Implementing the Complex Arcsine and Arccosine Functions Using Exception Handling

T. E. HULL
University of Toronto
THOMAS F. FAIRGRIEVE
Ryerson Polytechnic University
and
PING TAK PETER TANG
Lawrence Berkeley National Laboratory

We develop efficient algorithms for reliable and accurate evaluations of the complex arcsine and arccosine functions. A tight error bound is derived for each algorithm; the results are valid for all machine-representable points in the complex plane. The algorithms are presented in a pseudocode that has a convenient exception-handling facility. Corresponding Fortran 77 programs for an IEEE environment have also been developed to illustrate the practicality of the algorithms, and these programs have been tested very carefully to help confirm the correctness of the algorithms and their error bounds. The results of these tests are included in the article, but the Fortran 77 programs are not (these programs are available from Fairgrieve). Tests of other widely available programs fail at many points in the complex plane, and otherwise are slower and produce much less accurate results.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*error analysis*; *Numerical algorithms*; G.1.2 [**Numerical Analysis**]: Approximation—*elementary function approximation*; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis*; *reliability and robustness*; *verification*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Complex elementary functions, implementation

This work was supported by the Natural Sciences and Engineering Research Council of Canada, by the Information Technology Research Centre of Ontario, and by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under contract W-31–109–Eng-38, and by the STARS Program Office, under AF Order RESD-632. The work of T. Fairgrieve was performed while at the University of Toronto. Professor Emeritus T. E. Hull of the Department of Computer Science, University of Toronto, died on August 15, 1996.

Authors' addresses: T. F. Fairgrieve, Department of Mathematics, Physics and Computer Science, Ryerson Polytechnic University, 350 Victoria Street, Toronto, Ontario M5B 2K3, Canada; email: tfairgri@scs.ryerson.ca; P. T. P. Tang, National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory, Berkeley, CA 94720.

1. INTRODUCTION

Our purpose is to develop efficient algorithms, along with error bounds, for reliable and accurate evaluations of the complex arcsine and arccosine functions. These functions can be expressed mathematically in terms of formulas involving only real arithmetic and real elementary functions; complex arithmetic is not needed. However, serious cancellation or instability can arise during a straightforward evaluation of these formulas. But, with care, they can be rearranged so that such difficulties do not arise.

The remaining difficulties in such evaluations are that overflow or underflow might occur. Such exceptions are always "spurious" for the two functions considered in this article, in the sense that the final mathematical result can be accurately approximated by a machine-representable complex number. In these circumstances, the algorithms must provide for alternative calculations which are able to circumvent the spurious exceptional situations.

Section 2 provides a summary of the basic definitions and assumptions, especially with regard to the error analysis, that are used for the development of the algorithms in Sections 3 and 4. The algorithms are expressed in terms of a pseudocode with a convenient exception-handling facility.

Extensive testing of the algorithms, with Fortran 77 [ANSI 1978] versions of the programs running on a machine using IEEE binary arithmetic [IEEE 1985], is described in Section 5. The results help to confirm the correctness of the error bounds and show that the bounds are quite tight. Corresponding results for other available programs are also discussed. Then it is shown in Section 6 how the programs can be extended to be valid even for the rare case of a system which fails to satisfy one of our assumptions because its exponent range is too narrow. Concluding remarks are given in Section 7.

We assume throughout that the complex argument z of each of the two functions is exact. If this is not the case in a particular application, the additional error must be accounted for separately.

2. DEFINITIONS AND ASSUMPTIONS

The purpose of this section is primarily to introduce the basic material needed in carrying out error analyses of the algorithms and to specify the assumptions we make about the computing environment in which the algorithms can be successfully implemented. A brief explanation of the exception-handling facility is also included. Some of this material was presented in our earlier article on complex elementary functions [Hull et al. 1994a; 1994b], but it is extended here, especially in developing the technique for analyzing errors.

2.1 Analyzing Errors

We assume that all input and output values of the two functions dealt with in this article are normalized *complex* floating-point numbers, including numbers with one or both components equal to 0. Once the real and

imaginary parts of these arguments have been identified at the beginning of each algorithm, it is arranged that all subsequent operations will operate only on normalized *real* floating-point numbers.

The main assumption we make about the real arithmetic is that, if x and y are normalized real floating-point numbers, and op is one of the four basic arithmetic operations, then there is a relative roundoff error bound E such that

$$fl(x \text{ op } y) = (x \text{ op } y)(1 + \epsilon),$$

for some ϵ where $|\epsilon| \leq E$, provided no exception occurs. Here $fl(x \text{ op } y)$ is a floating-point approximation to x op y produced by the machine. We also assume that $fl(x \text{ op } y) = x \text{ op } y$ whenever x op y is machine-representable.

And we assume that corresponding error bounds exist for the machine implementations of the real elementary functions. For example, we assume there is a bound $E_{\log}$ such that

$$fl(\log(x)) = \log(x)(1 + \epsilon_{\log}),$$

for some $\epsilon_{\log}$ where $|\epsilon_{\log}| \leq E_{\log}$. $E_{\log}$ should be at most some small multiple of E.

With these assumptions we can derive expressions for errors and error bounds for the calculations that arise in the next two sections. For example, we can conclude that

$$fl(xy \log(x)) = xy \log(x)(1 + \epsilon)(1 + \epsilon)(1 + \epsilon_{\log}),$$

so that a relative error bound for the calculation is $(1 + E)^2(1 + E_{\log}) - 1$. Note that the different occurrences of ϵ do not generally represent the same value; each is simply some quantity that is bounded in magnitude by E.

For practical purposes, results like these need to be simplified. The relative error in evaluating $xy\log(x)$ is

$$2\epsilon + \epsilon_{\log}$$

if we neglect terms that are small multiples of products of ϵ terms. And a bound for the error is

$$2E + E_{\log}$$

if we neglect small multiples of products of E terms.

It is convenient to introduce an ϵ-notation, where ϵ_a is the relative error in evaluating the expression a, neglecting small multiples of products of ϵ terms (such as ϵ and $\epsilon_{\log}$). With this understanding, we can then write

$$\epsilon_{xy\,\log(x)} = 2\epsilon + \epsilon_{\log}$$

when we mean that the relative error in evaluating $xy\log(x)$ is $2\epsilon + \epsilon_{\log}$. In the next two sections the small multiples are usually less than 100 or so. The approximations we use are therefore good enough for practical purposes, since we know that the E terms (E, $E_{\log}$, etc.) are extremely small; for single precision in IEEE binary arithmetic [IEEE 1985] $E = 2^{-24}$. Of course the notation can be used only when no exceptions occur.

With the ϵ-notation, we not only have obvious results such as $3\epsilon + \epsilon = 4\epsilon$, but also not quite so obvious results such as $3\epsilon - \epsilon = 4\epsilon$. We will also write $\epsilon_a < \epsilon_b$ if $E_a < E_b$, where a and b are expressions, and similarly for $\leq$, $>$, and $\geq$.

Here is another example, which is typical of what occurs quite a few times in the next two sections. Consider determining the value of

$$\epsilon_{\sqrt{(x+1)^2+y^2}},$$

where $x \geq 0$. We begin with

$$\epsilon_{x+1} = \epsilon,$$

and continue with

$$\epsilon_{(x+1)^2} = 3\epsilon$$

and

$$\epsilon_{y^2} = \epsilon,$$

so that

$$\epsilon_{(x+1)^2+y^2} = 4\epsilon.$$

(This last step follows from the "max" rule that if a and b are of the same sign, then $\epsilon_{a+b} = \max(\epsilon_a, \epsilon_b) + \epsilon$, which in turn follows from the general rule for sums given below.) Then, if we also use the rule for functions given below, we end up with

$$\epsilon_{\sqrt{(x+1)^2+y^2}} = 2\epsilon + \epsilon_{\text{sqrt}},$$

since the *error propagation factor* (epf) for sqrt (as defined below) is 1/2.

The general rule for handling *sums* and *differences* is

$$\epsilon_{a+b} = \frac{a}{a+b}\epsilon_a + \frac{b}{a+b}\epsilon_b + \epsilon,$$

where a and b are expressions and provided that $a + b \neq 0$. The rule for handling sums can be strengthened if we know something about the relative magnitudes of a and b. For example, if a and b are of the same sign, and $|a| \geq |b|$ but $\epsilon_a \leq \epsilon_b$, we can derive

$$\epsilon_{a+b} = 0.5(\epsilon_a + \epsilon_b) + \epsilon,$$

which cannot be greater than, but may be smaller than, what was provided by the general rule; we take advantage of this "average" rule once, in the detailed error analysis of Section 3.4. Otherwise, the simpler "max" rule used in the preceding paragraph turns out to be all that is needed in most of the analysis of Section 3.4.

The general rule for *products* and *quotients* is simply

$$\epsilon_{a\times b} = \epsilon_{a/b} = \epsilon_a + \epsilon_b + \epsilon.$$

The rule required when *functions* are involved is

$$\epsilon_{f(a)} = \mathrm{epf}(f(a))\epsilon_a + \epsilon_f = \frac{af'(a)}{f(a)}\epsilon_a + \epsilon_f,$$

where ϵ_f is the relative error in evaluating f, and provided of course that ϵ_a is small enough. For an error bound in such a situation we need a bound for the error propagation factor for f, $\mathrm{epf}(f(a))$, which needs to be determined over the range of values of a that can occur in the circumstances being analyzed.

It is convenient to introduce an "approximately equal" $\approx$ notation, defined as follows:

$$A \approx \alpha \text{ if } A = \alpha(1 + s\epsilon),$$

where $s\epsilon$ is some small multiple of ϵ. It is helpful to use this notation in motivating the development of algorithms in the next two sections, leaving the actual quantification of $s\epsilon$ to the associated error analyses. We also use $<\approx$ (for "less than or 'approximately equal' to") and $>\approx$ (for "greater than or 'approximately equal' to"). And to distinguish mathematical numbers from machine-representable approximations, we use lowercase Greek and Roman letters for the former, and we substitute a single uppercase Roman letter to indicate the latter. Thus, in the above, A is an approximation to α; and later on, for example, we let $Am1$ denote the machine approximation to the mathematical value $\alpha m1$ (where $\alpha m1$ is used to denote the expression $\alpha - 1$), and we let Sqrt stand for the machine approximation to the true square root function sqrt.

2.2 The Overall Relative Error Bound

The technique described in the Section 2.1 is used to find the errors, ϵ_r and ϵ_i, and then the error bounds, E_r and E_i, for the calculated values of the real and imaginary parts, f_r and f_i, of the complex functions f considered in the next two sections. If F is the calculated value of f, and F_r and F_i are the calculated values of its real and imaginary parts, the magnitude of the overall relative error in F is

$$\left|\frac{F-f}{f}\right| = \left|\frac{\frac{F_r-f_r}{f_r}f_r + i\frac{F_i-f_i}{f_i}f_i}{f_r+if_i}\right|$$

$$= \sqrt{\frac{\epsilon_r^2 f_r^2 + \epsilon_i^2 f_i^2}{f_r^2+f_i^2}}$$

$$\leq \max(E_r, E_i),$$

assuming $f \neq 0$.

