An Implementation of Complex APL

Eugene E. McDonnell

> Once we conceive of the real line as embedded in a plane of complex numbers, we have entered a whole new domain of mathematics. All our old knowledge of real algebra and analysis becomes enlarged and enriched when reinterpreted in the complex domain. In addition, we immediately see countless new problems and questions which could not even have been raised in the context of the real numbers alone.
>
> Philip J. Davis and Reuben Hersh, The Mathematical Experience, Birkhausen, Boston (1980)

Complex numbers have been added to Sharp APL as an internal data type, and most of the primitive functions have been extended, where appropriate, to give complex results and to accept arrays containing complex numbers as arguments.

Without this extension, writers of APL applications requiring complex numbers have to simulate them by various devices which make them awkward to work with. Generally speaking, existing applications are not affected by this change. The potential differences are relatively minor and are local to just a few primitive functions. Those functions which would benefit from complex numbers now yield complex results instead of a Domain Error.

The extension is not complete. The floor, ceiling, residue, representation, and dyadic format functions have not been extended, because there remain points of uncertainty regarding their definitions. The way in which complex numbers are displayed is provisional and may be changed.

Complex-Value Input and Display

A complex constant is denoted by the letter *J* connecting two real scalar numeric constants. For example, in 3*J*¯4, the 3 is the real part and the ¯4 is the imaginary part. Each of the two real numbers can be in integer, decimal, or scaled representation; for example, 3*J*1*E*¯20 or 1*E*12*J*3.14.

In the default-format display of a column of numbers, some of which have nonzero imaginary parts, all the numbers in the column are right justified. For example, if *Z* is the vector whose elements are the fifth roots of ¯32, a one-column matrix *W* derived from it would be displayed as follows (with ⎕*PP* at 6):

```
      ⎕←W←5 1⍴Z
 1.61803J1.17557
¯0.61803J1.90211
               ¯2
¯0.61803J¯1.90211
 1.61803J¯1.17557
```

Each part of a complex number is formatted separately. For example, with ⎕*PP*←3, we have:

```
      ¯9 ¯11+.○ 1234 0.1234
1.23E3J0.123
```

Extensions of the Primitive Functions

Many of the primitive functions need no discussion since their extension to complex arguments is well understood. Here we shall describe only those functions whose extension is not obvious.

Conjugate: +⍵ is the conjugate of ⍵, that is, the value obtained by reflecting ⍵ on the real axis. The conjugate of a real number ⍵ is equal to ⍵. Thus, a test to see whether a number is real is ⍵=+⍵.

```
          o   |
       Z[2]   |      Z[1]
              |        o
              |
  Z[3]        |
  -o----------+-----+------
              |    1
              |    Z[5]
              |      o
       Z[4]   |
          o   |
```

Figure 1. Fifth roots of ¯32

The elements of the vector *Z* are shown in Figure 1. The numbers *Z*[1] and *Z*[5] are conjugates, as are *Z*[2] and *Z*[4]. The number *Z*[3] is equal to its own conjugate, since it is real.

Magnitude: |⍵ is the magnitude or modulus (as it is often called) of ⍵. It is the value obtained by rotating ⍵ about the origin onto the positive axis. It may be defined as (⍵×+⍵)*0.5. For example:

```
|3J¯4              ↔  5
|0.6J¯0.8          ↔  1
|1.61803J¯1.17557  ↔  2
|¯1J1              ↔  1.41421
```

Direction: ×⍵ is the direction of ⍵, and is an extension to the signum function on real numbers. The direction of 0 is 0. For nonzero ⍵, a ray drawn from the origin through ⍵ intersects the unit circle at ×⍵. Thus the magnitude of ×⍵ for nonzero ⍵ is 1, and ×⍵ may be defined as ⍵÷|⍵. For example:

```
    ×3J¯4           ↔   0.6J¯0.8
    ×0.03J¯0.04     ↔   0.6J¯0.8
    ×0J10           ↔   0J1
```

