
   FORMAT(*A00)
   PRINT * 'A', A =
   READ *, A
   PRINT * 'A', A
   PRINT * 'A', A
   READ *, A
   PRINT * A
   FORMAT(*A10)
   IF (.NOT. X) THEN
      PRINT *'FUNCTION',
      CALL G1F2 (X, Y)
   ELSE
      PRINT *'FUNCTION',
      CALL F1F2 (X, Y)
   END IF
END

END

END
SUBROUTINE CHEBY(N) COMPUTES THE CHEBYSHEV AND POLYNOMIAL

N
cOEFICIENTS FOR THE REGULATED FUNCTION.

CASE IN THE FUNCTION TO BE APPROXIMATED (IE EXP, SQRT, ETC)


do not exist if 

THE INTEGRAL OVER THETA FROM 0 TO PI TO GIVE THE

CHEBYSHEV COEFFICIENTS, CAN - 1 AND N ARE AVAILABLE THRU

IT IS THE ARRAY OF COEFFICIENTS FOR CONVECTIONS CHEBYSHEV

CONVENTS TO POLYNOMIAL COEFFICIENTS, THESE ARE CALCULATED

IT IS THE ARRAY OF CHEBYSHEV COEFFICIENTS FOR FUNCTION SSSS.

AS IS THE ARRAY OF COEFFICIENTS FOR THE POLYNOMIAL APPROXIMATION

OF SSSS, COMPUTED IN ACOEF.

THE POLYNOMIAL APPROXIMATION IS VERIFIED BY GENERATING THE

RELATIVE ERROR CURVE OVER THE INTERVAL AL TO AK.

SUBROUTINE CHEBY(N) SSSS

COMMON N, AL, L, A

DIMENSION IC(25*25), ICD(51)

EQUIVALENCE (ICD(51), IC(1:1))

DIMENSION A5(25), AS(42)

EQUIVALENCE (A5(25)-AS(1))

DIMENSION SSSSS(13), SSSSS(25)

EQUIVALENCE (SSSS(13)-SSSSS(1))

EXTERNAL SSSSS, SSSST

INTEGER I

DATA 0/0/, ZEFD/0.000/

DATA C3SSS/25*0.000/ 00193 DATA A5/C50*0.010/

DATA ICD(50)*1000, 1C/6.25*1000/

DATA I1/10H ++IS/10H************/

DATA PI/3.14159 26535 89793 23846 26433 83279D0/

PRINT *, ' QUAD ERR TOL ',

READ *, OE

PRINT *, OE

PRINT *, ' FUNCT ERR TOL ',

READ *, OEF

PRINT *, OEF

N = -1

REPEAT WHILE (DABS(CSSSS(N)).GT.OEF)

N = N + 1

CSSSS(N) = 2*ROMBER(ZERO, PI, SSSST, OE)/PI

IF (N.EQ.0) CSSSS(N) = CSSSS(N)/2

PRINT S, N, CSSSS(N)

FORMAT(IS, F35.25)

FIN

B-9
00114  \text{N = N - 1}
00115  \text{SUM = DABS(CSSSS(0))}
00116  \text{DO (J=1,N) SUM = SUM + DABS(CSSSS(J))}
00117  \text{PRINT * N, * SUMABS(SSSS) = ', SUM}
00118  \text{SUM = (CSSSS(0))}
00119  \text{DO (J=1,N) SUM = SUM + (CSSSS(J))}
00120  \text{PRINT * N, * SUM(SSSS) = ', SUM}
00121  \text{OEK = DABS(CSSSS(AK) - SUM)}
00122  \text{CALL TCH(IC, N)}
00123  \text{XBAR = (AK + AL)/2}
00124  \text{CALL ACOEF(AS, CSSSS, IC, (AK - AL)/2, N)}
00125  \text{DES = DABS(SSSS(XBAR) - AS(0))}
00126  \text{OE = AMAX1(DES, OEK)}
00127  \text{PRINT * N, DES, OEK,}
00128  \text{PRINT * AK = '; AK,}
00129  \text{PRINT * AL = '; AL}
00130  \text{JMAX = 5}
00131  \text{DELTA = (AK - AL)/249.9}
00132  \text{X = AL}
00133  \text{WHILE (X .LT. AK)}
00134  \text{. DO (J=1,JMAX)}
00135  \text{. \text{JMAX = 5}}
00136  \text{DELTA = (AK - AL)/249.9}
00137  \text{X = AL}
00138  \text{WHILE (X .LT. AK)}
00139  \text{. DO (J=1,JMAX)}
00140  \text{. \text{JMAX = 5}}
00141  \text{. DELTA = (AK - AL)/249.9}
00142  \text{. X = AL}
00143  \text{FSSSS = POLF(X - XBAR, AS, N)}
00144  \text{ESSSS = SSSS(X)}
00145  \text{. PRINT * FSSSS, ESSSS}
00146  \text{. OEKRS = (ESSSS - FSSSS)/ESSSS}
00147  \text{. IF (OEKRS.GT.OEMAXS) OEMAXS = OEKRS}
00148  \text{. IF (OEKRS.LT.OEMINS) OEMINS = OEKRS}
00149  \text{. \text{FIN}}
00150  \text{. PRINT * OEMINS, OEMAXS, OEMINCP, OEMAXCP}
00151  \text{. \text{FIN}}
00152  \text{. \text{FIN}}
00153  \text{. \text{FIN}}
00154  \text{. NS = 30#OEMINS/DE}
00155  \text{. ML = 30#OEMAXS/DE - NS}
00156  \text{. NS = 65 + NS}
00157  \text{. PRINT 19, (1B, J=1,NS), IS, (1B, J=1,ML), IS}
00158  \text{. FORMAT(1X, 130A1)}
00159  \text{. \text{FIN}}
00160  \text{RETURN}
00161  \text{END}

