A Quartic with a Thousand Roots

John Mills, David Tall & Michael Wardle

Mathematics Education Research Centre
University of Warwick

Introduction

In a first year university course on programming and numerical methods (in BBC BASIC), it was decided to give the students the quartic equation:

$$x^4 + 2.88x^3 - 19.23x^2 - 36.11x + 91.56 = 0$$

to solve numerically. One root of this equation is an integer — to help the students get started — but a second root is close to the first to make it more interesting. Initially we did not realize quite how interesting this would prove to be.

Students were allowed to use any methods at their disposal to find all four roots. At this stage we did not realize that a quartic might have more than four places where it was zero... What follows is a sequence of investigations which we followed to seek the roots using a computer. The results were confusing and intriguing, leading to the realization that, on a computer, this quartic can have more than a thousand roots.

Numerical Methods

It is relatively simple to type the quartic in (BBC) BASIC in the form

```
1000 DEF FNf(x)=x^4+2.88*x^3–19.23*x^2–36.11*x+91.56
```

and to perform a search such as:

```
FOR x=–10 TO 10 : PRINT x,FNf(x) : NEXT
```

This shows that $f(x)$ is positive for x=–10, –9, –8, –7, –6, –5, (almost) zero for x=–4, positive again for x=–3, –2, –1, 0, 1, then negative for x=2,3 and positive for x=4 and above. The roots between x=1 and x=2 and between x=3 and x=4 give no problems and succumb to almost any numerical method of calculation (they are 1.64955497 and 3.469734141). But clearly something is interesting around x=–4. Substituting x=–4 into the equation and calculating the value of the quartic by hand gives zero, so x=–4 is a root. But PRINT FNf(–4) gives 5.96E–8. The errors in calculation are serious enough to give a significant error.

With roots so close together, clearly a bisection method will be difficult to operate because we must first find one place where $f(x)$ is positive and one where it is negative and the latter are initially hard to find. But the Newton Raphson method is successful. Starting at x=–5 and using the iteration replacing x by $x–f(x)/f'(x)$ soon homes into –3.999999883, whilst starting at x=–3 gives –3.99292951. To four decimal places these are –4.0000 and –3.9929.

However, Newton Raphson is usually incredibly accurate, so why was the root at x=–4 given with an error in the 7th decimal place?

Published in *Mathematical Gazette*, **74**, 339–346 (1990).

To gain more insight, two more functions were typed into the computer, the derivative:

```
2000 DEF FNd(x)=4*x^3+3*2.88*x^2–2*19.23*x–36.11
```

and a function to print out a root:

```
3000 DEF FNroot(x,error)
3010 LOCAL k
3020 REPEAT
3030 k=x: x=x–FNf(x)/FNd(x)
3040 UNTIL ABS(k–x)<error
3050 = x
```

This repeats the Newton Raphson iteration until the last two approximations differ by a specified error. For instance

```
PRINT FNroot(–5,10^–10)
```

iterates from x=–5 until successive iterations differ by less than 10^{-10}, returning the value of x found.

The simple command

```
FOR x=–10 TO 10:PRINT ; x, FNroot(x,10^–10) :NEXT
```

gives the sequence of roots found starting from various points as:

–10	–3.999997587
–9	–3.999996085
–8	–3.9999977
–7	–3.999998967
–6	–3.999999247
–5	–3.999999883
–4	–3.999998013
–3	–3.999292951
–2	–3.999289413
–1	–3.999996761
0	3.469734141
1	1.64955497
2	1.64955497
3	3.469734141
4	3.469734141
5	3.469734141

with all the other starting points from x=6 to x=10 also ending up on the root x=3.469734141. Note that the two positive roots are registered in exactly the same form each time they appear, but the roots near x=–4 are slightly different every time.

Computer Graphics

Perhaps a picture would help. *Superzoom* (the *Supergraph* program with a zoom feature) drew the graph impeccably, with two roots near together at x=–4 and two positive roots clearly visible (figure 1).