However, it can happen that the relative error in one part is very large, while the corresponding absolute error is so small in comparison with the absolute error in the other part that it can be neglected with only a negligible effect on the overall relative error bound. There are three such instances in the programs that follow in which the absolute error $|\epsilon_r f_r|$ can be neglected in comparison with $|\epsilon_i f_i|$, as will be pointed out. The overall relative error bound is then simply

$$\left|\frac{F-f}{f}\right| \leq E_i,$$

assuming $f \neq 0$.

These results will be used in the next two sections, and the resulting error bounds will be tabulated in Section 5 along with the corresponding test results for comparison.

2.3 The Pseudocode and Exception Handling

The exception-handling construct used in Sections 3 and 4 is shown in Figure 1. The calculations in the enable block normally produce the required result, but if an exception occurs in the course of these calculations, control is transferred to the handle block, or handler, where action is taken to circumvent or otherwise cope with the situation. The transfer of control can take place at any time after the exception has occurred, up to the end of the enable block.

We do not assume that any indication of what exceptions occurred is available to the handler. Exception-handling constructs can be nested

```
enable
    - - - -
    - - - -
handle
    - - - -
    - - - -
end
```

Fig. 1. The exception-handling construct. The enable block normally produces the desired result, but the handle block is executed if an exception occurs in the enable block.

within handlers. Otherwise we believe our pseudocode is reasonably self-explanatory.

If no such exception-handling facility is available, the programs can be modified to produce comparably accurate results, but in a more complicated and slower way, as indicated later in Section 7.

In a previous article [Hull et al. 1994a; 1994b] we described the conventions we have adopted about what to do if either the real or the imaginary part of the result should overflow or underflow. The only such situation that can occur in the two examples considered in this article is that the real part can underflow. However, this can happen only when the imaginary part is so large that the real part can be replaced by zero without affecting the overall relative error bound by any significant amount, as indicated in the previous subsection.

2.4 The Computing Environment

We assume that the smallest and largest positive normalized floating-point numbers, represented by u and M respectively, satisfy $1 / 257 < uM < 4$ (of course $u \ll 1$ and $M \gg 1$). The product uM is just less than 4 for IEEE binary arithmetic, just less than 1/2 for VAX arithmetic, and just less than 1/256 for IBM 360/370 arithmetic. (The assumptions about uM are satisfied by all the systems tested by Cody [1988, p. 309], except for the Cyber 180/855; the *xmax* and *xmin* that he gives for IEEE arithmetic have been rounded to three figures, and their product is greater than 4; but without the rounding their product would be just less than 4.) We also assume that $E^2 \geq 4u$ and $E^2 \geq M^{-1}$. This is easily satisfied by all the systems tested by Cody.

Throughout most of this article we also require that $E^2 > 4\sqrt{u}$ (rounding arithmetic) or $E^2 > 8\sqrt{u}$ (chopping arithmetic). This would rule out VAX D arithmetic, for example, because its exponent range is too narrow. But we show, later on (in Section 6), how to cope if this requirement is not satisfied.

In Sections 3 and 4 we need the function

$$\text{sign}(x) = \begin{cases} +1, & x \geq 0, \\ -1, & x < 0. \end{cases}$$

We also need the real elementary functions arcsin, arccos, arctan, log, and log1p, where $\text{log1p}(x) = \log(1 + x)$. (If the log1p function is not available in the system being used, it can be created quite easily with the help of the log function; for details, see Hull et al. [1994a; 1994b].)

Binary arithmetic is assumed throughout. If this is not the case the theoretical error bounds would be increased by an E or so. It is also assumed that computations are carried out in the order prescribed by parentheses. If this is not the case it is possible that serious cancellation can occur so that a useful error analysis would be impossible; but, even when no serious cancellation occurs, the error bounds would be larger than what we derive.

Certain constants are used in the programs, namely $Foursqrtu$, $Log2$, $Piby2$, and Pi. We assume they approximate $4\sqrt{u}$, $\log(2)$, $\pi / 2$, and π, respectively, as accurately as possible, that is, to within a relative error bounded by E.

3. COMPLEX ARCSINE FUNCTION CARCSIN

In this section we present and analyze an algorithm in the form of a pseudocode program for the complex arcsine function. The enable block is developed in Section 3.1. A safe region in the complex plane is identified in Section 3.2; this is a region in which we are able to prove that the enable block encounters no difficulties with overflow or underflow. The handle block is developed, for regions outside the safe region, in Section 3.3. And the error analysis is carried out in Section 3.4.

3.1 Enable Block

The complex inverse sine function $\arcsin(z)$, where $z = x + iy$, can be defined by

$$\arcsin(z) = \arcsin(\beta) + i\ \text{sign}(y)\log(\alpha + \sqrt{\alpha^2 - 1}),$$

where

$$\alpha = 0.5(\sqrt{(x + 1)^2 + y^2} + \sqrt{(x - 1)^2 + y^2}),$$

$$\beta = 0.5(\sqrt{(x + 1)^2 + y^2} - \sqrt{(x - 1)^2 + y^2}),$$

and where the log function is the natural log. The branch cuts are on the real line from $-\infty$ to -1 and from 1 to ∞. The real and imaginary parts are based on those given by Abramowitz and Stegun [1972, pp. 80–81], but we have chosen the sign of the imaginary part to provide what is generally considered to be the principal value of this function. We note that α is the

```
function CARCSIN ( z : complex ) : complex
    real x, y, R, S, A, Am1, B
    real E, Foursqrtu, Log2, Piby2, Acrossover, Bcrossover ← appropriately initialized
    complex Answer
    x := abs(z.realpart)      y := abs(z.imagpart)
    enable
        R := Sqrt((x + 1) * *2 + y * *2)      S := Sqrt((x − 1) * *2 + y * *2)
        A := .5 * (R + S)      B = x/A
        if B ≤ Bcrossover then
            Answer.realpart := Arcsin(B)
        else  - - use arctan and an accurate approximation to α − x
            if x ≤ 1 then
                Answer.realpart :=
                    Arctan(x/Sqrt(.5 * (A + x) * ((y * *2)/(R + (x + 1)) + (S + (1 − x)))))
            else
                Answer.realpart :=
                    Arctan(x/(y * Sqrt(.5 * ((A + x)/(R + (x + 1)) + (A + x)/(S + (x − 1))))))
            endif
        endif
        if A ≤ Acrossover then  - - use log1p and an accurate approximation to α − 1
            if x < 1 then
                Am1 := .5 * ((y * *2)/(R + (x + 1)) + (y * *2)/(S + (1 − x)))
            else
                Am1 := .5 * ((y * *2)/(R + (x + 1)) + (S + (x − 1)))
            endif
            Answer.imagpart := Log1p(Am1 + Sqrt(Am1 * (A + 1)))
        else
            Answer.imagpart := Log(A + Sqrt(A * *2 − 1))
        endif
    handle
        - -
        - - (see handle block in Figure 4)
        - -
    end
    Answer.realpart := sign(z.realpart) * Answer.realpart
    Answer.imagpart := sign(z.imagpart) * Answer.imagpart
    return Answer
end CARCSIN
```

Fig. 2. A program for calculating an approximation to the complex arcsine function. We use $Acrossover = 1.5$ and $Bcrossover = 0.6417$, but these values could be adjusted to give a slightly better theoretical error bound, depending on the system being used.

average of the distances from z to the points (1,0) and (–1,0) in the complex z-plane, and in particular that $\alpha \geq 1$. Also β is in [–1,1].

We also note that the sign of the real part is the same as the sign of x. This fact, along with the rule for determining the sign of the imaginary part, is used to determine the signs of the real and imaginary parts at the end of the program in Figure 2. Then the calculations in the program need be done only for values of x and y which are nonnegative; x and y are therefore redefined to be $|x|$ and $|y|$ as the first step in carrying out the calculations. We assume that x and y are nonnegative in the remaining text of this section.

Then we try, in an enable block, to evaluate approximations to α and β, and the real and imaginary parts of the answer, in as straightforward a manner as possible, but ignoring any possible overflows or underflows; we leave such exceptions to be taken care of in the handler. However, we cannot use the formulas exactly as they stand, since there lurk within them major pitfalls, which do not themselves raise exceptions, but which could cause serious loss of accuracy if they are not avoided.

We begin the enable block by first determining approximations to

$$\rho = \sqrt{(x + 1)^2 + y^2}$$

and

$$\sigma = \sqrt{(x - 1)^2 + y^2}.$$

The approximations are denoted by R and S in the program; they are needed later on. Then an approximation to

$$\alpha = 0.5(\rho + \sigma)$$

is determined; it is denoted by A in the program. Then we approximate β. The original formula for β is $0.5(\rho - \sigma)$, but, as it stands, this expression contains a major pitfall, which is the possibility of serious cancellation error. However, we can get around this difficulty by considering instead

$$\beta = 0.5(\rho^2 - \sigma^2) / (\rho + \sigma),$$

which gives us

$$\beta = x / \alpha,$$

and its approximation is denoted by B in the program.

An approximation to the real part of the answer can then be obtained immediately, except for one more pitfall. This pitfall arises when B is near 1, for then even a rounding error or two in B can induce very large errors in the approximation to $\arcsin(\beta)$. A way around this difficulty is to determine an accurate approximation to $\alpha - x$ and then use

$$\arctan(\beta / \sqrt{1 - \beta^2}) = \arctan(x / \sqrt{(\alpha + x)(\alpha - x)})$$

in place of $\arcsin(\beta)$. To develop a formula for $\alpha - x$, we write

$$\begin{aligned} \alpha - x &= 0.5(\rho + \sigma) - x \\ &= 0.5(\rho - (x + 1) + \sigma + (1 - x)) \end{aligned}$$

$$= 0.5\left\{\begin{array}{l} \dfrac{y^2}{\rho + (x+1)} + \sigma + (1-x), x \le 1 \\ y^2\left(\dfrac{1}{\rho + (x+1)} + \dfrac{1}{\sigma + (x-1)}\right), x > 1 \end{array}\right\}.$$

Because of the way in which the above has been rewritten, and because of the parentheses around $1 - x$ and $x - 1$, there can be no serious loss of accuracy due to cancellations in this formula. The extra parentheses surrounding $S + (1 - x)$ in the program when $x \le 1$ and the extra parentheses surrounding $x + 1$ in the above formula make the error bound in Section 3.4 smaller than it would otherwise have been. When substituting into $\arctan(x / \sqrt{(\alpha + x)(\alpha - x)})$ it is important in the case when $x > 1$ to replace the y^2 factor inside the square root with a y factor outside the square root to reduce the risk of underflow or overflow.