*FLECS VERSION 22.51*

-------------------------------

00162  \text{C}
00163  \text{FUNCTION TWO(X)}
00164  \text{IMPLICIT DOUBLE PRECISION(A-H,P-Z)}
00165  \text{COMMON/CONSTS/PI, ALN2}
00166  \text{TW = DEXP(X*ALN2)}
00167  \text{PRINT*, * X = ', X, * TWO(X) = ', TWO}
00168  \text{RETURN}
00169  \text{END}
00170 C
00171 FUNCTION FTWOT(THETA)
00172 IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00173 COMMON N, AN, I, AL
00174 COST = DCOS(THETA)
00175 COSNT = DCOS(N*THETA)
00176 FTWOT = TWO((AK + AL + (AK - AL)*COST)/2)*COSNT
00177 RETURN
00178 END

(FLECS VERSION 22.51)

00179 C
00180 FUNCTION FCRTT(THETA)
00181 IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00182 COMMON N, AN, I, AL
00183 EXTERNAL CBR
00184 COST = DCOS(THETA)
00185 COSNT = DCOS(N*THETA)
00186 FTWOT = CBR((AK + AL + (AK - AL)*COST)/2)*COSNT
00187 RETURN
00188 END

(FLECS VERSION 22.51)

00189 C
00190 FUNCTION CBR(X)
00191 IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00192 CBR = 0
00193 IF X .LE. 0 THEN RETURN
00194 CBR = ASINH(EXP(MDL+16)+2*X)**3
00195 ELSE CBR = 1
00196 RETURN
00197 END

(FLECS VERSION 22.51)
COPY COMPI

PROGRAM BOTH(INPUT OUTPUT)
IMPLICIT DOUBLE PRECISION (A-H, I-P, Z)
COMMON M, AK, I, AL
COMMON/CONSTS/N, ALN2
EXTERNAL FLOG, ALOG,
X FCBRT, CBRT,
X FSQRT, SQAC,
X FTWOT, TWO,
X FEXP, EXP,
X FEXIX, EXIX,
X FFASTM, ASIN,
X FSQRT, SQAC,
X FSORT, SORT,
PI = 3.14159 26535 89793 23846 26433 83279110
ALN2 = ALOG(2.000)

9999 IF (.NOT. (.TRUE.)) GO TO 9999B
PRINT *, ' FUNCTION = ',
READ #, I
PRINT 10, I
10 FORMAT (I10)
PRINT *, ' AK = ',
READ #, AK
PRINT *, ' AL = ',
READ #, AL
PRINT *, AL
IF (.NOT. (I .EQ. 10NCBRT)) GO TO 99996
CALL BOTH(FCBRT, CBRT)
GO TO 99997
99996 IF (.NOT. (I .EQ. 10TWOT)) GO TO 99995
CALL BOTH(FTWOT, TWO)
GO TO 99997
99995 IF (.NOT. (I .EQ. 10HEXIX)) GO TO 99994
CALL BOTH(FEXIX, EXIX)
GO TO 99997
99994 IF (.NOT. (I .EQ. 10LOG)) GO TO 99993
CALL BOTH(FLOG, ALOG)
GO TO 99997
99993 IF (.NOT. (I .EQ. 10HASASIN)) GO TO 99992
CALL BOTH(FFASM, ASIN)
GO TO 99997
99992 IF (.NOT. (I .EQ. 10HEXEXP)) GO TO 99991
CALL BOTH(FEXP, EXP)
GO TO 99997
99991 IF (.NOT. (I .EQ. 10HESORT)) GO TO 99990
CALL BOTH(FSORT, SORT)
GO TO 99997
99990 IF (.NOT. (I .EQ. 10HASQAC)) GO TO 99989
CALL BOTH(FSQAC, SQAC)
GO TO 99997
99989 PRINT *, ' I'LL ASK AGAIN. ',
99997 GO TO 99999
99999 END

