From Henry Briggs to Modern Calculators

Harold Thimbleby

Gresham Professor of Geometry

November 29, 2001

Centuries of calculators

The abacus was invented somewhere around 200AD, though there is some controversy exactly when — and exactly what counts as an abacus. Various adding instruments were devised over the centuries, but no real developments occurred until 1614 when John Napier invented first his bones and then his logarithms, although they weren't quite what we'd call logarithms today: they were harder to use. His logarithms allowed people to multiply easily, by adding. In 1624 Henry Briggs (the first Gresham Professor of Geometry) invented common logs, inspired by Napier's ideas — they are often called Briggsian logarithms in his honour.

In 1642 Blaise Pascal, while still a teenager, invented a mechanical adding machine (illustrated on the right). He sold about 50 of them. About 40 years later, in the 1680s Gottfried Leibnitz devised ingenious gears that could be used for multiplying. His gears became the mainstay of calculating machines until the late nineteenth century when Charles Babbage designed (but didn't finish building) his various programmable difference engines. You can see some very nice replicas of Babbage's work in the London Science Museum.

The 1930s saw the widespread adoption of mechanical calculators, such as those made by Brunsviga (illustrated on the left). With the developments in electronics in 1970s, electronic calculators mostly copied the design ideas of the earlier mechanical ones.

Ten years later, when desktop computers could do almost anything, desktop calculators merely imitate the earlier electronic ones. Their "look and feel" is much the same (the talk will give some demonstrations). Yet present electronics can do a lot better than just imitating mechanical calculators.

Problems with today's handheld calculators

I got volunteers at a Royal Institution lecture to try some simple sums, 4×–5 and 10+20%, on a range of modern handheld calculators. For the sum 4×–5, we got results –20 (called out from the audience by two separate volunteers), –1 (called out three times), +20 (once). For 10+20%, we got results 12 (twice), 12.5 (twice), 22 (twice), 150 (three times). Some people had calculators with them, like the Sharp EL-546, which do not have a % key.

Note how only one of the volunteers got a sensible answer for 4×–5: it ought to be –20. There was a wide range of answers for 10+20% sum (most people would agree 10 plus 20% means 12, since 20% of 10 is 2, and 10+2 is 12).

Some of the audience may have been shy and not reported their results. Of course it's possible some volunteers made mistakes pressing keys or on reading the answers: but this can only really explain the +20 answer, as someone making random errors are very unlikely to get the same results as other people. We conclude that calculators are not as reliable as we might have thought! It's not so much us who do not understand percent as, perhaps, the manufacturers can't decide what it should do.

The Gresham lecture handout describes other problems that can be tried out on your own calculators, or on the calculators handed out at the lecture. These things are not as easy and straight forward to use as you may have imagined. Calculators do not do mathematics as we understand it.

We now turn to finding out why, and what can be done about it.

Compiling: Expressing your problem so the calculator "understands it"

There is something we take for granted when entering sums into a calculator: we have to "compile" the sum into the specific instructions the calculator requires, with the right key strokes in the right order. First, take an obvious and simple sum, such as what $x = 2+3$? What we press to find out the answer is fairly obvious:

[2] [+] [3] [=]

For a more complex sum, such as what x ensures $2+x = 5$, the key pressing is only a bit more complex:

[5] [–] [2] [=]

You can see how + has had to be changed to –. These two examples seem so simple, "compiling" hardly seems to deserve further comment — but then these are examples we could do in our heads. Things rapidly get more complex as the sums get a bit more realistic. For more sophisticated sums, the nature of compiling is more evident. Consider finding the x such that $2^x = 5$. Here's how you'd find out using various calculators:

`[LOG]` 5 / `[LOG]` 2 = *on the Sharp EL-531VH*

5 `[LOG]` / 2 `[LOG]` = *on the Hewlett Packard HP6S*

5 `[LOG]` 2 `[LOG]` / *on the RPN Hewlett Packard H32SII* (see below for more on RPN)

Calculators can do much more than simple sums. Many calculators have a memory, so let's see how to compile "Store the displayed number in memory." Here's how to do it on a variety of calculators:

X	M	— *on the Hewlett Packard HP6S*

STO	M+	— *on the Sharp EL-531VH*

M+	–	MRC	M+	MRC	— *on the Casio HS-8V*

The last two examples can't be left without comment: they are bizarre. The Sharp makes it look like you are adding to the memory; the Casio (which is typical of many simple memory calculators) doesn't have a "store to memory" key, and the user has to work out — with considerable ingenuity — how to store to memory. It'd be easier and more reliable not to use the function! *What are the buttons for*? I'd suggest many buttons are there to make the calculators attractive, to sell the calculators, *not* to help people do sums.