These calculations are required only if B is too close to 1, specifically when $B > Bcrossover$. We have chosen 0.6417 as the crossover value of B because in carrying out the (later) error analysis we discovered that beyond about this point the extra error introduced by the complexity of $\arctan(\beta / \sqrt{1 - \beta^2})$ becomes less than the extra error introduced by the sensitivity of $\arcsin(\beta)$ to small errors in β.

We turn now to the evaluation of the imaginary part, and we encounter one more pitfall, this time when the argument of the log function is near 1. If it is near 1, even if it itself is quite accurate, the value of its log may be very inaccurate. A way around this difficulty is to determine an accurate approximation to $\alpha m1 = \alpha - 1$ and then use $\mathrm{log1p}(\alpha m1 + \sqrt{\alpha m1(\alpha + 1)})$ in place of $\log(\alpha + \sqrt{\alpha^2 - 1})$. If $\alpha m1$ is accurate, the argument of log1p will be accurate and so will the resulting value of log1p. Our technique for approximating $\alpha m1$ accurately is to rewrite the formula for $\alpha m1$ as follows:

$$\begin{aligned} \alpha m1 &= 0.5(\rho + \sigma) - 1 \\ &= 0.5(\rho - (x+1) + \sigma + (x-1)) \\ &= 0.5\left\{\begin{array}{l} \dfrac{y^2}{\rho + (x+1)} + \dfrac{y^2}{\sigma + (1-x)}, x < 1 \\ \dfrac{y^2}{\rho + (x+1)} + \sigma + (x-1), x \ge 1 \end{array}\right\}. \end{aligned}$$

Its approximation is denoted by $Am1$ in the program. There can be no loss of accuracy due to cancellations in these formulas. (In the program, extra parentheses surround $S + (x - 1)$ when $x \ge 1$; otherwise the error bound derived in Section 3.4 would be larger.)

Since the calculation of $Am1$ in this situation is more costly than the calculation of A, we do it only when A is near 1, specifically when $A \leq Acrossover$. We have chosen 1.5 as the crossover value of A because in carrying out the (later) error analysis we discovered that the overall error bound would have to be increased if any cut-off lower than 1.5 is used.

The enable block of the program in Figure 2 is based in a straightforward manner on the formulas developed so far. The formulas are more complicated than those used at the beginning of this section to provide the mathematical specification of the complex arcsine function, but this is in order to avoid possible losses of accuracy in the real part due to cancellation, or to the sensitivity of $\arcsin(\beta)$ to errors in β when $B > Bcrossover$, or to avoid possible loss of accuracy in the imaginary part when the argument of the log function is near 1. Our later error analysis will show that the computed results can be guaranteed to be within acceptably small error bounds.

The remaining computational problems have to do with possible overflows or underflows. There are many opportunities for such exceptions to arise. Before considering the details of a handler for coping with these exceptional cases it is helpful to identify a "safe" region.

3.2 A Safe Region

The region of the complex plane, which we identify as "safe," is to be safe in the sense that we are able to prove that the enable block produces a function value for each point in the region without raising any overflow or underflow exception. It will be very useful in the next two subsections to have identified such a region.

The part of our safe region in the first quadrant is the square in Figure 3 defined by $4\sqrt{u} \leq x, y \leq \sqrt{M}/8$, where u and M are the underflow and overflow levels respectively. Our assumptions about u and M were given earlier, in Section 2.4.

The scale of the diagram in Figure 3 is severely distorted. This is necessary in order to highlight the different circumstances that must be taken into account in designing the handler in the next subsection. The dashed lines in Figure 3 indicate where the lines $x = 4\sqrt{u}$ and $y = 4\sqrt{u}$ could be for a "bad" machine, a "bad" machine being one in which the exponent range is unreasonably small, as is the case with the VAX D format, for example. As indicated in Figure 3, a portion of the complex plane in the first quadrant (the inverted triangular region centered on the line $x = 1$), other than what is on the line $x = 1$, can then be *outside* the safe region, i.e., below the line $y = 4\sqrt{u}$ *and* above the lines $y = E|x - 1|$. Thus a machine is "bad" if $E^2 < 4\sqrt{u}$ (rounding arithmetic) or $E^2 < 8\sqrt{u}$ (chopping arithmetic), as mentioned earlier in Section 2.4. Special provision must be made for this possibility, but we postpone to

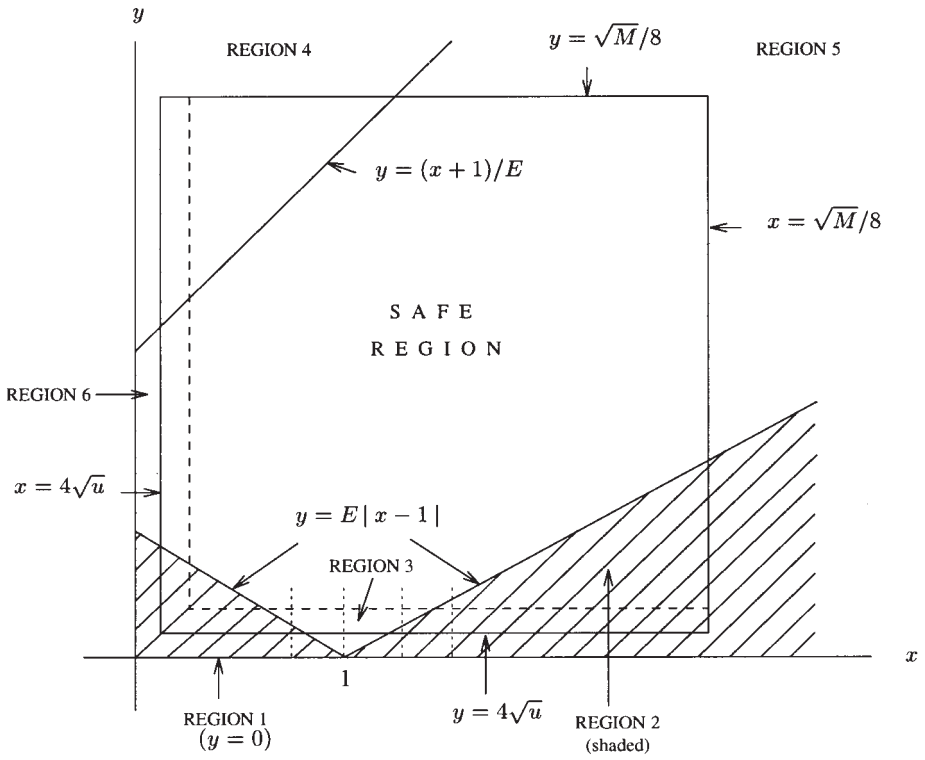

Fig. 3. The safe region in the first quadrant is the square defined by $4\sqrt{u} \le x,y \le \sqrt{M}/8$, where u and M are the underflow and overflow levels respectively. The scale is severely distorted to highlight the different circumstances that must be taken into account. The dashed lines indicate where the lines $x = 4\sqrt{u}$ and $y = 4\sqrt{u}$ could be for a "bad" machine. The dots are the floating-point numbers in the neighborhood of $(1,0)$; notice that, for a "good" machine, no such points lie below $y = 4\sqrt{u}$ and above $y = E|x - 1|$, except for points on the line $x = 1$.

Section 6 an explanation of how the program of this section can be modified to cope with "bad" machines.

To prove that the region is safe, we show that points in the square cannot cause overflow or underflow at any stage in the calculations of the enable block. We have immediately that

$$(x + 1)^2, (x - 1)^2, \text{ and } y^2 <\approx M/64,$$

so no overflow exception can be raised at these stages of the calculations in the region. It follows easily that no overflow can be raised in calculating R, S, and A, and that

$$R, S, \text{ and } A <\approx \sqrt{M/32}.$$

We note also that the calculations of $(x + 1)^2$, $(x - 1)^2$, and y^2 cannot cause underflow to be raised either, and we end up with

$$R, S >\approx \sqrt{32u} \text{ and } A >\approx 1.$$

The latter inequality follows from the fact that $\alpha \geq 1$. Thus the calculations of R, S, and A cannot raise an exception in the safe region.

As for B, we know that $A >\approx 1$ and $x \leq \sqrt{M} / 8$, so that the calculation of B cannot overflow. Moreover,

$$B \approx x / A >\approx 4\sqrt{u} / \sqrt{M / 32} = 16\sqrt{2u} / \sqrt{Mu} > 8\sqrt{2u},$$

because of our assumption that $uM < 4$, so the calculation of B cannot underflow. Thus the calculation of B cannot raise an exception in the safe region.

We turn now to the calculation of the real part. If $B \leq Bcrossover$, the calculation of $\arcsin(B)$ cannot raise an exception in the safe region. However, if $B > Bcrossover$ the situation is more complicated. We consider $x \leq 1$ and $x > 1$ separately. If $x \leq 1$, we have

$$y^2 / (R + (x + 1)) <\approx y^2 / y \leq \sqrt{M} / 8,$$

and then

$$y^2 / (R + (x + 1)) + S + (1 - x) <\approx \sqrt{M} / 8 + \sqrt{M / 32} < \sqrt{M},$$

so no overflow can arise in evaluating this expression. Multiplying by $0.5(A + x)$ leads to an upper bound $\approx M / (8\sqrt{2})$ for the argument of the square root function. Taking the square root and dividing into x leads to a lower bound greater than $6u$ for the argument of arctan, since $uM < 4$, so the arctan function cannot underflow if $B > Bcrossover$ and $x \leq 1$. With regard to possible overflow in this case, we have

$$y^2 / (R + (x + 1)) > \approx y^2 / (\sqrt{4 + y^2} + 2) >\approx 4u,$$

since the middle expression is an increasing function of y and $y \geq 4\sqrt{u}$. Then multiplying by $0.5(A + x)$ leads to a lower bound approximately equal to $2u$ for the argument of the square root, since $A >\approx 1$. Then taking the square root and dividing into x leads to an upper bound approximately equal to $1 / \sqrt{2u}$ for the argument of the arctan function, since $x \leq 1$. Since $u > 1 / (257M)$, this bound is only slightly more than $8\sqrt{2M}$, so the argument of the arctan function cannot overflow. Thus calculating the real part cannot cause an exception to occur in the safe region when $B > Bcrossover$ and $x \leq 1$.

That leaves the case of $x > 1$ to be considered. Arguing as in the case of $x \leq 1$, it is easy to show that the calculations $A + x$, $R + (x + 1)$, and $S + (x - 1)$ cannot overflow or underflow in the safe region when $x > 1$. Nor can the quotients $(A + x) / (R + (x + 1))$ and $(A + x) / (S + (x - 1))$, or the multiplication by 0.5, or taking the square root. It therefore remains only to show that multiplying by y and then dividing into x cannot cause overflow or underflow to occur.