SUBROUTINE FLOG(FSSST1, S555)
IMPLICIT DOUBLE PRECISION (A-H, I-P, Z)
COMMON N, AK, I, AL
DIMENSION IC(25*25), I(NEG(51))
EQUIVALENCE (IC(1), IC(1))
DIMENSION AS(25*25), ASO(2)
EQUIVALENCE (ASO(2), AS(1))
DIMENSION CSSS50(3), CSSS51(25)
EQUIVALENCE (CSSS50(3), CSSS51(25))
EXTERNAL SSS5, FSS5
INTEGER 0
DATA 0/0/, ZERO/0.000/
DATA CSSS52/250.000/
DATA AS/250.0000/
DATA ICNEG/50*1000/ ID/45*10000/
DATA IF/100000/ IS/10000000000/
DATA FI/3.14159265358979323846264338327950/
PRINT * QUAD ERR TOL *
READ * DE
PRINT *
PRINT * 'FUNCTION ERR TOL *
READ * DE
PRINT *
N = -1
GO TO 99998
99998 IF NOT(AABS+CSS5) < K1, DE = GO TO 99997
99997 N = N + 1
CSS5(N) = ALIERM(ZERO + FI + FSS5) DE/N
IF (N.1000) CSS5(N) = CSS5(N-1)
PRINT * N, CSS5(N)
5 FORMAT (5 CSS5
GO TO 99997
99996 N = N - 1
SUM = AABS + CSS5(N)
GO 99998, IF N = N
SUM = SUM + AABS + CSSS5(N)
99995 CONTINUE
PRINT * N, SUMABS + CSSS5(N) = SUM
SUM = CSSS5(N)
GO 99995 J = 1. N
SUM = SUM + CSSS5(N)
99994 CONTINUE
PRINT * N, SUMCSSS5(N) = SUM
DEN = AABS + CSSS5(N)
CALL TOCIC (N)
XBAR = (AN + AL)/2
CALL AODEF (CSSS5, XBAR, AR, AL, XBAR)
RES = AABS + CSSS5(XBAR) - AR
DE = AR + 1000 + RES
PRINT * N, DE, RES
PRINT * ' AN = ', AR
PRINT * ' AL = ', AL
MAX = 5
DIFMAX = (AN, AL, 24.0)
= MAX
99994 IF NOT(A), AN = GO TO 99993
DEMAX = 0
OFMINS = 0
DO 99997 J = 1, MAX
X = X + DELIA
FSS5 = FOTIX + XBAR, RES, N
CSSS5 = CSSS5 + X
UIEFA = FSS5 - CSSS5, ESSE1, ESSE5
IF K0UIS, GI, DEMAX = DEMAX + OFMINS
IF 0 K0UIEFA, OFMINS = K0UIEFA
99993 CONTINUE
NS = 4*OFMINS
DE = 400.0*MAX + NS

THIS FUNCTION DOES ROMBERG QUADRATURE ON THE FUNCTION F FROM THE LOWER LIMIT A TO THE UPPER LIMIT B, AND TO THE REQUESTED ACCURACY DE.

THIS VERSION DOES NOT USE THE END POINTS.

FUNCTION ROMBRG(A, B, F, DE)
IMPLICIT DOUBLE PRECISION(A-H, P-Z)
DIMENSION O(20)
INTEGER FOUR(20)
DIMENSION G(20), NO(2)
EQUIVALENCE (G(2), O(1))
DATA 0/0/, LMAX=11, FOUR(1) = 1/
IMAX = 1
DO (I=1,20)
  IF (L.LT.LMAX)
    LMAX = L + 1
  FOUR(L+1) = 4*FOUR(L)
END

