September 8, 2016

QUIZ & Floating point math

In this class:

- *Examples of why simple floating point operations lose significant digits.*
- *Some rules of thumb for good floating point codes*

Next class

Monte Carlo Methods
G&C – Chapter 3

Next next class

Review of probability and stats
G&C – Chapter 3

Rules of thumb for floating point

1. Use as much precision as possible (unless you want to study your problem).

   It'd be good if we had access to hardware quad precision for the "big-data" era!

2. Refactor to avoid catastrophic cancellation.

   It'd be good if we had access to hardware quad precision for the "big-data" era!

3. Avoid for unnecessary squaring.
4. Use logs for small probabilities.

A simple floating point computation

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

```
function enorm_simple(x)
n = 0.
for i=1:length(x)
    n = n + x[i]*x[i]
end
return sqrt(n)
end
```

… test in Julia …

A simple floating point computation

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

```
function enorm_simple(x)
n = 0.
for i=1:length(x)
    n = n + x[i]*x[i]
end
return sqrt(n)
end
```

Results

1. Can give Inf. when unnecessary
2. Inaccurate ?

Not-so simple floating point computation

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2} = |x_{\max}| \sqrt{\sum_{i=1}^{n} \left(\frac{x_i}{x_{\max}}\right)^2}$$

```
function enorm_guard1(x)
n = 1.; xmax = abs(x[1])
for i=2:length(x)
  ab = abs(x[i])
  if ab > xmax
    n = 1 + n*(xmax/ab).^2; xmax=ab
  else n += (ab/xmax).^2
end; end; return xmax*sqrt(n); end
```

… test in Julia …

Not-so simple floating point computation

$$\|\mathbf{x}\| = \sqrt{\sum_{i=1}^{n} x_i^2} = |x_{\max}| \sqrt{\sum_{i=1}^{n} \left(\frac{x_i}{x_{\max}}\right)^2}$$

```
function enorm_guard1(x)
n = 1.; xmax = abs(x[1])
for i=2:length(x)
  ab = abs(x[i])
  if ab > xmax
    n = 1 + n*(xmax/ab).^2;
    xmax=ab
  else n += (ab/xmax).^2
end; end; return xmax*sqrt(n); end
```

Results

1. ~~Can give Inf. when unnecessary~~
2. Better accuracy

Are these answers wrong?

Yes

We *could* have computed a higher-accuracy answer and didn't.

No

They are still highly accurate *(12 digits of precision!)* which is larger than needed for most applications.

But you never know when it isn't!