New circular-function left arguments: Ten new left arguments α have been provided for α○ω, primarily for use with complex numbers. The correspondence between left argument value and function is different from that given in [1]. The assignments described here preserve, as much as possible, the association of odd functions (real part, imaginary part) with odd left arguments (9 and 11), and even functions (magnitude) with even left arguments (10). They also eliminate the gap which the earlier scheme had left between 8 and 11:

```
(-α)○ω  |  α  |   α○ω
--------+-----+---------------------------
  -8○ω  |  8  | 0J1×ω×(1+ω*¯2)*0.5
     ω  |  9  | (ω++ω)÷2     real part
    +ω  | 10  | |ω           magnitude
 0J1×ω  | 11  | (ω-+ω)÷0J2   imaginary part
*0J1×ω  | 12  | 11○⍟ω    arc or phase angle
```

Two of these, 8○ω and ¯8○ω, are new Pythagorean functions, each a modification of the expression (¯1-ω*2)*0.5. The two forms allow for both signs of the square root. The value of these functions on real numbers is never real, which is why they haven't been defined until now. With them, the set of Pythagorean functions is complete.

The remaining new left arguments for α○ω are useful in forming and decomposing complex numbers, based on the rectangular and the polar representations of these numbers. A number ω may be decomposed into its real and imaginary parts by 9 11○ω. For example:

```
    9 11○3J¯4   ↔   3 ¯4
```

Conversely, a pair of real numbers ω representing the real and imaginary parts of a complex number may be formed into that number by ¯9 ¯11+.○ω. For example:

```
    ¯9 ¯11+.○3 ¯4   ↔   3J¯4
```

A number ω may be decomposed into its magnitude and arc by 10 12○ω. For example:

```
    10 12○3J¯4   ↔   5 ¯0.927295
```

Conversely, a pair of real numbers ω representing the magnitude and arc of a complex number may be formed into that number by ¯10 ¯12×.○ω. For example:

```
    ¯10 ¯12×.○5 ¯0.927295   ↔   3J¯4
```

The arc is given in radians and is always greater than minus pi radians and less than or equal to pi radians. A positive number has an arc of 0. A negative number has an arc of pi. The arc of 0 is defined to be 0. For example:

```
      12○W
 0.628319
 1.88496
 3.14159
¯1.88496
¯0.628319
```

If we define DEG:(180×ω)÷○1 ⍝ RADIANS TO DEGREES, we can display the values of 12○W in degrees. For example:

```
      DEG 12○W
  36
 108
 180
¯108
 ¯36
```

Equals and not equals: Two complex numbers are considered equal if the one smaller in magnitude lies on or within a circle whose center is at the one with larger magnitude, and whose radius is equal to ⎕CT times the larger magnitude.

Greatest common divisor and least common multiple: A complex integer is one whose real and imaginary parts are integers.

If A and B are complex integers, there are four complex integers with the property that they are the largest in magnitude of all the complex integers which evenly divide both A and B. A∨B is that one of these which is in the first quadrant, or on the positive axis. For example, 117J44 and ¯63J¯16 have as greatest divisors the following numbers:

```
    3J¯4 4J3 ¯3J4 ¯4J¯3
```

Of these, 4J3 is given as the value of 117J44∨¯63J¯16 since it is the one in the first quadrant.

A∨B is defined for noninteger complex numbers as well:

```
      1.17J0.44∨¯0.63J¯0.16
0.04J0.03
```

The least common multiple of two complex numbers, α∧ω, is defined by (α×ω)÷α∨ω. For example:

```
    ¯182J¯107∧¯7J55   ↔   ¯75J¯289
```

The least common multiple function is also defined on nonintegral complex numbers.

Functions Not Defined on Complex Numbers

Because the complex numbers are not ordered, those functions which depend on ordering are not extended to complex numbers. They are the dyadic functions α<ω, α≤ω, α≥ω, α>ω, α⌊ω, α⌈ω, α⍋ω, and α⍒ω, and the monadic functions ⍋ω and ⍒ω.

Functions with Deferred Extensions

Because we have not agreed on definitions for the functions ⌊ω, ⌈ω, α|ω, and α⊤ω, they have not been extended at this time.