SUM = 0
DO (I=1,IMAX)
  U = (2*IMAX - IMAX - 1.000)/IMAX
  U = U*U - U*U/2
  UP = 1.50441 - U*U
  X = A + (U + 1)*H
  SUM = SUM + F(X)*UP
END
SUM = SUM + IMAX*G(L-1)
IMAX = 2*IMAX
DO (I=1,IMAX)
  F = L - M
  O(k) = (FOUR(k)*O(k+1) - O(k))/FOUR(k) - 1
END
    ROMBRG = (B - A)*O(1)
PRINT*, ROMBRG, (O(k), K=1,I)
PRINT*, ABS(AO(I) - O(I)*(B - A))
IF (AO.LE.DE) RETURN
RETURN
THIS FUNCTION DOES ROMBERG QUADRATURE ON THE FUNCTION F FROM THE LOWER LIMIT A TO THE UPPER LIMIT B, AND TO THE REQUESTED ACCURACY OE.

FUNCTION ROMBER(A, B, F, OE)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
DIMENSION QO(20)
INTEGER FOUR(21), 00(2)
EQUIVALENCE (00(2),Q(1))
DATA O/O/ + LMAX/1/ + FOUR(1)/4/
QO(0) = (F(A) + F(B))/2
H = B - A
IMAX = 1
LMAX = L + 1
FOUR(L+1) = 4*FOUR(L)
DO (L=1,20)
    IF (L.GE.LMAX)
        LMAX = L + 1
    END

    SUM = 0
    DO (I=1,IMAX)
        SUM = SUM + F(A + H*S(I-I))
    END

    IMAX = 2*IMAX
    Q (L) = SUM/IMAX

    DO (M=L+1)
        K = L - M
        Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)
    END

    ROMBER = (B - A)*Q(O)

    DO (K=1,L)
        QD(K) = (Q(K) - Q(K-1))/(B - A)
        PRINT*, ROMBER, (QD(K), K=1,L)
    END

    QDQ = DABS((QO) - Q(1))*(B - A)
    IF (QDQ.LE.OE) RETURN

    PRINT*, QDQ
END
00107 FUNCTION SIN(X)
00108 IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00109 SIN = DSIN(X)
00110 PRINT*, ' X=', X, ' SIN(X) =', SIN
00111 RETURN
00112 END

(FLECS VERSION 22.51)

00113 C
00114 FUNCTION TAN(X)
00115 IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00116 TAN = DSIN(X)/DCOS(X)
00117 PRINT*, ' X=', X, ' TAN(X) =', TAN
00118 RETURN
00119 END

(FLECS VERSION 22.51)

00120 C
00121 FUNCTION ASIN(X)
00122 IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00123 ASIN = DATAN(X/SQRT(1-X*X))
00124 PRINT*, ' X=', X, ' ASIN(X) =', ASIN
00125 RETURN
00126 END

(FLECS VERSION 22.51)

00127 C
00128 FUNCTION ACOS(X)
00129 IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00130 ACOS = DATAN(1/SQRT(1-X*X))/X
00131 PRINT*, ' X=', X, ' ACOS(X) =', ACOS
00132 RETURN
00133 END

(FLECS VERSION 22.51)

00134 C
00135 FUNCTION EXP(X)
00136 IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00137 EXP = DEXP(X)
00138 PRINT*, ' X=', X, ' EXP(X) =', EXP
00139 RETURN
00140 END

(FLECS VERSION 22.51)

00141 C
00142 FUNCTION SORT(X)

B-17
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00143  SOR1 = DSORT(x)
00144  PRINT *, X=*, X, SOR1 = *, SOR1
00145  RETURN
00146  END

(FLECS VERSION 27.51)

FUNCTION ATAN(X)
00148  X
00149  IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00150  ATAN = ATAN(X)
00151  PRINT *, X=*, X, ATAN = *, ATAN
00152  RETURN
00153  END

(FLECS VERSION 27.51)

FUNCTION ALG(X)
00155  X
00156  IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00157  ALG = ALOG(X)
00158  PRINT *, X=*, X, ALOG = *, ALOG
00159  RETURN
00160  END

(FLECS VERSION 27.51)

FUNCTION COS(X)
00162  X
00163  IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00164  COS = COS(X)
00165  PRINT *, X=*, X, COS = *, COS
00166  RETURN
00167  END

(FLECS VERSION 27.51)

FUNCTION SIN(X)
00169  X
00170  IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00171  COMMON A, X
00172  A = -1D0 - THE_ T
00173  X = 1D0 - COS(X)
00174  RETURN
00175  END

(FLECS VERSION 27.51)

FUNCTION TAN(X)
00177  X
00178  B-18
FUNCTION PSI(A, THETA)

