Integer-Only Functions
v.
Floating Point Numbers

Bob Smith
Sudley Place Software
Originally Written
31 Sep 2017
Updated
24 Feb 2020

Integer-Only Functions

The functions **GCD**, **LCM**, **Residue**, and **Encode** on FLTs/MPFRs employ some form of Comparison Tolerance (perhaps not exactly □CT) to decide when to terminate. These functions reference Comparison Tolerance in two ways: directly through some form of (Fixed System-wide?) Comparison Tolerance and indirectly as □CT through their reliance on Floor and/or Ceiling.

I view the primitives **GCD**, **LCM**, **Residue**, and **Encode** fundamentally as **Integer-only** functions. For better or worse and solely for the sake of datatype completeness, these functions have been extended to Fixed-Precision (FLT) and Multiple-Precision (MPFR) Floating Point numbers. It's true that the **Integer-only** algorithms used by these functions can be rewritten to provide an answer given FLTs/MPFRs, but often only after tweaking the algorithm to flush to zero certain small enough intermediate values – an instance of **Absolute Tolerance** and a problem the **Integer-only** versions of these algorithms don't share. The change made for the **FLT/MPFR-only** versions is done solely to allow the algorithm to terminate, but I believe, at the price of **User Confusion**.

Recently, there was a long thread in the J Programming Forum (**Puzzling result**) which revolved around the issue of the Residue of FLTs. The result was puzzling because the J Residue function with a particular right argument produced a different result depending on whether the small modulus (`196`) was stored as an INT or FLT.

One measure of success of our implementations is to check for what values of `N` will cause the expression `N|N×⍳10` to produce non-zero results with `⎕CT←0`? Try it for `N∈0.1..0.9`.

Rather than continue to tweak our FLT/MPFR algorithms to force them to provide correct-looking answers it's time to take a stand. It is not worth the confusion it introduces.

This means that my solution to how these four functions should behave on FLTs/MPFRs is to **ban non-integer arguments** (as in `DOMAIN ERROR`). That is, only INTs, MPIRs (whose denominator is one), or FLTs/MPFRs whose value is within **Integer Tolerance** of an integer should be allowed.

If you need to use small FLTs/MPFRs with these functions, scale (and possibly round) the arguments to integers, calculate the result on the integers, and scale back that result to the original datatype. If you find that you can't easily scale the arguments to integers, then that's a sign you are using the wrong algorithm.

This scaling could be handled via the Variant operator along with **Integer Tolerance** were the Variant operator applied to these functions to accept a FLT scaling factor (instead of `⎕CT`) as in

```
      ⎕CT←0 ⋄ ⎕PP←10
      0.7|0.7×⍳10
0 0 0.7 0 0 0.7 0 0 0 0
      0.7 (|⌸10) 0.7×⍳10
0 0 0 0 0 0 0 0 0 0
```

If you need to use very small FLTs/MPFRs with these functions, you are asking for trouble.

Bob Smith
bsmith@sudleyplace.com
304-707-7963