To do this we take advantage of the fact that

$$\arcsin(\beta) = \arcsin((2x) / (\sqrt{(x + 1)^2 + y^2} + \sqrt{(x - 1)^2 + y^2}))$$

is a decreasing function of y for each fixed x. It follows that the rewrite in terms of the arctan function must also be a decreasing function of y for each fixed x. This means that the argument of the arctan function, namely

$$\frac{x}{y} / \sqrt{0.5\left(\frac{A + x}{R + (x + 1)} + \frac{A + x}{S + (x - 1)}\right)}$$

must also be a decreasing function of y for each fixed x. An upper bound for the computed value of that argument in the safe region is therefore approximately the value of the argument at $y = 4\sqrt{u}$. But for this value, we have $y^2 = 16u$, and $16u$ can be neglected in comparison with $(x + 1)^2$ and $(x - 1)^2$, since the smallest value of $x - 1$ when $x > 1$ is $2E$ (rounding arithmetic) or E (chopping arithmetic). An upper bound then works out to be approximately

$$\left(\frac{x}{4\sqrt{u}\sqrt{0.5}}\right) / \left(\sqrt{\frac{x}{x + 1} + \frac{x}{x - 1}}\right) \leq \left(\frac{\sqrt{M} / 8}{4\sqrt{u}\sqrt{0.5}}\right) / \sqrt{2} = \frac{\sqrt{M}}{32\sqrt{u}},$$

for each fixed x. This cannot overflow, since $uM > 1 / 257$, so $1 / \sqrt{u} < \sqrt{257M}$, and an upper bound is therefore approximately $M\sqrt{257} / 32$.

Similarly a lower bound for the computed value of the argument of the arctan function is approximately the value of the argument at $y = \sqrt{M}/8$ for each fixed x. This lower bound is, therefore, approximately

$$\left(\frac{8x}{\sqrt{2M}}\right) / \sqrt{\frac{A + x}{R + (x + 1)} + \frac{A + x}{S + (x - 1)}},$$

where now, since $x > 1$, we have

$$A + x = 0.5(\sqrt{(x + 1)^2 + M / 64} + \sqrt{(x - 1)^2 + M / 64}) + x$$

$$< \sqrt{(x + 1)^2 + M / 64} + x,$$

$$R + (x + 1) = \sqrt{(x + 1)^2 + M / 64} + (x + 1),$$

and

$$S + (x - 1) = \sqrt{(x - 1)^2 + M / 64} + (x - 1).$$

We note that

$$\frac{A + x}{R + x + 1} + \frac{A + x}{S + (x - 1)} < 1 + \frac{\sqrt{(x + 1)^2 + M / 64} + x}{\sqrt{(x - 1)^2 + M / 64} + (x - 1)},$$

where the second term in this sum is a decreasing function of x and is therefore a maximum near $x = 1$. But this maximum value is approximately 1, so a lower bound for the computed value of the argument of the arctan function works out to be approximately

$$4x / \sqrt{M},$$

for each fixed x. Since $x > 1$, we can set $x = 1$ in this expression, and, noting that $uM < 4$ so that $1 / \sqrt{M} > \sqrt{u} / 2$, we see that

$$2\sqrt{u}$$

is, approximately, a lower bound for the argument of the arctan function when $x > 1$. Thus underflow cannot occur. All cases related to the calculation of the real part have now been covered, and we can conclude that the calculation cannot raise an exception in the safe region.

Now for the imaginary part. We have already shown that the calculation of A cannot raise an exception in the safe region. It is then easy to see that the calculation of the imaginary part cannot raise an exception when $A > Acrossover$. When $A \leq Acrossover$ we need to calculate $Am1$. In this case arguments analogous to those given above can be used to determine that $Am1$ cannot raise an exception, and consequently that the calculation of the imaginary part cannot raise an exception when $A \leq Acrossover$.

Altogether then, we have established the safety of the proposed safe region. Of course many points outside the safe region will also not raise exceptions and will not cause a transfer of control to the handler. The point in identifying a safe region is that we have to make provision for coping with exceptions *only* for points that are not in the safe region.

3.3 Handle Block

The main handle block in the program looks after all x, y pairs lying outside the safe region that cause overflow or underflow to occur in the enable block. This is accomplished in Figure 4 by treating separately six cases that define regions in Figure 3 which together cover all of the points

```
if y ≤ E * abs(x − 1) then  - -  Cases (1) and (2)
    if x < 1 then
        Answer.realpart := Arcsin(x)
        Answer.imagpart := y/Sqrt((1 + x) * (1 − x))
    else
        Answer.realpart := Piby2
        enable
            Answer.imagpart := Log1p((x − 1) + Sqrt((x − 1) * (x + 1)))
        handle  - -  (x − 1) * (x + 1) must have overflowed
            Answer.imagpart := Log2 + Log(x)
        end
    endif
elsif y < Foursqrtu then  - -  Case (3)
    Answer.realpart := Piby2 − Sqrt(y)
    Answer.imagpart := Sqrt(y)
elsif E * y − 1 ≥ x then  - -  Case (4)
    enable
        Answer.realpart := x/y
    handle  - -  must have underflowed
        Answer.realpart := 0
    end
    Answer.imagpart := Log2 + Log(y)
elsif x > 1 then  - -  Case (5) - x + 1 and y are both very large
    Answer.realpart := Arctan(x/y)
    Answer.imagpart := Log2 + Log(y) + .5 * Log1p((x/y) * *2)
else  - -  Case (6) - x alone is very small, and E <≈ y <≈ E^−1
    A := Sqrt(1 + y * *2)
    enable
        Answer.realpart := x/A
    handle  - -  must have underflowed
        Answer.realpart := 0
    end
    Answer.imagpart := .5 * Log1p(2 * y * (y + A))
endif
```

Fig. 4. Handle block for the program in Figure 2 that calculates an approximation to the complex arcsine function. The **elsif** clause for Case (3) has to be modified for "bad" machines, as explained later in Section 6.

that lie outside the safe region. The regions are numbered 1,2, . . . , 6 according to the cases to which they correspond.

Case (1): $y = 0$. The corresponding region consists of all points lying on the line $y = 0$. It is convenient to deal with this case separately, mainly because the formulas involved are distinctive and especially simple.

If $y = 0$ then

$$\alpha = 0.5(x + 1 + |x - 1|)$$

$$= \begin{cases} 1, & \text{if } x < 1, \\ x, & \text{if } x \geq 1, \end{cases}$$

so that

$$\beta = \begin{cases} x, & \text{if } x < 1, \\ 1, & \text{if } x \geq 1, \end{cases}$$

and the real part is approximated accordingly, noting that $\arcsin(1) = \pi / 2$.

For the imaginary part,

$$\alpha - 1 + \sqrt{\alpha^2 - 1} = \begin{cases} 0, & \text{if } x < 1, \\ x - 1 + \sqrt{(x-1)(x+1)}, & \text{if } x \geq 1. \end{cases}$$

This is in a form that avoids serious cancellation error in $x^2 - 1$. However, $x - 1 + \sqrt{(x-1)(x+1)}$ might still overflow; however, when that happens, it is approximately $2x$, and the imaginary part can be approximated by $\log(2) + \log(x)$.

Case (2): $y \leq E|x - 1|$. The corresponding region consists of all points on or below the lines $y = E|x - 1|$, except for the points on $y = 0$. This region is shaded in Figure 3.

Here we have $y \neq 0$, and consequently $x \neq 1$, so we can write

$$\alpha = 0.5((x+1)\sqrt{1 + (y/(x+1))^2} + |x-1|\sqrt{1 + (y/(x-1))^2})$$

$$\approx \begin{cases} 1, & x < 1, \\ x, & x > 1, \end{cases}$$

since $(y/(x+1))^2$ and $(y/(x-1))^2$ can be neglected in comparison with 1. For $\beta = x / \alpha$ we have to be more careful. In the first expression for α we can expand in powers of $y^2/(x+1)^2$ and $y^2/(x-1)^2$ and this leads to

$$\beta \approx \begin{cases} x(1 + 0.5y^2/(x^2 - 1)), & x < 1, \\ 1 - 0.5y^2/(x^2 - 1), & x > 1. \end{cases}$$

We then have

$$\beta \approx \begin{cases} x, & x < 1, \\ 1, & x > 1. \end{cases}$$

However, it is important to notice that the relative error in this approximation is a small multiple of E^2, and not just a small multiple of E. (This is crucial in the error analysis of Case (2).) We now set the real part to $\arcsin(x)$ when $x < 1$ and to $\pi / 2$ when $x > 1$, which is the same as what was done in Case (1). In Case (1) the argument of arcsine is exact, but in Case (2) it is only approximate.

For the imaginary part when $\alpha < Acrossover$ we need

$$\alpha m1 \approx \begin{cases} 0.5y^2 / ((1 + x)(1 - x)), & x < 1, \\ x - 1, & x > 1, \end{cases}$$

and we use log1p, with the argument

$$\alpha m1 + \sqrt{(\alpha m1)(\alpha + 1)} \approx \begin{cases} y / \sqrt{(1 + x)(1 - x)}, & x < 1, \\ x - 1 + \sqrt{(x - 1)(x + 1)}, & x > 1. \end{cases}$$

Here the expression for $x < 1$ is so small that

$$\text{log1p}(y / \sqrt{(1 + x)(1 - x)}) \approx y / \sqrt{(1 + x)(1 - x)},$$

while the expression for $x > 1$ might overflow, in which case the imaginary part can be approximated by $\log(2) + \log(x)$.

For the imaginary part when $\alpha \geq Acrossover$ either

$$\log(x + \sqrt{(x - 1)(x + 1)})$$

or

$$\text{log1p}(x - 1 + \sqrt{(x - 1)(x + 1)})$$

can be used. We choose the latter expression because it coincides with the expression used when $\alpha < Acrossover$, as long as $x > 1$. We also note that all the results in Cases (1) and (2) are the same, except for the imaginary parts when $x < 1$, but even then the result in Case (2) reduces to the result in Case (1). The two cases can therefore be merged as indicated in Figure 4.

Case (3): $y < 4\sqrt{u}$. The corresponding region consists of all points below the line $y = 4\sqrt{u}$ which have not been dealt with in Cases (1) and (2). This case is quite simple if $x = 1$, which is the only situation in which Case (3) can arise on "good" machines.