IMPLICIT DOUBLE PRECISION(A-H,Z)

COMMON N, A
COMMON COS, SIN

COS = COS(THETA)
SIN = SIN(THETA)

RETURN

END

END

FUNCTION PSI(A, THETA)

IMPLICIT DOUBLE PRECISION(A-H,Z)

DIMENSION M, N, A

CALL PSI(M, N, A)

RETURN

END

END

END

SUBROUTINE FUNCTION

DIMENSION M, N, A

CALL PSI(M, N, A)

RETURN

END

END

END

FILE: PROG 1011
FUNCTION TAN(THETA)
COMMON N, IN
EXTERNAL TAN

IF (N.NE.0) RETURN

TAN = TAN(THETA)
RETURN
END

FUNCTION ATAN(TAN)
EXTERNAL TAN
COMMON N, IN

IF (N.NE.0) RETURN

ATAN = ATAN(TAN)
RETURN
END

FUNCTION ASIN(THETA)
EXTERNAL ASIN
COMMON N, IN

COST = COS(THETA)
COSN = COS(N*THETA)
FASI = COS(THETA)
IF (COST.EQ.0) RETURN
FASI = (ASIN(K*COST)/K) + COSN/COST
RETURN
END
C FUNCTION FACOST(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ACOS
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FACOST = ACOS(AK*COST)*COSNT
RETURN
END

C FUNCTION RASIN(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASIN
RASIN = 1/ASIN(X)
PRINT*, " X =", X, " \n RASIN(X) =", RASIN
RETURN
END

C FUNCTION FRASINT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASIN
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FRASINT = COSNT
IF (COST.EQ.0) RETURN
FRASINT = COST*RASIN(AK*COST)*AK*COSNT
RETURN
END

C FUNCTION FALNT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASIN
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FALNT = COSNT
IF (COST.EQ.0) RETURN
RETURN
END
FUNCTION ALN(Z)
IMPLICIT DOUBLE PRECISION(A-H-P-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COMMON N, AK, I, AL
COSN = DCOS(N#THETA)
FSORT = DSORT((AK + AL + (AK - AL)*COST)/2)*COST
RETURN
END

FUNCTION FLOGI(IHL1tA)
IMPLICIT DOUBLE PRECISION(A-H-P-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COMMON N, AK, I, AL
COSN = DCOS(N#THETA)
FLOGI = DLOG((IHL1tA + AL + (AK - AL)*COST)/2)*COST
RETURN
END

FUNCTION FSORT(THETA)
IMPLICIT DOUBLE PRECISION(A-H-P-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COMMON N, AK, I, AL
COSN = DCOS(N#THETA)
FSORT = DSORT((AK + AL + (AK - AL)*COST)/2)*COST
RETURN
END

FUNCTION FEXPT(THETA)
IMPLICIT DOUBLE PRECISION(A-H-P-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COMMON N, AK, I, AL
COSN = DCOS(N#THETA)
FEXPT = DEXP((AK + AL + (AK - AL)*COST)/2)*COST
RETURN
END

B-22
C
FUNCTION FAASNT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASIN
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FAASNT = ASIN((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END

(FLECS VERSION 22.51)
-------------------------------------------------------------

C
FUNCTION EX1X(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXIX = 1
IF (X.NE.0) EXIX = DEXP(X) - 1)/X
PRINT*, ' X =', X, ' EXIX(X) =', EXIX
RETURN
END

(FLECS VERSION 22.51)
-------------------------------------------------------------

C
FUNCTION FEXIXT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FEXIXT = EXIX((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END

(FLECS VERSION 22.51)
-------------------------------------------------------------

C
FUNCTION SOAC(Y)
NOTE . . THIS FUNCTION COMPUTES 1/2 OF ACOS((1-Y)**2/Y
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ACOS
SOAC = 1
IF (Y.EQ.0) RETURN
SOAC = ACOS((1 - Y)**2/(2*Y))
SOAC = DATAN(DSRT((Y*(2 - Y))/(1 - Y)**2/(2*Y))
PRINT*, ' Y = ', Y, ' SOAC(Y) = ', SOAC
RETURN
END

(FLECS VERSION 22.51)
-------------------------------------------------------------
FUNCTION FSQACT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
COMMON N, AK, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FSQACT = SQAC((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END

(FLECS VERSION 22.51)

FUNCTION SINH(X)
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
E = DEXP(X)
SINH = (E - 1/E)/2
PRINT*, ' X = ', X, ' SINH(X) = ', SINH
RETURN
END

(FLECS VERSION 22.51)

