1 Recall case study number 2

\( N + 1 \) integers representing times: \( 0 = t_0 < t_1 < \cdots < t_N \)
\( N + 1 \) reals representing cash flows: \( c_0, c_1, \cdots, c_N \)
(positive for incoming, negative for outgoing)

The internal rate of return is a root of the equation:

\[
\sum_{j=0}^{N} \frac{c_j}{(1 + x)^j} = 0
\]

Often have many cash flows the same (e.g. all but first, last cash flow for a lease or mortgage)
Newton’s iteration may go off to (infinity) (1 of fig) or converge painfully slowly (2 of fig). If it gets bumped past -1, it can get completely lost.
If there are lots of the same cash flows, we’d be inclined to use

\[
\sum_{j=c}^{m-1} \frac{(1 + x)^{-m} - 1}{(1 + x)^{-1} - 1} = 0
\]

We might evaluate this via \( \exp(m \cdot \ln(1 + x)) \). What goes wrong?

\[
\begin{array}{c|c}
1 & 00000 \\
+0 & 000xxxxx \\
\hline
1 & 000xx???
\end{array}
\]

During addition, may lose some digits to get \( 1 + \hat{x} \), where \( \hat{x} \) is a mediocre approximation to \( x \). Taking \( \ln(1 + \hat{x}) \) gives \( \approx \hat{x} \) (vs. \( \ln(1 + x) \approx x \)).

Now we multiply by \( m \) and exponentiate, possibly losing a couple more digits. And then we subtract 1, and incur cancellation.

It looks like computing \( ((1 + x)^m - 1 = \exp(m \cdot \ln(1 + x)) - 1 \) gives digits which are mostly accidents of roundoff!

Exercise for Student:

\[
\frac{(1 + x)^m - 1}{(1 + x) - 1}
\]
Computed as in preceding can lose no more than about half the significant figures.

The dangerous thing here is that the answers will look reasonable.

Citation: Marketing v. Mathematics, and discussion of some of the bizarre things with w/Quattro.

Hardly anyone is likely to diagnose this correctly. Either they won’t notice, or they will cynically declare the entire computation to be junk.

On Casio calculator:

\[
\left(\frac{1}{3} - 1\right) \times 3 - 1: \text{yields } 10^{-11} \quad \text{(indicates 12 digits)} \\
\left(\frac{1}{3} - 1\right) \times 3 - 1: \text{w "=" at each step indicates only 9..}
\]

What you see isn’t always what you get. Other anomalies may arise to confuse issues.

Example: on TI

\[
e \cdot \pi - \pi \cdot e = w \cdot 10^{-13} \\
\pi \cdot e - e \cdot \pi = w \cdot 10^{-13}
\]

same w .. including same sign!

What happened? When pushing a number and op on the internal stack, the op code overwrites the last digit of the number! (Discovering and demonstrating this irritated the IT guys who requested he demo the calculator at a talk.)

Another source of confusion is cosmetic rounding. (Casio calculators do this) ..

Lots of people think this doesn’t matter. How should arithmetic be done? What should our model be? We want advantageous rules of computed arithmetic. But the rules of exact arithmetic are hardly in dispute. So why approximate?

Can represent algebraic numbers exactly by a polynomial of minimum degree, appropriately normalized, and an indicator of which root. Transcendentals are our excuse for approximation (along with the fact that we get nasty exponential and doubly exponential costs in manipulating algebraic numbers.

Note that what is transcendental vs. algebraic get murky in some ways. The cube root of a Gaussian integer is algebraic, but the construction would require trisecting an angle. Tartaglia in the 19th century discovered that a cubic with all real numbers cannot necessarily have all roots computed by finitely many real +/-/*/.. . This is the "refractory" or "irreducible" case (see Math H85 notes).

In general, given two representations of a real, there is no uniform algorithm to test equality. It is a conjecture that for arbitrary transcendentals, the problem is undecidable.

So there’s no choice about approximation. We need rules about how to approximate, then... know rules, so that we can reason about them.

Example: Cray 1 had noncommutative multiplication. This was eventually fixed when some government agent refused to buy a machine with noncommutative multiplication. How did it happen?
Crays lopped off the stuff that doesn’t appear, in some sense. The way they did so broke symmetry. Exercise for the student: Argue convincingly why (or why not) noncommutative multiplications should matter.

Don’t know where you’d find that information authoritatively any more. Call this type of story motivation to take good notes.

Back to the original problem. A number of challenges.

- How to even compute the expression.

- Where to start. Newton or some other iterative method? (Sometimes, there are multiple roots, and it’s not always the case that one is the only non-spurious case.)

- When to stop? This can be more interesting than you might think. How to tell what interest rate to use in the case of multiple root? See which way the roots moved when perturbed. Thus we want some monotonicity in order to observe the direction of perturbation. This makes the accuracy requirement rather higher than obvious.

The purpose of computation is to draw inference about the ”real” world... so it should at least behave qualitatively like reality.

This is why the HP12c carries out computations to 13 digits, though the data is only to 10.

What can be computed in floating point? It is easy to be misled... Blum, Shub, Smale: $1 + \epsilon$ model Can do lots with this model, but some machines (Crays) don’t even manage that. For Crays,

$$c = a(1 \pm \epsilon) \odot b(1 \pm \epsilon)$$

If you believe in this model, you can be off by a factor of the radix!

\[
\begin{array}{c}
1 & 000000000 \\
-0 & 9999999999 \\
0 & 0000000001 : \text{Correct} \\
0 & 00000000010 : \text{Cray throws away the last digit.}
\end{array}
\]

This model doesn’t damage most computations, but it hurts some.
Example: Ming Gu's divide-and-conquer code Computes \( \frac{f(x)-f(y)}{x-y} \). Result may be in error if \( x, y \) are \( \cdots \) but the computation turns out to be exact for representable \( x, y \). On Crays, this computation could be off by a factor of 2. Dirty fix was

\[
\begin{align*}
x &:= (x+x)-x \\
y &:= (y+y)-y
\end{align*}
\]

to "lop off" the last bit on Crays (does nothing on most binary machines).

**Theorem 1** \( \frac{1}{2} \leq \frac{x}{y} \leq 2 \) implies \( x - y \) exact in IEEE.

Assignment: Prove it!

A similar thing occurs when working on an ODE solver. Want to show rounding error subsumed by LTE... on a Cray you have to do that oddly.


Example: In \( 1 + \epsilon \) model, cannot get \( x + y - z \) to high relative accuracy. Not unreasonable if data uncertain to begin with.

But what if \( x, y, z \) are physically meaningful?

\[
A = \frac{1}{4} \sqrt{(x + y + z)(-x + y + z)(x - y + z)(x + y - z)}
\]

But... even though \( x, y, z \) may not be known exactly, the area is well-determined by the data when angles are acute. The formula can lose accuracy according to the \( 1 + \epsilon \) mode, but works fine on IEEE machines.

What about the volume of the tetrahedron? Nobody (else) seems to have figured out how to compute it as accurately as deserved. We could compute with simulated extra-precise arithmetic, but there is another way. However the algorithm is so unreasonably devious that we can't expect most mortals to find it.

This is an exercise in error analysis designed to show that we shouldn't do it. We should instead make it unnecessary.

Sometimes it's a good idea to use more precision than the data "deserves". But who determines how much data deserves?

Thesis: C, FORTRAN, Pascal, Java, ... are bad languages for doing casual floating point computations when the user is not an expert.