System Support for Scientific Computation

Notes by David Bindel from a lecture of W. Kahan and talk by Joe Darcy

Notes from April 9, 2001

Handout: Trichotomy vs. NaN

1 Recall from last time

Radix $\rho = 2, 10, \cdots$

$N = \text{number of "sig. digits" of radix } \rho$

Finite float is $\pm \rho^{\text{exponent}} \cdot m$, where $| \pm m | < \rho^{N+1} - 1$

Normalized number has $\rho^N \leq | \pm m | < \rho^{N+1} - 1$

Recall also Metropolis, Ashenhurst - significance arithmetic and the folk theorem which says things may be exponentially good/bad. (This has not been published... maybe you could embed it in some other publication; a list of floating point jokes, and this could be one of them.)

2 Zero is an exception in the world of normalized numbers

This causes a headache:

DEC VAX - all floats are normalized, so zero requires a special exponent.

... "Where you put the point only matters for purposes of explanation... but it does matter for that." ...

Since on the VAX, a zero is indicated by this special exponent, there are multiple encodings for zero. $-0$ was a "reserved operand" which behaved somewhat like a NaN.

On Cray-built machines, the minimum exponent was set aside for zero (some representations meant more for certain operations, but not all). The max exponent was set aside for infinities and indeterminates.

After WWII, all sides were stealing as much German technology as they could (including rocket technology). Von Neumann was part of the group looking at German computing.

The (possibly apocryphal) story is that Zuse, wanting an electronic computer capable of doing what his electromechanical machine did. He applied to the Lufts ministerium for a grant in December 1939. They said it was much too complicated, and there was no way it would be finished by ’42.

Von Neumann’s lesson from Zuse seems to have been not to do floating point (this is in part inferred from reading the collected works of Von Neumann, compiled by Taub), because it was so complicated. Recall, he claims that fixed point should be enough.
Zuse wanted to include correct rounding, infinity, and an indefinite symbol propagated like NaN. In this way there was no need to trap? Since the machine had limited memory, the lack of need for trap-handlers was a big advantage.

Von Neumann also introduced jamming:

[ 1. x x x x ] y y y y : where x’s significant, y’s or-ed

Why do this? The trailing bits could simply be chopped, but then things drift down. Indeed, this did happen in the case of the Vancouver stock exchange.

(See ”roundtalk.ps” from the pub/misc directory, linked from Pete Stewart’s web page: http://www.cs.??/?/stewart. His summary:

In 1982 the Vancouver stock exchange introduced an index with a nominal value of 1,000,000. After each transaction, it was recomputed and truncated to the third place to the right of the decimal. After 22 months, the index was 524.881. The true value was 1098.811.)

(Kahan’s version disagrees slightly; should do some checking to figure out which version is right.)

The stock price index was maintained on an IBM 360/370. After each transaction, it was updated:

\[(\Delta \text{ price})/(\text{number stocks})] + (\text{old ave}) = (\text{new ave})\]

After some years passed, people started to feel it was very low. When they recomputed, they found it had decayed a lot.

```
\begin{center}
\begin{tabular}{cccc}
\hline
 & & & \\
\hline
\end{tabular}
\end{center}
```

old ave

```
\begin{center}
\begin{tabular}{cccc}
 & & & \\
\hline
\end{tabular}
\end{center}
```

increment

```
\begin{center}
\begin{tabular}{cccccc}
 & & & & & \\
\hline
\end{tabular}
\end{center}
```

Over Many thousands of transactions per day, errors of about half an ULP added up!

They insisted the program would change.

How would you change it?

What did the programmer actually do?

```
\begin{center}
\begin{tabular}{cccccc}
 & & & & & \\
\hline
\end{tabular}
\end{center}
```

```
\begin{center}
\begin{tabular}{cccccc}
 & & & & & \\
\hline
\end{tabular}
\end{center}
```

```
\begin{center}
\begin{tabular}{cccccc}
 & & & & & \\
\hline
\end{tabular}
\end{center}
```

```
\begin{center}
\begin{tabular}{cccccc}
 & & & & & \\
\hline
\end{tabular}
\end{center}
```

+ \[\frac{1}{2}\]

Now the index is creeping up... because the trailing digits are not random.

If one digit ”hangs over”

2
Digit: 0 1 2 3 4 5 6 7 8 9
Error: 0 -1 -2 -3 -4 5 6 7 8 9

If two digits "hang over"
Digit: 00, 01, 02, ..., 49, 50, ..., 99
Error: 00, -01, -02, ..., -49, +50, ..., +1

So it turns out there is a bias. But it is not as large, and since it is positive, the exchange won’t likely mind.