FUNCTION COSH(X)
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
E = DEXP(X)
COSH = (E + 1/E)/2
PRINT*, ' X = ', X, ' COSH(X) = ', COSH
RETURN
END

(FLECS VERSION 22.51)

FUNCTION FSINHT(THETA)
EXTERNAL SINH
COMMON N, AK
COS = DCOS(THETA)
COSNT = DCOS(N*THETA)
FSINHT = COSNT
IF (COST .NE. 0) RETURN
FSINHT = (SINH(AK*COST*AK)/COSNT*COST)
RETURN
END

(FLECS VERSION 22.51)

FUNCTION FCOSH(THETA)
EXTERNAL COSH
COMMON N, AK

COST = DCOS(THETA)
COSTNT = DCOS(N*THETA)
FCOSHT = COSH(AK*COST)*COSTNT
RETURN
END

(FLECS VERSION 22.51)

C
FUNCTION ASINH(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ATANH
ASINH = 0.000
IF (X .EQ. 0) RETURN
ASINH = ATANH(X/DSGRT(1 + X**2))
PRINT*, 'X = ', X, ' ASINH(X) = ', ASINH
RETURN
END

(FLECS VERSION 22.51)

C
FUNCTION FASINH(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASINH
COMMON N, AK
COST = DCOS(THETA)
COSTNT = DCOS(N*THETA)
FASINH = COSNT
IF (COST EQ 0) RETURN
FASINH = (ASINH(AK*COST)*AK*COSHT COST)
RETURN
END

(FLECS VERSION 22.51)

C
FUNCTION ACOSH(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ATANH
ACOSH = ATANH(DSGRT(X**2-1)/X)
PRINT*, 'X = ', X, ' ACOSH(X) = ', ACOSH
RETURN
END

(FLECS VERSION 22.51)

C
FUNCTION FACOSH(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ACOSH
COMMON N, AK
COST = DCOS(THETA)
00471  COSNT = DCOS(N*THETA)
00472  FACOSHT = ACOSH(AK*COST)*COSNT
00473  RETURN
00474  END

(FLECS VERSION 22.51)

00475 C
00476  FUNCTION ATANH(Y)
00477  IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00478  ATANH = 0.0DO
00479  IF (-X .EQ. 0) RETURN
00480  ATANH = LOGC(1+X)/(1-X))/2
00481  PRINT*, " X =", X, " ATANH(X) =", ATANH
00482  RETURN
00483  END

(FLECS VERSION 22.51)

00484 C
00485  FUNCTION ATANHHT(HETA)
00486  IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00487  EXTERNAL ATANH
00488  COMMON N, AK
00489  COST = DCOS(THETA)
00490  COSNT = DCOS(N*THETA)
00491  ATANHHT = COSNT
00492  IF (COST.EQ.0) RETURN
00493  ATANHHT = (ATANH(AK*COST)/AK)*(COSNT/COST)
00494  RETURN
00495  END

(FLECS VERSION 22.51)

00496 C
00497  FUNCTION TANH(X)
00498  IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00499  TANH = DTANH(X)
00500  PRINT*, " X =", X, " TANH(X) =", TANH
00501  RETURN
00502  END

(FLECS VERSION 22.51)

00503 C
00504  FUNCTION ATANHHT(HETA)
00505  IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00506  EXTERNAL TANH
00507  COMMON N, AK
00508  COST = DCOS(THETA)
00509  COSNT = DCOS(N*THETA)
00510  ATANHHT = COSNT
00511  IF (COST.EQ.0) RETURN
FTANHT = \( \frac{\tanh(\text{AK} \times \text{COST})}{\text{AK}} \) \* \( \frac{\text{COSHT}}{\text{COST}} \)

RETURN

END

(FLECS VERSION 22.51)