The formatting function α⍕ω will take a complex argument, but its definition has not been extended to display imaginary parts; it formats only the real part. For example:

```
      0⍕3J¯4 0J2 0.2 5.2
3 0 0 5
```

Differences Affect Existing Applications

Users should be aware that some differences arise even where complex numbers are not used as arguments. This section describes these differences in detail.

There are two kinds of differences. The first is that some functions which used to signal a Domain Error for certain real arguments now have a value which is not, in general, a real number. The functions in this category are ⍟ω and α⍟ω for negative arguments, α*ω for negative α and certain ω, and α○ω for certain arguments. The second is that some functions now have a different value for certain arguments. There are two cases of this: α*ω for negative α and certain ω, and ¯4○ω. All this is covered in detail below.

In the first case, it seems unlikely that a user program would be affected, since one doesn't ordinarily write a program to cause a Domain Error to be signalled. However, with automatic trapping of errors, it is possible that someone could have written a program in such a way that a Domain Error used to be signalled by an expression which now has a value. This would, of course, cause the program behavior to be different.

I.P. Sharp monitored usage of their system extensively to try to judge the impact of those changes where a different value is given. Over a one-month period in 1979 there were 439 evaluations of α*ω with α<0 and ω not an integer. On investigation, almost all of these uses were tests by Sharp development people. There were only 176 uses of ¯4○ω, for all arguments. This compares with 6,149,426 uses of 1○ω, for example. Thus, in both of these cases where a change in value occurs, very few programs could be affected.

First case--Domain Error replaced by value: There are four functions which are affected.

1. The monadic logarithm function used to signal a Domain Error for negative arguments. With complex numbers available, we can use the compatibly extended definition of logarithm for all numbers except 0:

```
      ⍟ω  ←→  (⍟|ω)+0J1×12○ω
```

This definition is compatible, since for positive arguments, which have arcs of 0, the second term of the sum disappears. We can thus provide a logarithm for arbitrary nonzero complex numbers, and in particular for negative numbers. For a negative number, the imaginary part of its logarithm will be equal to pi, since the arc of a negative number is pi radians.

2. The dyadic logarithm function α⍟ω formerly signalled a Domain Error when given negative left or right arguments. Since dyadic logarithm is defined in terms of monadic logarithm (α⍟ω ←→ (⍟ω)÷⍟α), and since we can now give monadic logarithms of arbitrary nonzero numbers, we are thus able to give dyadic logarithms for arbitrary nonzero numbers to arbitrary nonzero bases, and in particular to negative numbers and/or to negative bases.

3. The power function α*ω used to signal a Domain Error for α negative and ω close or equal to a rational number having an even denominator. With complex numbers, such exponents are now permitted, and the result, in general, is not real. For example, ¯1*0.5 is 0J1.

4a. The dyadic circle function α○ω has had the domain of its left argument extended, as described above, to include the new values ¯12 ¯11 ¯10 ¯9 ¯8 8 9 10 11 12. An attempt to use any of these as a left argument used to give a Domain Error. These new left arguments are intended primarily for use in forming and decomposing nonreal complex numbers, but they are valid also for real right arguments.

4b. Several of the functions determined by particular left arguments of α○ω have had their domains extended to include more real arguments, as well as having been extended to complex numbers in general. These are ¯7 ¯6 ¯4 ¯2 ¯1 0○ω:

¯7○ω: Formerly ω had to be strictly between ¯1 and 1; now all arguments are valid except ¯1 and 1.

¯6○ω: Formerly ω had to be greater than or equal to 1; now all values are permitted.

¯4○ω: Formerly ω could not be strictly between ¯1 and 1; now only 0 is prohibited.

¯2 ¯1 0○ω: Formerly ω had to be between ¯1 and 1; now all numbers are permitted.

Second case--changes in value: There are two functions in this category.

1. The defining expression for the power function α*ω is:

```
      *ω×⍟α                                  (A)
```

but this could not be used hitherto for negative α, since logarithms of negative numbers were not defined. Nonetheless we gave an answer if ⍵ was (a close approximation to) a rational number *P*÷*Q*, with *Q* odd, since this in effect gave us an equation of odd degree *Q*, which we knew had one real root. We computed this real root by (¯1⋆*P*)×(|α)⋆⍵, and gave this as the value of α⋆⍵, even though strict application of expression (A) would have given a value somewhere in the complex plane off the real axis.