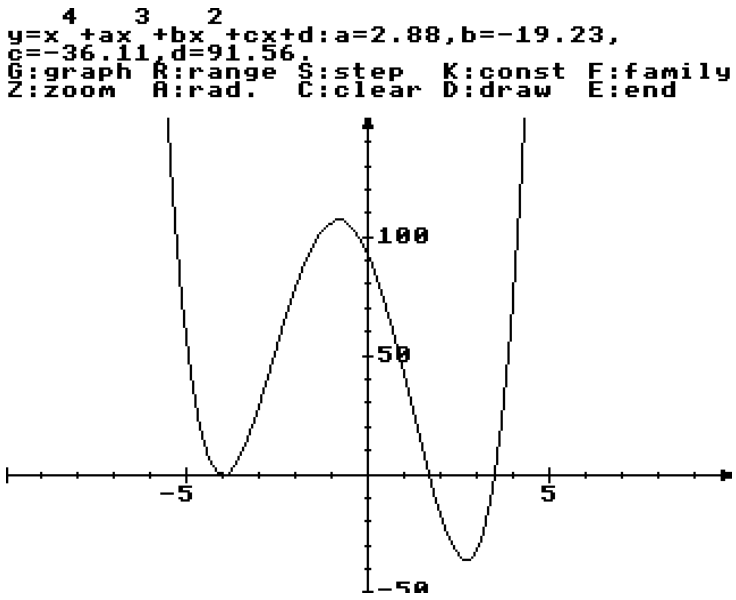

figure 1

Zooming in, centred on (–4,0) shows the graph to be very flat here, and only by taking a smaller *y*-range (and hence stretching the graph vertically) does the picture reveal the two close negative roots (figure 2).

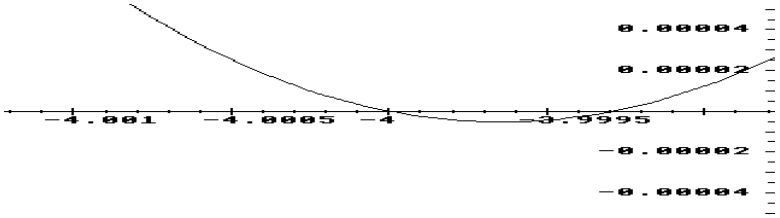

figure 2

However, zooming in closer and closer, on the root –4, keeping the same proportional scales, reveals an unexpected picture. Superzoom joins up the points plotted and has an "asymptote search routine" to check where the graph changes direction so that it can attempt to identify asymptotes. This graph initially almost brings the routine to a halt because the graph is bobbing up and down so much that the asymptote search routine is constantly being triggered. After this had happened a few times (causing the graph to break up on the left), the routine was switched off and the number of points plotted was increased to give the picture (figure 3).

figure 3

The current versions of Supergraph on the Master, Nimbus and Archimedes computer have a "DOTplot" feature, which makes a number of passes across the interval, the first pass plotting the centre point, the second those at the quarter and three-quarter marks, the third at 1/8, 3/8, 5/8 and 7/8 of the interval, and so on. The *n*th pass fills in 2^n–1 equally spaced points and does not join them. Initially there is a speckled effect, filling

out as each pass plots twice as many points. After half a minute or so on a BBC computer, ten passes plot more than a thousand points to give an interesting picture (figure 4).

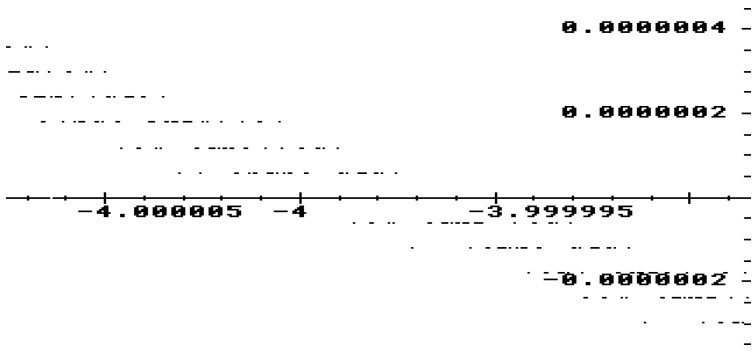

figure 4

Nearly twenty minutes later, after fifteen passes with over 32000 points, the picture has filled out to appear as a collection of horizontal line segments. As new points are plotted they jump wildly from one segment to another (figure 5).

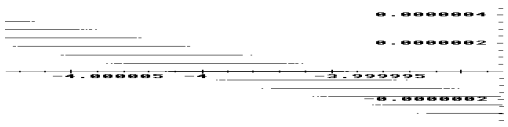

figure 5

What is going on?

The clue lies in the actual value of $f(x)$ near $x=-4$, which is shown as approximately $0.00000005=5\times10^{-8}$. This is close to a power of 2, namely $2^{-24}=5.96\times10^{-8}$. The values of f(x) are therefore jumping up and down in steps of size 2^{-24}.