-----------------------------------

EDI.  0 FILES.  1 RECS.  2321 WORDS.
COPY,COMPI

FUNCTION ROMBRG(A, B, F, OE) 00008
IMPLICIT DOUBLE PRECISION(A-H,P-Z) 00009
INTEGER FOUR(21), 0 00011
DIMENSION Q(20), QO(2) 00012
EQUIVALENCE (QO(2), Q(1)) 00013
DATA Q/0/, LMAX/1/, FOUR(1)/4/
Q(0) = 0 00015
H = (B - A)/2 00017
IMAX = 1 00019
DO 99999 L=1,20 00021
IF(.NOT.(L.GE.LMAX)) GO TO 99998 00022
LMAX = L + 1 00023
FOUR(L+1) = 4*FOUR(L) 00024
99998 SUM = 0 00027
DO 99997 I=IMAX 00028
V = (2*I - IMAX - 1.0D0)/IMAX 00029
U = V*3 - V*V)/2 00030
UP = 1.5D0*(1 - V*V) 00031
X = A + (U + 1)*H 00032
SUM = SUM + F(X)*UP 00033
99997 CONTINUE 00034
SUM = SUM + IMAX*G(L-1) 00035
IMAX = 2*IMAX 00036
Q(L) = SUM/IMAX 00037
DO 99998 M=1,L 00039
K = L - M 00040
Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1) 00041
99998 CONTINUE IF(.NOT.(L.LE.0E.)) RETURN 00044
99999 CONTINUE 00047
RETURN 00048
END

FUNCTION ROMBEP(A, B, F, OE) 00051
IMPLICIT DOUBLE PRECISION(A-H,P-Z) 00052
INTEGER FOUR(21), 0 00054
DIMENSION Q(20), QO(2) 00056
DATA Q/0/, LMAX/1/, FOUR(1)/4/
Q(0) = (F(A) + F(B))/2 00070
H = B - A 00072
IMAX = 1 00074
DO 99999 L=1,20 00076
IF(.NOT.(L.GE.LMAX)) GO TO 99998 00077
LMAX = L + 1 00079
FOUR(141) = 4*FOUR(L) 00081
99998 H = H/2 00082
SUM = 0 00083
DO 99997 I=1,IMAX 00084
SUM = SUM + F(A + H*(2*I - 1)) 00084
99997 CONTINUE 00087
SUM = SUM + IMAX*G(L-1) 00089
IMAX = 2*IMAX 00090
Q(L) = SUM/IMAX 00091
DO 99998 M=1,L 00092
K = L - M 00094
Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1) 00096
B-28
CONTINUE

ROMBER = (B - A)*Q(0)
ODQ = DABS((Q(0) - Q(1))*(B - A))
IF (ODQ.LE.OE) RETURN

CONTINUE

FUNCTION SIN(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
SIN = DSIN(X)
RETURN
END

FUNCTION TAN(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
TAN = DSIN(X)/DCOS(X)
RETURN
END

FUNCTION ASIN(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
ASIN = ATAN(X/DSRT(1-XX))
RETURN
END

FUNCTION ACOS(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
ACOS = DATAN(DSRT(1-XX)/X)
RETURN
END

FUNCTION EXP(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXP = DEXP(X)
RETURN
END

FUNCTION SORT(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
SORT = DSORT(X)
RETURN
END

FUNCTION ATAN(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
ATAN = DAAN(X)
RETURN
END

FUNCTION ALOG(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
ALOG = DLOG(X)
RETURN
END

FUNCTION COS(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
COS = DCOS(X)
RETURN
END

FUNCTION FCOST(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
COMMON N, AK
COST = DCOS(THETA)
COSTNT = DCOS(N*THETA)
FCOST = DCOS(AK*COST)*COSTNT
RETURN
END

FUNCTION FSINT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(2*THETA)
FSINT = COSNT
IF (COST.EQ.0) RETURN
FSINT = (DSIN(AK*COST)/AK)*COSNT/COST
RETURN
END
FUNCTION POLY(X, A, N)
DIMENSION A(25)
PROD = A(N)
DO 99999 J=1,N
PROD = PROD*X + A(N-J)
99999 CONTINUE
POLY = PROD
RETURN
END
SUBROUTINE ACOEF(A, B, IC, AK, N)
DIMENSION A(25), B(25)
DO 99999 J=1,N
SUM = SUM + B(J)*IC(J+1)
99999 CONTINUE
A(J) = SUM/AK(J)
RETURN
END
FUNCTION FTANTHEI,
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00180
COMMON N, AK
00181
COST = DCOS(THETA)
00182
COSNT = DCOS(2*THETA)
00183
FSINT = COSNT
00184
IF (COST.EQ.0) RETURN
00185
FSINT = (DSIN(AK*COST)/AK)*COSNT/COST
00186
RETURN
00188
END
FUNCTION POLY(X, A, N)
DIMENSION A(25)
00191
IMPLICIT DOUBLE PRECISION (A-HP-Z) 00192
DIMENSION A(25)
00193
PROD = A(N)
00194
DO 99999 J=1,N
00195
PROD = PROD*X + A(N-J)
99999 CONTINUE
POLY = PROD
RETURN
END
SUBROUTINE ACOEF(A, B, IC, AK, N)
DIMENSION A(25), B(25)
DO 99999 J=1,N
SUM = SUM + B(J)*IC(J+1)
99999 CONTINUE
A(J) = SUM/AK(J)
RETURN
END
FUNCTION FTANTHEI,
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FANT = COSNT
IF (COST.LE.0) RETURN
FANT = (TAN(AK*COST)/AK)*(COSNT/COST)
RETURN
END