Some calculators provide complex numbers. How would you compile the (simple) complex addition (1+2*i*) + (3+4*i*) so a calculator would do it? On the Hewlett Packard HP20S you'd have to press: `[MODE]` 1 `[PRGM] [CLPRGM] [LOAD] [LN] [PRGM]` 1 `[INPUT]` 2 `[XEQ] [A]` 3 `[INPUT]` 4 `[RS]` to get the imaginary part (6 in this case) followed by `[SWAP]` to see the real part (4). Perhaps it's worth noting the HP20S is sold in a box that says, "Algebraic entry using the [=] key. All the essential math and trig functions … Built-in program library including: … complex number operations." The simple sum required sequence of button presses which isn't very obvious, and is unlikely to be remembered by anyone. It certainly isn't algebraic in any normal sense of the word.

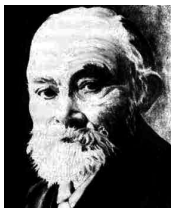
In contrast the Casio *fx83WA* has a complex mode, and when set to it you can at least use the + button to add complex numbers. In fact, in every case on every calculator it'd be easier to do complex sums explicitly (e.g., in this case add 1+3 and 2+4 separately), which would be more reliable and the user would be certain they'd done the right things. In short, complex numbers are a gimmick — particularly because you are restricted to just the four basic arithmetic operations (+ – × ÷), so working out the classic $\sqrt{-1}$ (which is *i*) or even $e^{-i\pi}$ would be out of the question.

Certainly school-level calculators seem to be covered in useful functions, such as sin, cos, logarithms and so forth — often running into hundreds and in some cases thousands of functions. How good are these A level functions?

Let's try compiling (i.e., finding out the value of) $10^{-\pi}$, which should be around 0.0007. On the Sharp EL-531VHB we might try 10^x – π first, but this means "Error." Since 10^x 2 ± gets 0.01 correctly, this suggests we should try 10^x π ± (three button presses) but this gets "Error." Curiously, pressing the three keys 10^x – 2 can get *anything at all*, depending on the last answer!

These problems, which are by no means limited to this Sharp model, arise because π is not being treated like a number or even like a memory. For example if we store π in memory A, by pressing π `[STO]` A we should now be able to use A for π since they are the same value: pressing π or `[RCL]` A should get the same results. Yet we find that 10^x `[RCL]` A ± really does mean 0.00072, even though 10^x π ± was an error. So the name A works differently to π on this calculator. Compiling is difficult; it's even harder when the calculators are inconsistent.

Finally, there is no calculator where finding the *x* to solve *x*! = 5040 is obvious. Compiling is not necessarily a trivial process, as this example shows. Nevertheless the other examples show that compiling has become a pretty arbitrary process, made more complex by the arbitrary variations amongst and within calculators. Some calculators, notably the Casio HS-8V (used in an example above, and which is typical of many basic calculators) make storing numbers in memory bizarre and — ironically — unmemorable.



To put it briefly, modern calculators ignore the insights of mathematicians, such as Gottlob Frege (who wrote the *Foundations of Arithmetic* in 1884). His nineteenth century contribution to the foundations of mathematics was to have a very clear idea of what variables (such as π and A) stand for, and how expressions should have clear values. Much of what he said was formulating what mathematicians already knew: expressions have to have clear meanings if we are to know what they mean.

The evidence from modern handheld calculators is that they do not do mathematics as we know it; worse, given that people like Frege have been quite explicit about what mathematics is, the manufacturers are ignoring the well-established foundations of the subject they are purporting to support with their products. (The technology itself has no such limitations.)

One important idea of Frege's is called *referential transparency*. Names should mean the same things. Thus in *a*+*a*=20, both occurrences of the letter *a* mean the same thing, in this case 10. A major failing of calculators is that they do not have referential transparency, and thus they make doing mathematics very difficult. Further, expressions should behave like the numbers they represent: thus if 2+2 means 4, you can substitute (2+2) anywhere 4 can be used, and you can get the same results. These ideas are very fundamental, and almost so obvious they should not need stating. Yet on almost every calculator, if 2+2% means 2.04, then you won't be able to write 1+(2+2%) to get 3.04. Try it!

The Sharp calculator's different treatment of π, memories (like A), numbers, and values like (2+2%), is a clear example of referential opacity: failure to support referential transparency. Frege would claim that calculators like this *cannot* be doing mathematics. Where they seem to be doing mathematics, they are not reliable — quite a damning criticism given that people using calculators presumably do so because they don't know the answers.

In passing, it's worth noting Jan Lukasiewicz invented a simple and consistent mathematical notation in the early twentieth century. His name is so unpronouceable that his notation is commonly called Polish Notation. Hewlett Packard use "reverse Polish notation" (abbreviated RPN) on their HP32SII: it's main advantage is that it makes — or should make — compiling much more straight forward. *Except* that on Page 2–16 on the HP32SII owner's manual there is a warning: "…be sure that no more than four intermediate numbers (or results) will be needed at one time…" In other words, just when somebody has worked out a straight forward and consistent approach, the manufacturers make a calculator with arbitrary restrictions. In turn this makes compiling even harder and more error-prone for calculator users. RPN is no solution if it is inadequately implemented.

To be more positive about RPN and the HP32SII in particular: at least it gets % right. 2+2% or 10+20% are treated as numbers in RPN, and the HP32SII implements this correctly.