Since $x = 1$, we have

$$\alpha = 0.5(\sqrt{4 + y^2} + y) = 1 + y / 2 + y^2 / 8 + \cdots$$

and

$$\beta = 1 / \alpha = 1 - y / 2 + y^2 / 8 + \cdots.$$

Since β is near 1 we need to use $\arctan(\beta / \sqrt{1 - \beta^2})$ for the real part. We have $1 - \beta \approx y / 2$ and $1 + \beta \approx 2$, while $\beta \approx 1$, so that

$$\beta / \sqrt{1 - \beta^2} \approx 1 / \sqrt{y}.$$

The real part then becomes, approximately,

$$\arctan(1 / \sqrt{y}) = \pi / 2 - \sqrt{y} + (\sqrt{y})^3 / 6 - \cdots$$

$$\approx \pi / 2 - \sqrt{y}.$$

Since α is also near 1, we need

$$\alpha m1 \approx 0.5y$$

and

$$\alpha m1 + \sqrt{(\alpha m1)(\alpha + 1)} \approx \sqrt{y} + y / 2$$

for the imaginary part. The imaginary part then becomes, approximately,

$$\log1p(\sqrt{y} + y / 2) = (\sqrt{y} + y / 2) - 1 / 2(\sqrt{y} + y / 2)^2 + \cdots$$

$$\approx \sqrt{y}.$$

Case (4): $y \geq (x + 1) / E$. The test in the program is in the form $Ey - 1 \geq x$ to avoid possible overflow if $(x + 1) / E$ were to be computed. (Ey cannot underflow since $E^2 \geq 4u$ and $y \geq 4\sqrt{u}$.) The corresponding region consists of all points on or above the line $y = (x + 1) / E$. Here y is so much larger than $x + 1$ that $(x + 1)^2$ and $(x - 1)^2$ can be neglected in comparison with y^2. Then we have

$$\alpha \approx y$$

and

$$\beta \approx x / y.$$

In this case β is certainly less than $Bcrossover$. In fact it is so small that $\arcsin(\beta) \approx \beta$. However, β might underflow, but then the real part is so small compared to the imaginary part that the real part can be set to 0 without having more than a negligible effect on the overall relative error bound, as explained in Section 2.2.

Also in this case α and y are so large that

$$\alpha + \sqrt{\alpha^2 - 1} \approx 2y,$$

and the imaginary part is approximately $\log(2) + \log(y)$.

Case (5): $x > 1$. The corresponding region consists of all points to the right of $x = \sqrt{M} / 8$ or above $y = \sqrt{M} / 8$ which have not already been dealt with in Cases (2) and (4). In this case x and y are both very large, so that

$$\alpha \approx \sqrt{x^2 + y^2}$$

and

$$\beta \approx x / \sqrt{x^2 + y^2}.$$

Then, for the real part,

$$\arcsin(\beta) \approx \arctan(x / y).$$

Since x and y cannot differ by more than a factor of E, x / y cannot overflow or underflow.

For the imaginary part, we need only

$$\alpha + \sqrt{\alpha^2 - 1} \approx 2\alpha,$$

so the imaginary part is close to

$$\begin{aligned}\log(2\alpha) &= \log(2) + 0.5\log(x^2 + y^2) \\ &= \log(2) + 0.5\log(y^2(1 + (x / y)^2)) \\ &= \log(2) + \log(y) + 0.5\log1p((x / y)^2),\end{aligned}$$

and here $(x / y)^2$ cannot overflow or underflow (because of our assumption that $E^2 \geq 4u$ and $E^2 \geq M^{-1}$).

Case (6): Otherwise. The corresponding region consists of all points to the left of $x = 4\sqrt{u}$ which have not already been dealt with in Cases (2) and (4). Thus, the only remaining possibility is that x is very small. In fact $x < 4\sqrt{u}$, so it is negligible in comparison with 1, while $E <\approx y <\approx E^{-1}$.

Then we have

$$\alpha \approx \sqrt{1 + y^2},$$

$$\alpha^2 - 1 \approx y^2,$$

and

$$\beta \approx x / \sqrt{1 + y^2}.$$

Here β is so small that the real part is simply $x / \sqrt{1 + y^2}$, except that this might underflow, in which case it turns out that the real part can be set to 0, with only a negligible effect on the overall relative error bound.

For the imaginary part we need

$$\alpha + \sqrt{\alpha^2 - 1} \approx \sqrt{1 + y^2} + y.$$

We can then rewrite $\log(\alpha + \sqrt{(\alpha^2 - 1)})$ as follows:

$$\begin{aligned}\log(\alpha + \sqrt{\alpha^2 - 1}) &= 0.5\log((\alpha + \sqrt{\alpha^2 - 1})^2) \\ &= 0.5\log(\alpha^2 + 2\alpha\sqrt{\alpha^2 - 1} + \alpha^2 - 1) \\ &= 0.5\log1p(2(\alpha^2 - 1 + \alpha\sqrt{\alpha^2 - 1})) \\ &\approx 0.5\log1p(2y(y + \alpha)).\end{aligned}$$

3.4 Error Analysis

We turn now to the error analysis, and we begin by considering only the paths through the enable block, i.e., we assume for the time being that no exceptions occur. Based on Section 2.1 we have

$$\epsilon_\rho = 2\epsilon + \epsilon_{\text{sqrt}}$$

and

$$\epsilon_\sigma = 2\epsilon + \epsilon_{\text{sqrt}},$$

so that

$$\epsilon_\alpha = 3\epsilon + \epsilon_{\text{sqrt}}$$

and

$$\epsilon_\beta = 4\epsilon + \epsilon_{\text{sqrt}}.$$

Further,

$$\text{epf}(\arcsin(\beta)) = \beta / (\sqrt{1 - \beta^2}\arcsin(\beta)),$$

so that the relative error for the real part when β is not close to 1 is

$$\epsilon_{\text{answer.realpart}} = \frac{\beta}{\sqrt{1 - \beta^2}\arcsin(\beta)}(4\epsilon + \epsilon_{\text{sqrt}}) + \epsilon_{\arcsin}.$$

This is an increasing function of β and can become unacceptably large as β becomes close to 1.

If β is close to 1 the program uses alternative formulas to approximate the real part. For their analysis we need

$$\epsilon_{\alpha+x} = 4\epsilon + \epsilon_{\text{sqrt}},$$

as well as

$$\epsilon_{\rho+(x+1)} = 3\epsilon + \epsilon_{\text{sqrt}}$$

and

$$\epsilon_{\sigma+(1-x)} = 3\epsilon + \epsilon_{\text{sqrt}}, \text{ if } x \leq 1,$$

and

$$\epsilon_{\sigma+(x-1)} = 3\epsilon + \epsilon_{\text{sqrt}}, \text{ if } x > 1.$$

The parentheses around $x + 1$, $1 - x$, and $x - 1$ in these formulas are important; without them each error would be larger by another ϵ. Applying these results to the argument of the sqrt function when $x \leq 1$, we obtain

$$\begin{aligned}\epsilon_{\text{argofsqrt}} &= \epsilon_{\alpha+x} + (\max(\epsilon_{y^2} + \epsilon_{\rho+(x+1)} + \epsilon, \epsilon_{\sigma+(1-x)}) + \epsilon) + \epsilon \\ &= 4\epsilon + \epsilon_{\text{sqrt}} + (6\epsilon + \epsilon_{\text{sqrt}}) + \epsilon \\ &= 11\epsilon + 2\epsilon_{\text{sqrt}},\end{aligned}$$

so that the resulting error in the real part becomes

$$\begin{aligned}\epsilon_{\text{answer.realpart}} &= \text{epf}(\arctan(\beta / \sqrt{1 - \beta^2}))(\epsilon_x + \epsilon_{\sqrt{\text{argofsqrt}}} + \epsilon) + \epsilon_{\arctan} \\ &= (\beta\sqrt{1 - \beta^2} / \arcsin(\beta))(0 + (0.5(11\epsilon + 2\epsilon_{\text{sqrt}}) + \epsilon_{\text{sqrt}}) + \epsilon) \\ &\qquad + \epsilon_{\arctan} \\ &= (\beta\sqrt{1 - \beta^2} / \arcsin(\beta))(6.5\epsilon + 2\epsilon_{\text{sqrt}}) + \epsilon_{\arctan}\end{aligned}$$

when β is close to 1 and $x \leq 1$.

Otherwise, when β is close to 1, and $x > 1$, we have

$$\begin{aligned}\epsilon_{\text{argofsqrt}} &= \max(\epsilon_{\alpha+x} + \epsilon_{\rho+(x+1)} + \epsilon, \epsilon_{\alpha+x} + \epsilon_{\sigma+(x-1)} + \epsilon) + \epsilon \\ &= (4\epsilon + \epsilon_{\text{sqrt}} + 3\epsilon + \epsilon_{\text{sqrt}} + \epsilon) + \epsilon \\ &= 9\epsilon + 2\epsilon_{\text{sqrt}},\end{aligned}$$

so that the resulting error in the real part becomes

$$\begin{aligned}\epsilon_{\text{answer.realpart}} &= \text{epf}(\arctan(\beta / \sqrt{1 - \beta^2}))(\epsilon_x + (\epsilon_y + \epsilon_{\sqrt{\text{argofsqrt}}} + \epsilon) + \epsilon) \\ &\qquad + \epsilon_{\arctan}\end{aligned}$$

$$= (\beta\sqrt{1-\beta^2}\,/\,\arcsin(\beta))(0 + (0 + (4.5\epsilon + \epsilon_{\text{sqrt}} + \epsilon_{\text{sqrt}}) + \epsilon) + \epsilon)$$

$$+ \epsilon_{\text{arctan}}$$

$$= (\beta\sqrt{1-\beta^2}\,/\,\arcsin(\beta))(6.5\epsilon + 2\epsilon_{\text{sqrt}}) + \epsilon_{\text{arctan,}}$$

which is the same as in the case when β is close to 1 but $x \leq 1$. So this is the relative error for the real part when β is close to 1. Note that it is a decreasing function of β.

The remaining task regarding the real part is to decide what crossover value to use for β. The error when β is not close to 1 is an increasing function of β, but the error when β is close to 1 is a decreasing function of β. So the best choice for the crossover value of β is when these two errors are equal. Equating the two, and assuming that $E_{\text{arcsin}} = E_{\text{arctan}}$ and that $E_{\text{sqrt}} = E$, the crossover value of β satisfies the equation

$$5\beta\,/\,(\sqrt{1-\beta^2}\,\arcsin(\beta)) = 8.5\beta\sqrt{1-\beta^2}\,/\,\arcsin(\beta),$$

which leads to our choice of $Bcrossover = 0.6417$.

Our assumptions about E_{arcsin}, E_{arctan}, and E_{sqrt} will not be true in general, but as a reasonable compromise for a variety of systems, we use 0.6417 as the crossover value. In any case the larger of the resulting two error bounds, namely

$$\max(4.804E + 1.201E_{\text{sqrt}} + E_{\text{arcsin}},\ 4.592E + 1.413E_{\text{sqrt}} + E_{\text{arctan}}),$$

is a bound on the relative error in the real part of the complex arcsine function. This bound would be slightly smaller if the crossover value were to be adjusted to make it appropriate for a particular system.