Near $x=-4$ the monomials in the quartic are fairly large. For $x=-4$ we have

$$\begin{aligned} x\text{^}4 &= 256 \\ 2.88*x\text{^}3 &= -184.32 \\ -19.23*x\text{^}2 &= -307.68 \\ -36.11*x &= 144.44 \\ 91.56 &= 91.56. \end{aligned}$$

Adding up these numbers, each with a tiny error in x, can lead to a substantial error in $f(x)$. The numbers involve are of the order of 2^8, so the error step shown on the graph has an effect of the order of

$$2^8\pm2^{-24} = 2^8(1\pm2^{-32}).$$

As BBC BASIC works keeps a number in memory in the form $2^k\times m$ where m has at most 32 binary digits, the errors shown on the graph are precisely the errors given by a change of one or more units in the least significant binary place. Using a computer language with the accuracy limits of BBC BASIC therefore will lead to numerical features of this kind.

Another numerical search

To get more information we set an Archimedes computer going printing out x and the sign of $f(x)$ from just before –4 to just after, in steps of 10^{-8}. (In retrospect, perhaps the step would have been better as a power of 2.) We found the value of $f(x)$ going almost randomly positive, negative and zero. Counting every sign change, we found that this quartic seems to have over a thousand roots...

Computerized Symbolic Manipulation

If approximate methods give errors which mount up and eventually give such problems, the question is whether there are better methods on the computer that will give accurate results. Exact symbolic methods are already with us. The software Derive (which runs on IBM compatible computers and therefore will run on both Nimbus and Archimedes in IBM emulation mode) performs exact arithmetic and can solve polynomials up to quartics algebraically.

The quartic was entered as

```
x^4+2.88x^3–19.23x^2–36.11x+91.56
```

and the software immediately printed it prettily as:

```
 4      3       2
x +2.88x –19.23x –36.11x+91.56.
```

The “factorize” command was selected, first to factorize into rational form, which it revealed after 79.3 seconds as

$$\frac{(x+4)(100x^3-112x^2-1475x+2289)}{100}$$

It was then factorized further as radicals, taking 125.9 seconds to give a product of four linear
factors:

$$(x-4)\left(x-\frac{\sqrt{113761}\cos\left(\frac{\operatorname{atan}\left(\frac{29158759\sqrt{767916251}}{691124625900}\right)}{3}+\frac{\pi}{6}\right)}{75}-\frac{28}{75}\right)\ldots$$

$$\ldots\left(x-\frac{\sqrt{113761}\cos\left(\frac{\operatorname{atan}\left(\frac{29158759\sqrt{767916251}}{691124625900}\right)}{3}+\frac{5\pi}{6}\right)}{75}-\frac{28}{75}\right)\ldots$$

$$\ldots\left(x-\frac{\sqrt{113761}\cos\left(\frac{\operatorname{atan}\left(\frac{29158759\sqrt{767916251}}{691124625900}\right)}{3}+\frac{\pi}{2}\right)}{75}-\frac{28}{75}\right).$$

In this form the meaning of the results is hardly transparent, but there is an “approximate” command which, when selected, gives the display:

0.0000000000138733 (x + 4) (2805 x + 11218) (4230 x –14677) (6075 x – 10021)

This shows the roots to be approximately

–4, –11218/2805, 14677/4230, 10021/6075,

or

–4, –3.9993, 3.470, 1.650.

As an alternative, we selected the approximate mode of calculation with 20 decimal places of accuracy to factorize the quadratic. After 43 seconds we were rewarded with the factorization:

```
(x–3.4697341409416201297)
(x+3.9992891108740135650)
(x–1.6495549699323934352)
(x+4).
```

We had no idea how the computer made these calculations, but one of us calculated the sum of the roots, to find that this gave the coefficient of x^3 in the polynomial correct to within 1 in the nineteenth decimal place. At least we had some evidence that the computer might be giving us an appropriately accurate answer.

More Graphs

We decided to draw the graphs using **Derive**. The software will make exact calculations using rational arithmetic, but when it comes to drawing graphs, it must approximate the result to calculate the position of the pixels on the screen. We began plotting the quartic over the range -4.02 to –3.98 using the default number of digits used in exact mode. The results surprised us, with errors clearly visible as kinks in the curve (figure 6)