FUNCTION FANT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FANT = COSNT
IF (COST.EQ.0) RETURN
FANT = (TAN(AK*COST)/AK)*(COSNT/COST)
RETURN
END

FUNCTION FACOST(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ACOS
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FACOST = ACOS(AK*COST)*COSNT
RETURN
END

FUNCTION RASIN(X)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ASIN
RASIN = 1/ASIN(X)
RETURN
END

FUNCTION FRASINT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL RASIN
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FRASINT = COSNT
IF (COST.EQ.0) RETURN
FRASINT = COST*RASIN(AK*COST)*AK*COSNT
RETURN
END

FUNCTION FALNT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,P-Z)
EXTERNAL ALN
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FINT = COSNT
IF (COST.EQ.0) RETURN
FINT = (ALN(AK*COST)/AK)*(COSNT/COST)
RETURN
END
FUNCTION ALN(Z)
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
ALN = DLOG((Z + 1)/(1 - Z))/2
RETURN
END
FUNCTION FLOGT(THETA)
IMPLICIT DOUBLE PRECISION (A-H,F-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FLOGT = DLOG((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END
FUNCTION FSORTT(THETA)
IMPLICIT DOUBLE PRECISION (A-H,F-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FSORTT = DSORT((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END
FUNCTION FFPT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FFPT = DEXP((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END
FUNCTION FAASNT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
EXTERNAL ASIN
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FAASNT = ASIN((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END
FUNCTION EXIX(X)
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
EXIX = 1
IF (X.NE.0) EXIX = (DEXP(X) - 1)/X
RETURN
END
FUNCTION FEXIXT(THETA)
IMPLICIT DOUBLE PRECISION (A-H,F-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FEXIXT = EXIX((AK + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END
FUNCTION SOAC(Y)
IMPLICIT DOUBLE PRECISION(A-H,F-Z)
SOAC = 1
IF (Y.EQ.0) RETURN

B-32
```fortran
SOAC = DATAN(DSQR(Y*(2 - Y))/(1 - Y)**2/(2*Y))
RETURN
END

FUNCTION FSOACT(THETA)
IMPLICIT DOUBLE PRECISION(A-H,F-P-Z)
COMMON N, AK, I, AL
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FSOACT = (2*SOAC(THETA + AL + (AK - AL)*COST)/2)*COSNT
RETURN
END

FUNCTION SINC(X)
IMPLICIT DOUBLE PRECISION(A-H,F-P-Z)
E = DEXP(X)
SINC = E - 1/E + 1/2
RETURN
END

FUNCTION COSC(X)
IMPLICIT DOUBLE PRECISION(A-H,F-P-Z)
COMMON N, AK
COST = DCOS(THETA)
COSNT = DCOS(N*THETA)
FCOSCT = COSC(THETA + AL)*COST)*COSNT
RETURN
END

FUNCTION FASINC(X)
IMPLICIT DOUBLE PRECISION(A-H,F-P-Z)
EXTERNAL ATANH
ASINC = 0.0E0
IF (X.EQ.0) RETURN
ASINC = ATANH(X/DSQR(1 + X**X))
RETURN
END

FUNCTION FASNC(X)
IMPLICIT DOUBLE PRECISION(A-H,F-P-Z)
EXTERNAL ASINH
FASNC = ASINH(THETA + AL)*COST)*COSNT
RETURN
END

FUNCTION FASNL(THETA)
IMPLICIT DOUBLE PRECISION(A-H,F-P-Z)
EXTERNAL ANH
FASNL = ASINH(THETA + AL)*COST)*COSNT
RETURN
END
```

B-33
FUNCTION ACOSH(X)
DOUBLE PRECISION(A-H+F-Z)
EXTERNAL ATAN
ACOSH = ATANH(COSH(x)) / x
RETURN
END

FUNCTION FATANH(THETA)
DOUBLE PRECISION(A-H+F-Z)
EXTERNAL ATAN
THETA + X 0.0
RETURN
END

FUNCTION FATANHTHETA
DOUBLE PRECISION(A-H+F-Z)
EXTERNAL ATAN
THETA 0.0
RETURN
END

END
MISSION
of
Rome Air Development Center

RAOC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.