For the imaginary part, we first consider the situation when α is close to 1. There are two cases, but whether $x < 1$ or $x \geq 1$, the error analysis leads to the same result for $\epsilon_{\alpha m1}$, namely

$$\epsilon_{\alpha m1} = 6\epsilon + \epsilon_{\text{sqrt}}.$$

The next stage is to determine the relative error in the argument of the log1p function. This is also straightforward except that the final error bound turns out to be somewhat smaller if we use the general rule for sums in Section 2.1, rather than the max rule, when determining the relative error in approximating $\alpha + 1$, so that

$$\epsilon_{\alpha+1} = (\alpha\,/\,(\alpha + 1))\epsilon_{\alpha} + \epsilon.$$

This leads to

$$\epsilon_{\alpha m1+\sqrt{\alpha m1(\alpha+1)}} = \max(7\epsilon + \epsilon_{\text{sqrt}},\ 5\epsilon + 1.5\epsilon_{\text{sqrt}} + (0.5\alpha\,/\,(\alpha + 1))(3\epsilon + \epsilon_{\text{sqrt}})),$$

and we finally obtain

$$\epsilon_{\text{answer.imagpart}} = \text{epf}\,(\log 1\text{p})\epsilon_{\alpha m1+\sqrt{\alpha m1(\alpha+1)}} + \epsilon_{\log 1\text{p}}$$

$$= \frac{\alpha - 1 + \sqrt{\alpha^2 - 1}}{(\alpha + \sqrt{\alpha^2 - 1})\log(\alpha + \sqrt{\alpha^2 - 1})}$$

$$\times \max(7\epsilon + \epsilon_{\text{sqrt}}, 5\epsilon + 1.5\epsilon_{\text{sqrt}} + (0.5\alpha / (\alpha + 1))(3\epsilon + \epsilon_{\text{sqrt}}))$$

$$+ \epsilon_{\log 1\text{p}}.$$

It can be shown that this expression is bounded by its limiting value as $\alpha \rightarrow 1^+$. To see that this is so it is simplest to plot the two relevant expressions, first

$$(\alpha - 1 + \sqrt{\alpha^2 - 1}) / ((\alpha + \sqrt{\alpha^2 - 1})\log(\alpha + \sqrt{\alpha^2 - 1}))$$

and then

$$(\alpha - 1 + \sqrt{\alpha^2 - 1}) / ((\alpha + \sqrt{\alpha^2 - 1})\log(\alpha + \sqrt{\alpha^2 - 1})) \times (\alpha / (\alpha + 1)).$$

Analytical proofs are also possible. For example, substitute $u = \alpha - 1 + \sqrt{\alpha^2 - 1}$ into the first expression and show that its derivative with respect to u is positively proportional to $\log(1 + u) - u$, which is negative for $u > 0$, i.e., for $\alpha > 1$. The limiting value of the first expression as $\alpha \rightarrow 1^+$ is 1. The analysis for the second expression is more complicated. Its limiting value as $\alpha \rightarrow 1^+$ is 1/2.

We therefore have

$$\epsilon_{\text{answer.imagpart}} = \max(7\epsilon + \epsilon_{\text{sqrt}}, 5.75\epsilon + 1.75\epsilon_{\text{sqrt}}) + \epsilon_{\log 1\text{p}}$$

as the desired result for the imaginary part when α is close to 1.

When α is not close to 1, we use $\epsilon_\alpha = 3\epsilon + \epsilon_{\text{sqrt}}$ and the rule for differences in Section 2.1 to obtain

$$\epsilon_{\alpha^2-1} = \frac{\alpha^2}{\alpha^2 - 1}(7\epsilon + 2\epsilon_{\text{sqrt}}) + \frac{-1}{\alpha^2 - 1}(0) + \epsilon$$

and

$$\epsilon_{\sqrt{\alpha^2-1}} = \frac{\alpha^2}{\alpha^2 - 1}(3.5\epsilon + \epsilon_{\text{sqrt}}) + 0.5\epsilon + \epsilon_{\text{sqrt}}.$$

Since $\alpha \geq \sqrt{\alpha^2 - 1}$ and $\epsilon_\alpha < \epsilon_{\sqrt{\alpha^2-1}}$, we can use the "average" rule for sums given in Section 2.1 to conclude that

$$\epsilon_{\alpha+\sqrt{\alpha^2-1}} = \left(\frac{1.75\alpha^2}{\alpha^2-1} + 2.75\right)\epsilon + \left(\frac{0.5\alpha^2}{\alpha^2-1} + 1\right)\epsilon_{\text{sqrt}}.$$

Then

$$\begin{aligned}\epsilon_{\text{answer.imagpart}} &= \text{epf}(\log(\alpha + \sqrt{\alpha^2-1}))\epsilon_{\alpha+\sqrt{\alpha^2-1}} + \epsilon_{\log} \\ &= (1 / \log(\alpha + \sqrt{\alpha^2-1})) \\ &\times \left(\left(\frac{1.75\alpha^2}{\alpha^2-1} + 2.75\right)\epsilon + \left(\frac{0.5\alpha^2}{\alpha^2-1} + 1\right)\epsilon_{\text{sqrt}}\right) + \epsilon_{\log}\end{aligned}$$

is the relative error in the imaginary part when α is not close to 1.

This second error expression for the imaginary part is a decreasing function of α for $\alpha > 1$. The best choice for the crossover value of α is therefore the value of α which makes this error equal to the maximum of the earlier one for the imaginary part. This is the value of α that minimizes the overall error bound for the imaginary part and, at the same time, ensures that the program uses the less costly of the two ways of approximating the imaginary part as often as is possible.

Assuming that $E_{\text{sqrt}} = E$ and that $E_{\text{log1p}} = E_{\log}$, and equating the two error expressions, we obtain an equation for the crossover value of α, namely

$$8\epsilon = (1 / \log(\alpha + \sqrt{\alpha^2-1}))(3.75 + (2.25\alpha^2 / (\alpha^2-1)))\epsilon.$$

The solution is 1.5088 . . . ; as a reasonable compromise for a variety of systems, we have chosen $Acrossover = 1.5$.

Using the value $\alpha = 1.5$, an error bound for the imaginary part, when α is close to 1, is

$$\max(7.000E + 1.000E_{\text{sqrt}}, 5.750E + 1.750E_{\text{sqrt}}) + E_{\text{log1p}},$$

while the bound, when α is not close to 1, turns out to be

$$6.131E + 1.975E_{\text{sqrt}} + E_{\log}.$$

The larger of these two bounds is therefore a relative error bound for the imaginary part.

To complete the details of the error analysis we need to consider the handler. There is only one special circumstance to mention. Otherwise it is a tedious but straightforward matter to check that none of the cases in the handler can have a bound which exceeds the largest bounds found so far under the assumption that no exception occurs. The special circumstance is

in Case (2) where the real part, $\arcsin(\beta)$, is set to $\arcsin(x)$ when $x < 1$ and $\pi / 2$ when $x > 1$. The resulting error is

$$\epsilon_{\arcsin(\beta)} = \frac{\beta}{\sqrt{1 - \beta^2}\arcsin(\beta)}(\epsilon_\beta) + \epsilon_{\arcsin},$$

and our concern here is that this might be too large when β is near 1. However it was pointed out earlier that ϵ_β is only a small multiple of E^2, whereas $\sqrt{1 - \beta^2} = \sqrt{(1 - \beta)(1 + \beta)}$ is a small multiple of $\sqrt{E}$. From this we can conclude that $\epsilon_{\arcsin(\beta)}$ is only a small multiple of $E^{3/2}$. It should also be noted that many $\approx$ relationships used in deriving the formulas for the handler involve errors which can be neglected because they are only small multiples of E^2, while others have to be taken into account because they involve small multiples of E.

Combining this result with the earlier ones for the real and imaginary parts leads to an overall relative error bound for the complex arcsine function, namely

$$\begin{aligned}\max(&4.804E + 1.201E_{\text{sqrt}} + E_{\arcsin},\\ &4.592E + 1.413E_{\text{sqrt}} + E_{\arctan},\\ &7.000E + 1.000E_{\text{sqrt}} + E_{\text{log1p}},\\ &5.750E + 1.750E_{\text{sqrt}} + E_{\text{log1p}},\\ &6.131E + 1.975E_{\text{sqrt}} + E_{\log}).\end{aligned}$$

This bound is valid under the assumptions given in Section 2.4. The value of this bound will vary from system to system, but if a system's real elementary functions are reasonably accurate, the value should be in the range of about $9E$ to $11E$. In the system we use for our tests in Section 5, the bound turns out to be $9.488E$.

4. COMPLEX ARCCOSINE FUNCTION CARCCOS

The complex inverse cosine function $\arccos(z)$, where $z = x + iy$, can be defined by

$$\arccos(z) = \arccos(\beta) - i\ \text{sign}(y)\log(\alpha + \sqrt{\alpha^2 - 1}),$$

where

$$\alpha = 0.5(\sqrt{(x + 1)^2 + y^2} + \sqrt{(x - 1)^2 + y^2}),$$

$$\beta = 0.5(\sqrt{(x + 1)^2 + y^2} - \sqrt{(x - 1)^2 + y^2}),$$

```
function CARCCOS ( z : complex ) : complex
    real x, y, R, S, A, Am1, B
    real E, Foursqrtu, Log2, Piby2, Pi, Acrossover, Bcrossover ← appropriately initialized
    complex Answer
    x := abs(z.realpart)      y := abs(z.imagpart)
    enable
        R := Sqrt((x + 1) * *2 + y * *2)      S := Sqrt((x − 1) * *2 + y * *2)
        A := .5 * (R + S)      B = x/A
        if B ≤ Bcrossover then
            answer.realpart := Arccos(B)
        else  - - use arctan and an accurate approximation to α − x
            if x ≤ 1 then
                Answer.realpart :=
                    Arctan(Sqrt(.5 * (A + x) * ((y * *2)/(R + (x + 1)) + (S + (1 − x))))/x)
            else
                Answer.realpart :=
                    Arctan((y * Sqrt(.5 * ((A + x)/(R + (x + 1)) + (A + x)/(S + (x − 1)))))/x)
            endif
        endif
        if A ≤ Acrossover then  - - use log1p and an accurate approximation to α − 1
            if x < 1 then
                Am1 := .5 * ((y * *2)/(R + (x + 1)) + (y * *2)/(S + (1 − x)))
            else
                Am1 := .5 * ((y * *2)/(R + (x + 1)) + (S + (x − 1)))
            endif
            Answer.imagpart := Log1p(Am1 + Sqrt(Am1 * (A + 1)))
        else
            Answer.imagpart := Log(A + Sqrt(A * *2 − 1))
        endif
    handle
        - -
        - - (see handle block in Figure 6)
        - -
    end
    if z.realpart < 0 then
        Answer.realpart := Pi − Answer.realpart
    endif
    Answer.imagpart := −sign(z.imagpart) * Answer.imagpart
    return Answer
end CARCCOS
```

Fig. 5. A program for calculating an approximation to the complex arccosine function. We use $Acrossover = 1.5$ and $Bcrossover = 0.6417$, but these values could be adjusted to give slightly better error bounds, depending on the system being used.

and where the log function is the natural log. The branch cuts are on the real line from $-\infty$ to –1 and from 1 to ∞. The real and imaginary parts are based on those given by Abramowitz and Stegun [1972, pp. 80–81], but we have chosen their signs to provide what is generally considered to be the principal value of this function.