John Backus

If it wasn't surprising enough that calculators appear to ignore standard mathematical knowledge, it is even more surprising that they ignore relevant progress in computer science. Way back in 1954 John Backus and his team implemented FORTRAN, what might be regarded as the first modern programming language (it's still in widespread use). To get a computer to understand FORTRAN, you have to work out how to get a computer to automatically compile expressions like: 2+3×4 and (2+3)/(4–5). The whole point of a programming language like FORTRAN is to get the computer to compile problems for the human. Backus's work is well known, and all aspects of it are now standard mainstream stuff. No undergraduate computer scientist would pass their degree without knowledge of the relevant techniques.

Although Backus and his team took 18 people-years working on their FORTRAN project, using modern tools the same work could be done in a couple of days by any competent individual — and it would probably be done better because modern tools are so good. Today's automatic tools make this sort of work really easy. With the classic contributions from people like Noam Chomsky and Chris Strachey (in the 1950s and 60s), everything is now very well understood and common knowledge amongst professionals.

The main point is that with basic computer science you can remove the burden of compiling from the user. When programming, this is what we all expect. Modern programming languages like Java do a huge amount of compiling to make the programmer's like easier and more predictable. Hence it is all the more surprising that handheld calculators seem to have missed this fifty-year old development. Users of calculators are still expected to compile for themselves, rather than let the calculator do it for them (which it could).

To make everything easy

We've seen that modern handheld calculators are unnecessarily difficult to use; indeed they do not do mathematics as commonly understood. Let's get back to 1880s Frege, and treat maths with referential transparency, so that things mean what they say in all contexts. In particular, let's use the power of microelectronics to avoid all problems of *human* compiling. After all, calculators are used by people who are not usually in the position to notice errors in their use of the mathematical aids. ***Manufacturers have a responsibility to make calculators fit for use.***

The lecture provides an interactive demonstration of a new style of calculator that solves *all* of the problems discussed above. One of the calculators demonstrated in the lecture was developed by an undergraduate student (Will Thimbleby): the new solutions are not difficult, and are not beyond the resources of manufacturers.

Many examples and a full explanation of the calculator are given in my reference "A True Calculator" (which is also available from my web site: see below for details).

Teaching maths: Sink or swim?

This Gresham lecture relates to the various debates on using calculators in teaching maths. Current commercial calculators do not "do" mathematics in any reasonable sense; they are as primitive in their approach as the 1930s mechanical calculators, and they have not overcome the mechanical limitations they inherited.

We should be teaching how to use of calculators, given that they are ubiquitous and aren't going to improve any time soon. People have to get reliable results with these odd devices. *And* we should be teaching mathematics and how to calculate.

We have to teach calculator use, but this should never be confused with teaching mathematics. Learning to use a modern calculator is not maths, but learning to cope with bizarre design.

We should be teaching people to be more critical of modern technology. Technology won't improve if all we do is teach children to lean over backwards to use bad technology. We should be teaching a critical attitude (in the long run we'd then have fewer dot-com busts). At university level, we should be teaching students on computer science courses to use the knowledge they have learnt to improve the world, rather than to help produce cheap and substandard products. Computer science is technology plus mathematics, not just technology worship.

Let's make an analogy. We should teach people to swim, because it can save their lives. But most of us prefer to use boats, because they are a lot more convenient. But if boats were as bad as calculators, we'd all need to be able to swim. See my article "Smart to be Simple" for more on this.

Conclusion

Understanding maths makes technology work better. Not understanding maths creates oppression: we need calculators all the more to help us and, further, we are unable to recognise their problems. In turn, this increases our dependency on them: we end up with the culture we now live in, where the National Curriculum requires children to learn unnecessary and arbitrary features of calculators — and never know something better is possible. Children then grow up and some even go on to work for manufacturers: thus the cycle is repeated.

At least in the area of mathematics, it is quite clear what technology should be doing: namely, supporting calculation. We have shown that modern calculators fail to do this. We have shown that calculators can be built that work properly.

What are you going to do about it?

Further reading

A. V. Aho & J. D. Ullman, *Foundations of Computer Science*, W. H. Freeman, 1995.

A. Kenny, *Frege*, Penguin Books, 1995.

H. Thimbleby, "A New Calculator and Why it is Necessary," *Computer Journal*, **38**(6):418–433, 1996.
`http://www.cs.mdx.ac.uk/harold/srf/allcalcs.pdf`

H. Thimbleby, "A True Calculator," *Engineering Science and Education Journal*, **6**(3):128–136, 1997.
`http://www.cs.mdx.ac.uk/harold/srf/truecalc.pdf`

H. Thimbleby, "Smart to be Simple," *The Times Higher Education*, Multimedia Supplement, **1371**:15, 12 February 1999.
`http://www.cs.mdx.ac.uk/harold/srf/stbs.html`

H. Thimbleby, "Calculators are Needlessly Bad," *International Journal of Human-Computer Studies*, **52**(6):1031–1069, 2000. `http://www.cs.mdx.ac.uk/harold/srf/hucalc.pdf`

… see also the problem sheet handed out at the lecture.