Hotelling recommended rounding to nearest even for statistical reasons. (See Feller’s book)
The was also a study by Sweeney in the 1960s in the IBM System Journal (I missed on what..???
Even with rounding to nearest even, though, there will be drift. It will be slow; drift as a random walk, or by applying the central limit theorem to the sum of errors.
One idea: accumulate sums for some period, and then add them in and reset the period’s accumulation.
Another way, (Kahn, 1960, ACM; Also Muller in BIT):

2.1 Compensated Summations

The algorithm:

```
float OldAve, Increment, NewAve, C=0
.
.
Increment := ( PriceDiff / NewStocks ) + C
NewAve := OldAve + Increment
C := ( OldAve - NewAve ) + Increment
OldAve := NewAve
```

What happens?
We have effectively doubled our precision. So long as the increments are small, this is a foolproof way of suppressing (most of) the rounding error.

Where else might we do this? Perhaps numerical integration, or even quadrature:

\[
\frac{dy}{dt} = f(y) \\
Y(t) \approx y(t) \\
Y(t + \Delta t) = Y(t) + \Delta t F(y(t); \Delta t, f)
\]

\(F \approx \text{average of } f \text{ between } Y(t) \text{ and } Y(t + \Delta t)\)

What happens without this? Take too small a step and you seem to damage your solution! Old texts (and some modern ones) thus say there is an "optimal" step. If you keep a correction, this needn’t occur.

Another application of compensated summation is to add a slowly convergent series. Compensated summation is compared to other schemes in Higham’s book.

For some computers of the 60s and 70s, this didn’t quite work, and only God (and Seymour Cray) could seem to figure out why. See "A Survey of Error Analysis."

The idea of this trick is due to Gill, who figured it out in the 1940s (1947?). His thesis involved doing this on early electronic machines with fixed point (e.g. EDSAC) using very devious formulas which nobody but Stan Gill understood. He used these in Runge-Kutta-Gill methods.

Extensions of this idea lead to sparse floating point, which we will get to later.

3 There are lots of other ways to encode numbers

3.1 Continued Fractions

\[ A + \frac{1}{B + \frac{1}{C + \ldots}} \]

Useful for online arithmetic - save old computations and restart with higher precision if needed (... is that actually under continued fractions?)

3.2 Logarithmic Encoding

Let:

- \(x = \pm \rho^i\)
- \(\rho > 1 \text{ by a little (e.g. } 2^{2^{-30}} \approx 1 + \frac{0.99}{2^{63}}\)
- \(i \text{ an integer bounded (e.g. } |i| \leq 2^{63}\)
- \(x < 2^{63} \approx 2^{63} \text{ maximum.}\)

Now multiply and divide suffer no rounding error. What about addition?
given $I \geq J$
\[
\rho^I + \rho^J = \rho^I (1 + \rho^{J-I}) = \rho^{I+K}
\]
\[
\rho^K \approx 1 + \rho^{J-I}
\]
\[
K \approx \log(\rho) \cdot (1 + \rho^{J-I}) = \frac{\ln(1 + \rho^{J-I})}{\ln(\rho)}
\]

If the number of bits is small enough, we can find $K$ by table lookup. This was, in fact, used in our helicopter control scheme. $K$ is only trouble to compute on a subtraction: $\frac{\ln(1 + \rho^{J-I})}{\ln(\rho)}$ needs special attention when $J - I$ is small. But suppose you get it, and can round to get $K$ near as possible. This seems great.

We even have Associativity:

### 3.3 Exercise

$x \cdot (y + z)$ rounds the same as $x \cdot y + x \cdot z$ People even have patents on hardware for log arithmetic. Why doesn’t everyone do this?

You can’t estimate error on an add, so compensated summation isn’t guaranteed. But somehow, it still works (an attempt to prove it appears in Knuth:vol2) for the correction in RK schemes for tiny time steps. (exercise for the student: show why).

But nobody knows how to do sparse arithmetic (see Doug Priest’s thesis; also the 10th IEEE Symposium on Computer Arithmetic(1992). It’s also online somewhere (at ICSI)).

### 3.4 Exercise for the student

You can’t represent both 2 and 3 exactly in logarithmic encoding. And this is what kills logarithmic encoding, really. Whatever you might save, you’d lose it at the help desk explaining to users why they can’t write integers exactly.