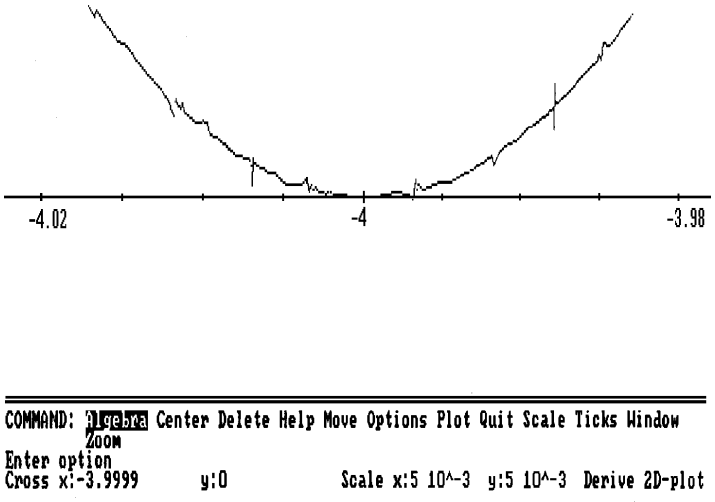

figure 6

Zooming in closer to the range (–4.004,–3.996) gives a picture with even greater fluctuations. There are clearly wild numerical errors in calculating the approximate values of the fractions (figure 7).

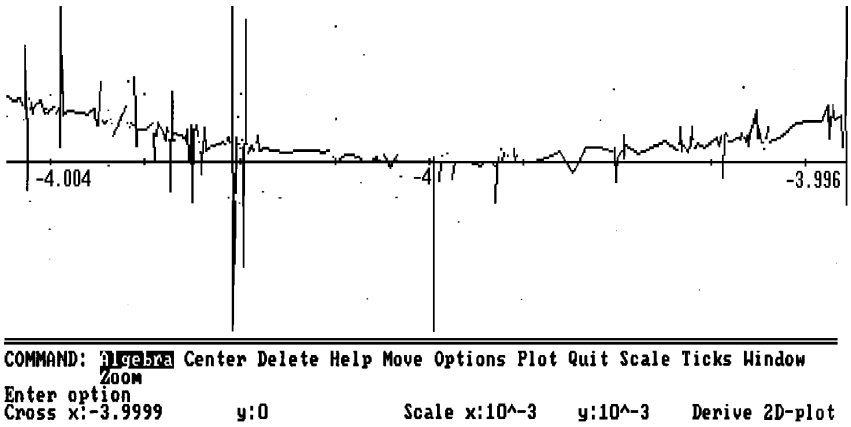

figure 7

There is an option to plot a *y*-value for every *x*-pixel, and this was selected in an attempt to give some clue as to the magnitude of the errors involved. The picture shows the general trend of the graph, but with pixels sprayed around in a cloud of errors (figure 8).

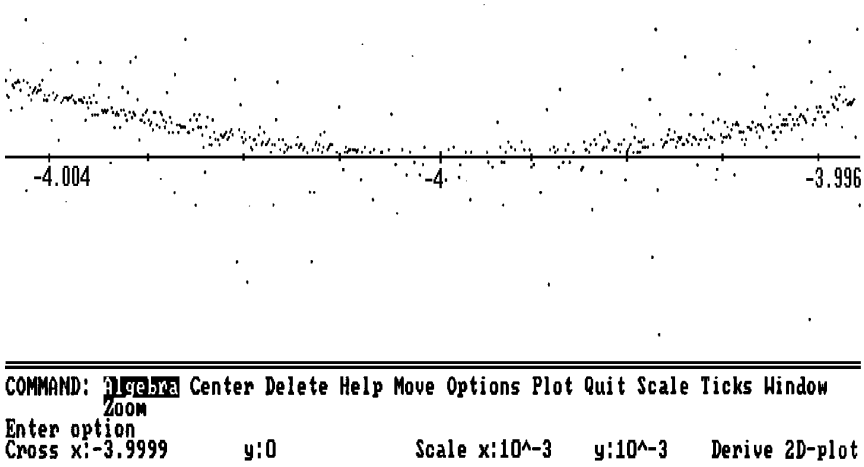

figure 8

Changing the precision of the digits from the default 6 to 7 removes all the scattering and produces a smooth-looking curve (figure 9). But on checking the precision, we found the system had reset the precision to 10 digits (perhaps it works in byte-sized chunks) so we were no longer sure what this precision meant.