For example, referring to Figure 1, the elements of *Z* are approximations to the fifth roots of ¯32. One of these, ¯2, is a real number, and is in fact the value that used to be given for the expression ¯32⋆0.2. However, now that complex numbers are available we can use the defining expression (A) in every case where α is not 0. This ensures that functions such as ¯32⋆⍵ are (except for branch cuts) continuous over their entire domain. For example, the value of ¯32⋆0.2 is now given as 1.61803*J*1.17557. The desired continuity can be seen by noting the closeness of adjacent values in the following example:

```
      3 1⍴¯32⋆0.1999 0.2 0.2001
 1.61784J1.17465
 1.61803J1.17557
 1.61823J1.17649
```

2. The definition of the ¯4○⍵ function has been changed from (¯1+⍵⋆2)⋆0.5 to ⍵×(1-⍵⋆¯2)⋆0.5. Effectively, this doesn't change the function for positive arguments, but for negative arguments the value is now negative instead of positive. For example, the value of ¯4○2 ¯2 used to be 1.73205 1.73205. With the new definition the value is 1.73205 ¯1.73205.

Conversion

An array whose type is complex can be used with a function which requires a Boolean, integer, or real value as argument, if the complex array values are sufficiently close to Boolean, integer, or real values, respectively. The tolerance used in making this determination is not affected by ⎕*CT*. For example, 3*J*1*E*¯20 may be used to index the third element of a vector, or as left argument to replicate.

Acknowledgments

The complex-number extension to APL has been discussed in the APL community for many years. Paul Penfield, Jr., of MIT, played a leading role in elucidating the design problems, and made a comprehensive proposal for complex APL in [1]. The Sharp complex APL extension follows this proposal in all details except for the numbering of the new left arguments to α○⍵ and in output formatting. Professor Penfield also was kind enough to criticize our implementation in the course of its development. The implementation was designed and developed by Doug Forkes and Gene McDonnell.

This description benefitted from comments given by Arlene Azzarello, Paul Berry, Caroline Colburn, Doug Forkes, Ken Iverson, and Roland Pesch, of I.P. Sharp Associates.

E.E. McDonnell
I.P. Sharp Associates
220 California Avenue, Suite 201
Palo Alto, California
USA 94306

Reference

[1] Paul Penfield, Jr. Proposal for a complex APL, _APL79 Conference Proceedings_, ACM (1979) pp. 47-53.

Bibliography

Most high-school algebra texts cover the definitions of addition, subtraction, multiplication, reciprocation, and division on complex numbers. A.M. Gleason's _Fundamentals of Abstract Analysis_, Addison Wesley (1966) in Chapters 10 and 15 covers the construction of the complex number system and the definitions of the exponential, logarithm, power, and trigonometric functions on complex numbers. A more elementary discussion of much of the same material is given in Chapter 8 of K.E. Iverson's _Elementary Analysis_, APL Press, Palo Alto (1976). In Milton Abramowitz and Irene Stegun's _Handbook of Mathematical Tables_, Dover, N.Y. (1965) may be found definitions of many of the analytic functions on complex arguments. A detailed exposition of the algorithm behind the complex factorial and complex binomial functions may be found in Hirondo Kuki's "Complex Gamma Function with Error Control", _CACM_ 15 4 (April 1972). The paper by Paul Penfield, Jr., "Principal Values and Branch Cuts in Complex APL", to appear in the _APL 81 Conference Proceedings_, discusses the choices for locations of branch cuts, direction of continuity of the branch cuts, and values at the end of the branch cuts, for all the analytic functions requiring them. We are obliged to Professor Penfield for providing us with an early draft of this valuable paper.

Those interested in the discussion regarding the extension of the floor, ceiling, residue, and representation functions may read E.E. McDonnell's "Complex Floor", _APL Congress 73_, North Holland/American Elsevier (1973) for one set of definitions, and D.L. Forkes, "Complex Floor Revisited", to appear in the _APL 81 Conference Proceedings_, for a counter-proposal.