These formulas are similar to the formulas given for the arcsine function. The real part becomes $\arccos(\beta)$ instead of $\arcsin(\beta)$, and the sign of the imaginary part has changed from $+\text{sign}(y)$ to $-\text{sign}(y)$. Therefore, the program we have developed for this function in Figures 5 and 6 is almost

```
if y ≤ E * abs(x − 1) then  - -  Cases (1) and (2)
    if x < 1 then
        Answer.realpart := Arccos(x)
        Answer.imagpart := y/Sqrt((1 + x) * (1 − x))
    else
        Answer.realpart := 0
        enable
            Answer.imagpart := Log1p((x − 1) + Sqrt((x − 1) * (x + 1)))
        handle  - -  (x − 1) * (x + 1) must have overflowed
            Answer.imagpart := Log2 + Log(x)
        end
    endif
elsif y < Foursqrtu then  - -  Case (3)
    Answer.realpart := Sqrt(y)
    Answer.imagpart := Sqrt(y)
elsif E * y − 1 ≥ x then  - -  Case (4)
    Answer.realpart := Piby2
    Answer.imagpart := Log2 + Log(y)
elsif x > 1 then  - -  Case (5) - x + 1 and y are both very large
    Answer.realpart := Arctan(y/x)
    Answer.imagpart := Log2 + Log(y) + .5 * Log1p((x/y) * *2)
else  - -  Case (6) - x alone is very small, and E <≈ y <≈ E^−1
    Answer.realpart := Piby2
    A := Sqrt(1 + y * *2)
    Answer.imagpart := .5 * Log1p(2 * y * (y + A))
endif
```

Fig. 6. Handle block for the program in Figure 5 that calculates an approximation to the complex arccosine function. The **elsif** clause for Case (3) has to be modified for "bad" machines, as explained later in Section 6.

exactly the same as the program for the complex inverse sine function. There are three differences. One is that the appearances of Arcsin must be replaced by Arccos; as a consequence of this, the arguments of Arctan must be inverted (in three places), and there is an occurrence of the real part being assigned the value of $Piby2$ in the handler (in the merged Cases (1) and (2)) which must be replaced by 0. (The real part can be set to 0 because the absolute error in the real part is negligible compared to the absolute error in the imaginary part.) There is also one occurrence of the real part being assigned the value 0 (in Case (6)) which must be replaced by $Piby2$, and finally there is a corresponding change in the real part of Case (3). The second difference is in the real parts of Cases (4), (5), and (6). In Case (4) of the arcsine program the real part is Arcsin(x / y), but because x / y is very small the real part is just x / y, whereas in the arccosine program the real part is Arccos(x / y), but because x / y is very small the real part can be approximated by $Piby2$. In Case (5) the real part of the arccosine program is changed to Arctan(y / x). In Case (6) the real part of the arcsine program is Arcsin(x / A) which can be replaced by x / A, but the corresponding real part in the arccosine program must be Arccos(x / A); but, because x / A is very small the real part is accurately approxi-

mated by $Piby2$. The third difference is in the final determination of the real and imaginary parts of the answer at the end of the program; their signs have to be accounted for at this point, and this necessitates replacing $\arccos(B)$ with $Pi - \text{Arccos}|B|$ when x and consequently $B < 0$, which in turn requires that Pi be declared and initialized at the beginning of the program.

The only resulting change in the error analysis is that $E_{\arcsin}$ must be replaced by $E_{\arccos}$. The cancellation in $Pi - \text{Arccos}|B|$ at the end of the program does not increase the error bound for the real part, since $\text{Arccos}(|B|) <\approx \pi / 2$. Thus

$$\begin{aligned}\max(&4.804E + 1.201E_{\text{sqrt}} + E_{\arccos},\\ &4.592E + 1.413E_{\text{sqrt}} + E_{\arctan},\\ &7.000E + 1.000E_{\text{sqrt}} + E_{\text{log1p}},\\ &5.750E + 1.750E_{\text{sqrt}} + E_{\text{log1p}},\\ &6.131E + 1.975E_{\text{sqrt}} + E_{\log})\end{aligned}$$

is an overall relative error bound for all input values of the argument z and is valid under the assumptions given in Section 2.4.

5. FORTRAN IMPLEMENTATIONS AND TESTING

We have implemented the algorithms presented in Sections 3 and 4 in Fortran 77 and run them on a Sun SPARCstation 5 (compiler version SC3.0.1), in order to test their correctness, especially the correctness of their error bounds. Both single- and double-precision versions are available. (Some care was necessary to make sure that the single-precision versions did not make any use of higher precision, other than what is done in the library functions they use, for that could have unfairly reduced the observed errors.) The exception-handling constructs are implemented using Sun's math library routine "**swapEX_**" [Sun Microsystems 1991], for the reasons given in our earlier paper [Hull et al. 1994a; 1994b].

We first present the results for the single-precision versions. For these we need to have the values of E_{sqrt}, $E_{\arcsin}$, etc. for single precision. These relative error bounds were found by comparing the results of real Sqrt, real Arcsin, etc., at all relevant single-precision arguments with their corresponding "true" values (which we approximate with accurate double-precision values). The results, in units of E, are presented in Table I. (Four of these bounds are simply 1.000; this is because, except for Sqrt, the corresponding single-precision functions derive rounded single-precision values from their double-precision versions.) These values are then substituted into the theoretical error bounds of Sections 3 and 4 to provide the

Table I. Observed Error Bounds for Single-Precision Real Elementary Functions in the Sun Library (version SC3.0.1) in Units of E, the Relative Error Bound for Single-Precision Real Arithmetic

E_{sqrt}	E_{arcsin}	E_{arccos}	$E_{\log}$	$E_{\log 1p}$	E_{arctan}
1.000	1.000	1.000	1.382	1.000	1.326

specific theoretical error bounds for the environment in which the tests were carried out. The results are shown in column 2 of Table II.

To determine the errors in the results produced by the programs in Sections 3 and 4, we need to know the "true" results. We therefore developed separate programs, and to be especially convincing, we based them on entirely different formulas, namely [Churchill et al. 1974, p. 70]

$$\arcsin(z) = -i \log(iz + \sqrt{1 - z^2}),$$

$$\arccos(z) = -i \log(z + i\sqrt{1 - z^2}).$$

We first note that $\arcsin(-z) = -\arcsin(z)$ and $\arcsin(\bar{z}) = \overline{\arcsin(z)}$, and similarly for arccos, so we had to analyze these formulas for only one quadrant.

The real part of $\sqrt{1 - z^2}$ can be calculated very accurately in double precision if one is careful about the order in which the operations in $1 - x^2 + y^2$ are carried out. Then double-precision complex Sqrt and Log functions can be used, being careful to use a real Log1p function when the argument of Log is near 1 in modulus. The details of the analysis are quite complicated and are not provided here.

This process provided us with the "true," i.e., double-precision, values to compare with the approximations produced by the programs in Sections 3 and 4. For each of CARCSIN and CARCCOS we generated approximately 900 million random numbers in a single quadrant, using random exponents and random significands over the full ranges of these values, and similarly another 900 million in the square $2^{-24} \le x, y < 2^2$, since it is in this region that larger errors are more likely to occur. The largest error observed for each of CARCSIN and CARCCOS is shown in column 3 of Table II, along with the points, in column 4, where they were observed. The experimentally observed bounds are respectively about 53% and 56% of the theoretical bounds, so the observed bounds are well within the theoretical error bounds, but close enough to indicate that the theoretical bounds are quite tight.

For the double-precision versions of CARCSIN and CARCCOS we did not try to determine theoretical bounds. This was mainly because it would be impractical to determine exactly the double-precision values of E_{sqrt}, E_{arcsin}, etc. The three values, other than E_{sqrt}, that are E in single precision will be a little larger in terms of double-precision E; as a consequence the

Table II. Comparison of Theoretical and Observed Relative Error Bounds for the Single-Precision Sun Fortran 77 (version SC3.0.1) Implementations of the Complex Function Programs of Sections 3 and 4, in Units of Single Precision E

Function	Theoretical Bound Based on Table I	Observed Bound	Observed at this Argument (in hex)
CARCSIN	9.488	4.982	3b04cc93 + i3c80f0b7
CARCCOS	9.488	5.273	3fc68f6b + i3a006e09

theoretical error bounds for double-precision versions of CARCSIN and CARCCOS will be correspondingly larger.

However, we can still use our technique for finding "true" values by repeating what was done before in higher precision, double instead of single, and quadruple instead of double. It did not seem to be worthwhile testing as many random points as we did with the single-precision versions, since the two versions are so similar. We generated only about 50 million points for each of CARCSIN and CARCCOS. The observed error bounds in terms of double-precision E are shown in Table III. The bounds are somewhat smaller than the observed bounds in single precision. This is not surprising, considering the relatively small number of random points used, even despite the fact that the theoretical bounds in double precision would be slightly larger.

It may be of some further interest to note that about 225 million of the first 900 million or so random points we generated in single precision for each function were in the safe region, but about 305 million went through only the enable block. The others went through the handler, about 8 thousand in Case (1), 347 million in (2), only 1 in (3), 201 million in (4), 43 million in (5), and 4 million in (6). We generated about 1 million more for Case (3), and the maximum observed bound for these inputs was only $0.477E$.

What about other complex arcsine and arccosine programs that are generally available? Some libraries (such as the NAG and PORT libraries) do not provide such programs. Others that do include such programs are FN, IMSL, and SLATEC, but they are all based on earlier work by Fullerton. For this reason we tested only his programs from the FN library.[1]

We first submitted his CASIN program to the same test described above for our CARCSIN program. Our test program continues until it has found about 900 million successful results, but it also counts the number of unsuccessful results, i.e., results that have raised overflow, underflow, or invalid. For CASIN more than 133 million unsuccessful results were counted. The maximum overall relative error observed for the 900 million or so successful results was almost 2, which, in units of E, is greater than 33,000,000 E. We repeated the test with random numbers restricted to the safe region defined in Section 3.2. We generated only one million such

[1]Available via http://www.netlib.org/fn/

Table III. Observed Relative Error Bounds for the Double-Precision Sun Fortran 77 (version SC3.0.1) Implementations of the Complex Function Programs of Sections 3 and 4, in Units of Double Precision E

Function	Observed Bound	Observed at this Argument (in hex)
CARCSIN	4.813	3fe05586534aa5bb + i3eac11599418af5b
CARCCOS	4.554	3ff93adf91f6645d + i3fbf643281bf9cc7

numbers. As expected, no unsuccessful results were counted; but the overall relative error bound did not change significantly. We repeated the test again in what should be a "very safe" region with a million random numbers restricted to the square in which $2^{-10} \leq x, y < 2^{10}$. Again no unsuccessful results were counted, but the maximum observed error was only $43.59E$. These results are shown in the third line of Table IV. The corresponding results for Fullerton's CACOS are shown in the fourth line. (The maximum error for CACOS in the "very safe" region is considerably smaller than for CASIN. This is because CACOS is based on the formula $\arccos(z) \equiv \pi / 2 - \arcsin(z)$, and $\pi / 2$ dominates $\arcsin(z)$ when the latter experiences its largest error.)

