Hidden Traps: Floating Point Math.

Posted on **January 30, 2016**

As a life long programmer, I've programmed a lot of calculations over the years. Yet, I was recently astounded to hear expressed the opinion that "Computer computations are error free and flawless". The reasons that this is not (and never will be) the case are fairly deep and fundamental.

Mathematics routinely deals with matters of the infinite like the infinite digits in the value of the famous irrational number π (called pi). Engineers and computers on the others hand live in a world of the finite: finite memory, finite processing power, and finite time. The IEEE standards for computer arithmetic are marvels of ingenuity. They are also a compromise between the needs of speed, storage, and accuracy.

Let's see a real example and test the associative property of addition. This crucial, fundamental rule of math says that:

$$(a + b) + c = a + (b + c)$$

Now lets try to program this. We'll use Ruby for brevity, but the results would be the same for virtually any programming language that uses the computer's floating point hardware.

```
a = 1.0e20
b = -1.0e20
c = 1.0

puts "(a+b)+c = #{(a+b)+c}"
puts "a+(b+c) = #{a+(b+c)}"
```

The output of this little snippet of code is:

```
(a+b)+c = 1.0
a+(b+c) = 0.0
```

The two results *should* be the same but they're not. In one case, the use of a finite amount of memory to represent the numbers has caused an important "bit" of data to fall off the "end" and be lost. Addition in computers is *not* always associative.

Are there ways around this? Sure, you can improve things. In Ruby I can switch from Floats to Rational data, and this problem mostly goes away. The new program will consume more memory and take longer to run. Eventually, if the required precision becomes excessive, the program will slow to a crawl and/or run out of memory. Processing power and memory remain finite no matter how sophisticated the tools we use.

The moral: Know your math, yes, but realize that computers can't do math, they can only approximate it! So know the limitations of your tools too.

Yours Truly

Peter Camilleri (aka Squidly Jones)

This entry was posted in **Hidden Traps**, **Ruby** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=1652]** .