Catastrophic cancellation

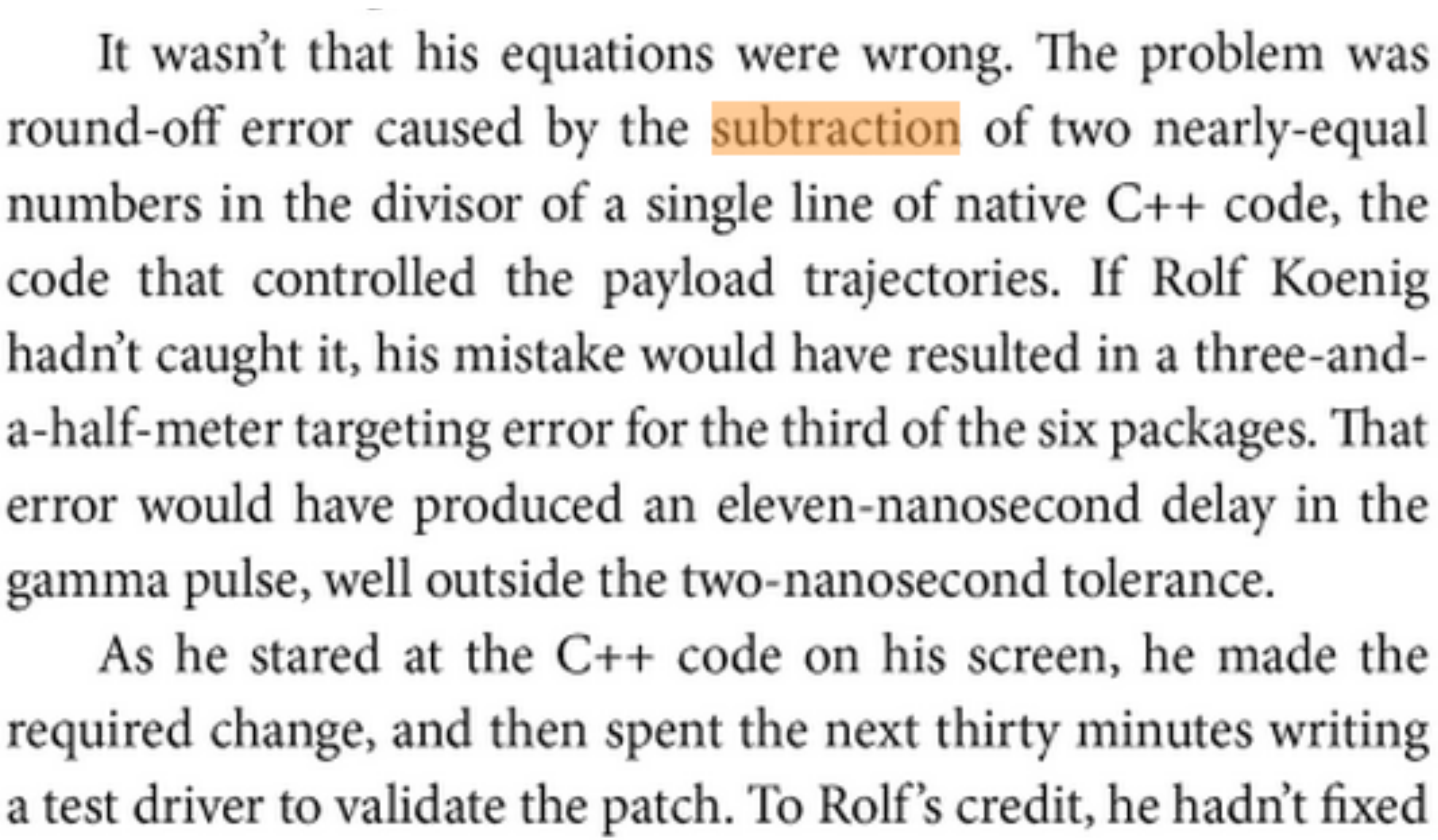
Suppose that we compute

$$z = a \ominus b \qquad a \approx b$$

$$\sum_{i=N}^{\infty} f_i = \sum_{i=1}^{\infty} f_i - \sum_{i=1}^{N-1} f_i$$

It's easy to lose ***all*** accuracy in z.

This happened to me about a year ago using someone else's code



It wasn't that his equations were wrong. The problem was round-off error caused by the subtraction of two nearly-equal numbers in the divisor of a single line of native C++ code, the code that controlled the payload trajectories. If Rolf Koenig hadn't caught it, his mistake would have resulted in a three-and-a-half-meter targeting error for the third of the six packages. That error would have produced an eleven-nanosecond delay in the gamma pulse, well outside the two-nanosecond tolerance.

As he stared at the C++ code on his screen, he made the required change, and then spent the next thirty minutes writing a test driver to validate the patch. To Rolf's credit, he hadn't fixed

Dead wrong by Richard Phillips, Chapter 55, Page 208

How to avoid cancellation?

Think!

$$(x^2 - y^2) = (x + y)(x - y)$$

$$f(x) = (x - t_1)(x - t_2) \cdots (x - t_k)$$
$$f'(x) = \sum_{i=1}^{n} \frac{f(x)}{x - t_i}$$

How to avoid cancellation?

Not always easy!

$$\sqrt{b^2 - 4ac}$$

Takes quite a few lines of Matlab code to do properly and involved proofs of floating point properties. (That have been their own research papers!)

How to avoid cancellation?

Not always easy!

$$\sum_{i=N}^{\infty} f_i = \sum_{i=1}^{\infty} f_i - \sum_{i=1}^{N-1} f_i$$

Needed to use a totally different derivation of the left-hand side.

e.g. Taylor series approximation around a different point.

The moral
Rules of thumb for floating point

1. **Use as much precision as possible (unless you want to study your problem).**

   It’d be good if we had access to hardware quad precision for the “big-data” era!

2. Refactor to avoid catastrophic cancellation.

   It’d be good if we had access to hardware quad precision for the “big-data” era!

3. Avoid for unnecessary squaring.
4. Use logs for small probabilities.