The corresponding results for our CARCSIN and CARCCOS programs are shown in the first two lines of Table IV. And the last column shows the times taken for one million random numbers in the safe region. To make the timing comparisons fair, we made those runs separately from the ones used to determine the maximum errors; we also removed the library support routines from Fullerton's programs which would otherwise have increased overhead time not present in our programs. However we did not remove provision for the handler from our programs, since the handler is a necessary part of those programs.

The results are summarized in Table IV. They show that our programs are both accurate and efficient. Even when the region is severely restricted so that Fullerton's programs are reasonably reliable, ours are both considerably more accurate.

6. WHAT IF THE EXPONENT RANGE IS TOO NARROW?

In developing Case (3) of the main handler for the CARCSIN function (Figure 4), the expressions for α, β, $\alpha - 1$, and $1 - \beta$ were very much simplified because $x = 1$. However, x can be $\neq 1$ in Case (3) for "bad" machines, and we must proceed more carefully.

We have

$$\rho \approx x + 1$$

and

$$\sigma = |x - 1| \sqrt{1 + (y / (x - 1))^2}.$$

Table IV. Comparisons of Our Single-Precision CARCSIN and CARCCOS Programs with Fullerton's CASIN and CACOS Programs (observed maximum errors are in units of E; times are in seconds)

	900 Million Successful Runs		1 Million Successful Runs			
Programs	Number Not Successful	Observed Max Errors Entire Quadrant	Number Not Successful	Observed Max Errors Safe Region	Observed Max Errors Very Safe Region	Timing in a Safe Region
CARCSIN	0	4.982	0	3.657	4.534	14.5
CARCOS	0	5.273	0	3.736	4.047	12.9
CASIN	1.336×10^8	3.355×10^7	0	3.353×10^7	43.59	24.7
CACOS	1.336×10^8	3.355×10^7	0	3.355×10^7	18.44	26.4

There is no possibility of overflow or underflow if σ is evaluated in this form, since $y < 4\sqrt{u}$ and $(x - 1)^2 \geq E^2$ (rounding arithmetic) or $E^2/4$ (chopping arithmetic). Here $(y / (x - 1))^2$ is so small that

$$\alpha \approx 0.5(x + 1 + \sigma) \approx \begin{cases} 1, & x < 1, \\ x, & x > 1, \end{cases}$$

and

$$\beta = x / \alpha \approx \begin{cases} x, & x < 1, \\ 1, & x > 1. \end{cases}$$

Both α and β are near 1, so we will need to use the arctan function to compute the real part of $\arcsin(z)$ and the log1p function to compute its imaginary part.

For $x < 1$, we obtain

$$\alpha - x \approx 0.5(\sigma + (1 - x))$$

and

$$\alpha + x \approx 1 + x,$$

and the real part is approximately

$$\arctan(x / \sqrt{0.5(1 + x)(\sigma + (1 - x))}).$$

For the imaginary part, we note that

$$\alpha - 1 \approx 0.25y^2(1 / (x + 1) + 2 / (\sigma + (1 - x))),$$

$$\alpha + 1 \approx 2,$$

and

$$\sqrt{\alpha - 1} \approx 0.5y\sqrt{1 / (x + 1) + 2 / (\sigma + (1 - x))}.$$

The imaginary part is

$$\text{log1p}(\sqrt{\alpha - 1}(\sqrt{\alpha - 1} + \sqrt{2})).$$

This expression cannot suffer intermediate underflow.

Similarly, for $x > 1$, we obtain

$$\alpha - x \approx 0.25y^2(1 / (x + 1) + 2 / (\sigma + (x - 1)))$$

and

$$\alpha + x \approx 2x,$$

so the real part is approximately

$$\arctan(x / (y\sqrt{x / (2(x + 1)) + x / (\sigma + (x - 1))})).$$

For the imaginary part we note that

$$\alpha - 1 = \alpha m1 \approx 0.5(\sigma + (x - 1))$$

and

$$\alpha + 1 \approx (1 + x),$$

and the imaginary part is $\text{log1p}(\alpha m1 + \sqrt{\alpha m1(1 + x)})$.

As a result the **elsif** clause for Case (3) in the handler of Figure 4 for CARCSIN should be replaced by what is shown in Figure 7, whereas the corresponding clause in Figure 6 for the CARCCOS function should be replaced by what is shown in Figure 8.

7. CONCLUDING REMARKS

We have presented algorithms for calculating approximations to the complex arcsine and arccosine functions, along with theoretical error bounds. Our implementations of these algorithms provide an approximation to within a relative error of less than $9.5E$ for every machine-representable point in the complex plane. The results of extensive testing help to confirm the correctness of the error bounds, and also show that the bounds are quite tight. We hope to publish a sequel to this article for the complex arctangent function.

Our algorithms have been expressed using an exception-handling facility which we have found to be very convenient and efficient. For each function there is a fast way of calculating an accurate result which works most of the time, so the handler needs to take over only in those cases, rare in

```
if x = 1 then
    Answer.realpart := Piby2 − Sqrt(y)
    Answer.imagpart := Sqrt(y)
else
    S := abs(x − 1) * Sqrt(1 + (y/(x − 1)) * *2)
    if x < 1 then
        Answer.realpart := Arctan(x/Sqrt(.5 * (1 + x) * (S + (1 − x))))
        SqrtAm1 := .5 * y * Sqrt(1/(1 + x) + 2/(S + (1 − x)))
        Answer.imagpart := Log1p(SqrtAm1 * (SqrtAm1 + Sqrt2))
    else
        Answer.realpart := Arctan(x/(y * Sqrt(x/(2 * (x + 1)) + x/(S + (x − 1)))))
        Am1 := .5 * (S + (x − 1))
        Answer.imagpart := Log1p(Am1 + Sqrt(Am1 * (1 + x)))
    endif
endif
```

Fig. 7. For "bad" machines, the **elsif** clause for Case (3) of the handler of Figure 4 should be replaced by what is shown here. The real constant $Sqrt2$ should be declared and appropriately initialized. The real variable $SqrtAm1$ also needs to be declared.

```
if x = 1 then
    Answer.realpart := Sqrt(y)
    Answer.imagpart := Sqrt(y)
else
    S := abs(x − 1) * Sqrt(1 + (y/(x − 1)) * *2)
    if x < 1 then
        Answer.realpart := Arctan(Sqrt(.5 * (1 + x) * (S + (1 − x)))/x)
        SqrtAm1 := .5 * y * Sqrt(1/(1 + x) + 2/(S + (1 − x)))
        Answer.imagpart := Log1p(SqrtAm1 * (SqrtAm1 + Sqrt2))
    else
        Answer.realpart := Arctan(y * Sqrt(x/(2 * (x + 1)) + x/(S + (x − 1)))/x)
        Am1 := .5 * (S + (x − 1))
        Answer.imagpart := Log1p(Am1 + Sqrt(Am1 * (1 + x)))
    endif
endif
```

Fig. 8. For "bad" machines, the **elsif** clause for Case (3) of the handler of Figure 6 should be replaced by what is shown here. The real constant $Sqrt2$ should be declared and appropriately initialized. The real variable $SqrtAm1$ also needs to be declared.

practice, when the fast way produces a spurious overflow or underflow. It is to be hoped that convenient exception-handling facilities will become more widely available than they are at present. (ADA [Intermetrics 1994] comes close, but it does not recognize underflow. A similar system has been proposed for Fortran [IFIP Working Group 2.5 1993].) In the meantime it is possible to adapt our programs to work in environments without exception-handling facilities. Essentially what needs to be done is to first check the argument of the function to see if it lies within a safe region and, if it does, calculate as in the enable block, but otherwise calculate as in the handler; the exception handling within the main handler itself has to be imple-

mented by pretesting the arguments in appropriate places to determine whether or not exceptions can occur.

Finally, we should acknowledge that we have made no special provision for input and/or output of complex numbers with denormalized components, or other components specified in the IEEE standard. Much more would have to be done to make provision for all such possibilities. Some possibilities involving denormalized numbers might be handled relatively easily (although the convenience of tight relative error bounds could be lost), but in general there is a major difficulty in that there is not even a consensus about what specifications should be adopted, especially in cases involving $\pm\infty$, or in distinguishing between $+0$ and -0 [Tydeman 1992].

ACKNOWLEDGMENTS

We wish to thank P. G. Rooney for helpful discussions during the preparation of this article, and Fred Tydeman for reading an earlier version and making many useful suggestions. We thank Vicky Shum for her assistance with the typesetting of the article. A referee's careful comments helped us clarify the presentation.

REFERENCES

ABRAMOWITZ, M. AND STEGUN, I. 1972. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Applied Mathematics Series, vol. 55. 10th printing. National Bureau of Standards, Washington, D.C.

ANSI. 1978. American National Standard programming language FORTRAN: ANSI X3.9-1978. American National Standards Institute, New York, NY.

CHURCHILL, R. V., BROWN, J. W., AND VERHEY, R. F. 1974. *Complex Variables and Applications*. 3rd ed. McGraw-Hill, Inc., New York, NY.

CODY, W. J. 1988. Algorithm 665: MACHAR: A subroutine to dynamically determine machine parameters. *ACM Trans. Math. Softw. 14*, 4 (Dec.), 303–311.

HULL, T. E., FAIRGRIEVE, T. F., AND TANG, P. T. P. 1994a. Implementing complex elementary functions using exception handling. *ACM Trans. Math. Softw. 20*, 2 (June), 215–244.

HULL, T. E., FAIRGRIEVE, T. F., AND TANG, P. T. P. 1994b. Corrigenda: Implementing complex elementary functions using exception handling. *ACM Trans. Math. Softw. 20*, 4 (Dec.).

IEEE. 1985. ANSI/IEEE standard for binary floating point arithmetic: Standard 754-1985. IEEE Press, Piscataway, NJ.

IFIP WORKING GROUP 2.5. 1993. The enable construct for exception handling in Fortran 90. *SIGNUM Newsl. 28*, 4 (Oct.), 7–16.

INTERMETRICS. 1994. Ada 9X reference manual. Intermetrics, Inc., Burlington, MA.

SUN MICROSYSTEMS. 1991. Numerical computations guide: Part number 800-5277-10, Revision A. Sun Microsystems, Incorporated, Mountain View, CA.

TYDEMAN, F. J. 1992. *Merging complex and IEEE-754*. Rep. 92-061 of ANSI X3J11.1 (NCEG). American National Standards Institute, New York, NY.

Received: January 1996; revised: February 1997; accepted: February 1997