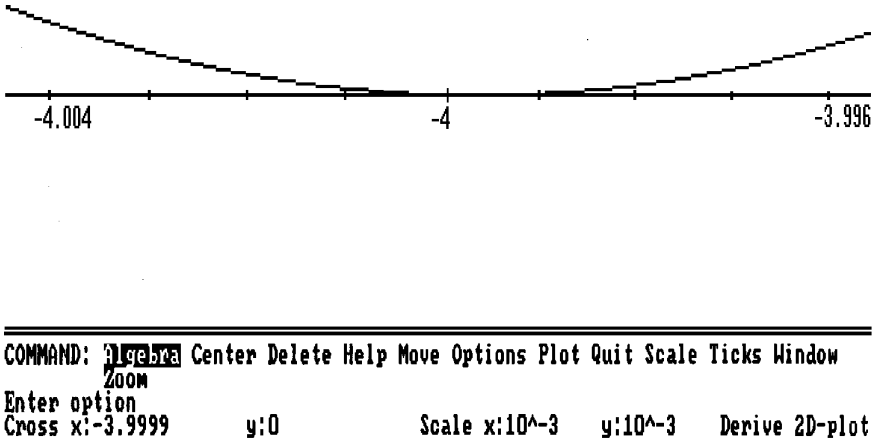

figure 9

Returning to 6 digit accuracy and zooming in a stage further totally flabbergasted us. The graph was drawn as a smooth curve as if all the errors had now disappeared!

We played about with the system to find all sorts of other "features". Zooming in on the point x=–4 with an x-interval $\pm 8 \times 10^{-5}$ and a y-interval bigger by a factor of 5 produced a strange jump in the curve. It shows a decreasing positive connected piece of graph, then a quantum leap (about 5.8×10^{-7}), a horizontal zero part, then another quantum leap to a negative piece (figure 10).

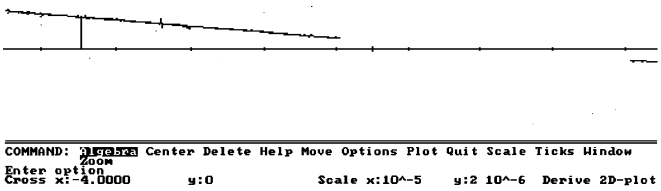

figure 10

Attempting to get a symmetrical graph, the centre point was moved a little to the right and the graph redrawn. Disaster! Where the old graph was zero (and possibly even positive), the new graph is now negative... (figure 11).

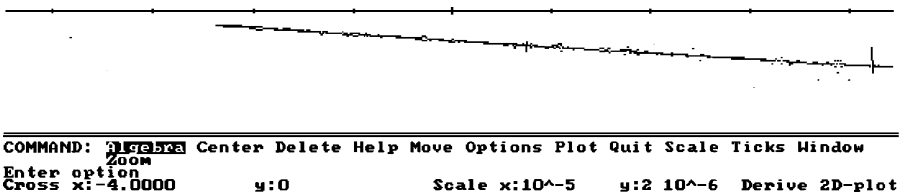

figure 11

At this stage the investigation was abandoned.

Reflections

Tall & Winkelmann (1988) hypothesise three different levels of insight into software:

- *external*: where the only facts known to the user are the input and the output, with no knowledge of the algorithms being used,
- *analogue*: where the user has some knowledge of the type of algorithms being used, and can therefore hypothesise why the program behaves in a given manner,
- *specific*: where the user understands the program both in terms of the algorithms and their implementation.

They suggest that, whilst specific insight may be an ideal which is desirable, at least analogue insight is necessary to successfully use a program to understand the unusual features it might manifest. Here clearly only external insight, at best, is achieved. Using just the published software guide, were unable to understand how the Derive software was giving such strange variations in graph-plotting.

Conclusion

After using the computer in a variety of ways we finally factorized the polynomial and understood some, but not all, of the underlying problems of computer arithmetic. Although there is undoubtedly some very powerful software around, it is clear that it cannot currently be used without some insight into the nature of the mathematics involved. The computer may help us solve problems but it still needs a human mind well-versed in mathematical concepts to understand the information the computer provides.

References

Stoutmyer & Rich 1989: Derive, Soft Wharehouse, Honolulu.

Tall 1985: Supergraph, Glentop, London.

Tall & Winkelmann 1988 : 'Hidden algorithms in the drawing of discontinuous functions', *Bulletin of the I.M.A.* 24 111–115

Postscript

The software used in the investigation is:

Supergraph by David Tall, available from Glentop Press Ltd, Unit 11 Stirling Industrial Centre, Stirling Way, Boreham Wood, Herts WD6 2BT. Upgrades to the latest versions available from Rivendell Software, 21 Laburnum Avenue, Kenilworth CV8 2DR.

Derive by David Stoutmyer and Al Rich, published by Soft Warehouse Inc. of Honolulu, Hawaii.

The illustrations printed in this article which originated on a BBC computer using *SuperGraph* were stored to disc on the BBC, transferred to a Macintosh computer using *Screen»Mac* (Human-Computer Interface Ltd), loaded into *Superpaint* (Silicon Beach Software) to be touched up and saved as postscript objects, allowing them to be rescaled in size and incorporated into a word-processor, such as Microsoft *Word* v.3.01.