Square roots by subtraction

Frazer Jarvis

When I was at school, my mathematics teacher showed me the following very strange method to work out square roots, using only subtraction, which is apparently an old Japanese method. I'll start by writing down the algorithm in a fairly formal way, which may, for example, make it easier to implement on a computer.

Although this method converges much more slowly than Newton's method for finding square roots, for example, this method also has its advantages. Perhaps the main advantage from the computational point of view is that, when finding square roots of integers, no infinite decimals are involved at any step, which can cause loss of precision due to rounding errors.

Algorithm to compute the square root of an integer n

Initial step

Let $a = 5n$ (this multiplication by 5 is the only time when an operation other than addition and subtraction is involved!), and put $b = 5$.

Repeated steps

(R1) If $a \geq b$, replace a with $a - b$, and add 10 to b.

(R2) If $a < b$, add two zeroes to the end of a, and add a zero to b just before the final digit (which will always be '5').

Conclusion

Then the digits of b approach more and more closely the digits of the square root of n.

An example

To illustrate the method, let's begin to work out the value of the square root of 2. The initial step sets $a = 5 \times 2 = 10$ and $b = 5$. So we start with the pair $(a, b) = (10, 5)$.

Now we perform the repeated steps. As $10 > 5$, we must apply **R1**. We therefore replace a with $a - b$, so that now $a = 10 - 5 = 5$, and add 10 to b, so that b becomes 15. The new pair is therefore $(5, 15)$.

We apply the rule again. This time, however, $a = 5 < 15 = b$, and so we use **R2**. This rule says: Add two zeroes to the end of a, to make $a = 500$, and put a zero before the final digit of b, so that now $b = 105$. The pair becomes $(500, 105)$.

Now $500 > 105$, so we apply **R1**, and we replace a by $a - b = 500 - 105 = 395$, and add 10 to b, so that b becomes 115. Our pair is then $(395, 115)$.

Repeatedly applying the rules in turn gives:

$$\begin{aligned}
&(10,5) \xrightarrow{\mathbf{R1}} (5,15) \xrightarrow{\mathbf{R2}} (500,105) \xrightarrow{\mathbf{R1}} (395,115) \xrightarrow{\mathbf{R1}} (280,125)\\
&\xrightarrow{\mathbf{R1}} (155,135) \xrightarrow{\mathbf{R1}} (20,145) \xrightarrow{\mathbf{R2}} (2000,1405) \xrightarrow{\mathbf{R1}} (595,1415)\\
&\xrightarrow{\mathbf{R2}} (59500,14105) \xrightarrow{\mathbf{R1}} (45395,14115) \xrightarrow{\mathbf{R1}} (31280,14125)\\
&\xrightarrow{\mathbf{R1}} (17155,14135) \xrightarrow{\mathbf{R1}} (3020,14145) \xrightarrow{\mathbf{R2}} (302000,141405)\\
&\xrightarrow{\mathbf{R1}} (160595,141415) \xrightarrow{\mathbf{R1}} (19180,141425) \xrightarrow{\mathbf{R2}} (1918000,1414205) \longrightarrow \cdots
\end{aligned}$$

and you can see that the digits of b are settling down to start with $14142\ldots$. Moreover,

$$\sqrt{2} = 1.41421356\ldots.$$

In fact, the method does not only work for positive integers. The method works for all positive numbers, even decimals such as 2.71, and even π. In these cases, rather than add two zeroes at the end, we have to multiply by 100 (which comes to the same thing for integers), i.e., to shift the decimal point two places to the right. Here's another example, where we work out the square root of 2.345. Initially, $a = 5 \times 2.345 = 11.725$, and $b = 5$. Now apply the steps as before to get:

$$\begin{aligned}
&(11.725,5) \xrightarrow{\mathbf{R1}} (6.725,15) \xrightarrow{\mathbf{R2}} (672.5,105) \xrightarrow{\mathbf{R1}} (567.5,115)\\
&\xrightarrow{\mathbf{R1}} (452.5,125) \xrightarrow{\mathbf{R1}} (327.5,135) \xrightarrow{\mathbf{R1}} (192.5,145)\\
&\xrightarrow{\mathbf{R1}} (47.5,155) \xrightarrow{\mathbf{R2}} (4750,1505) \xrightarrow{\mathbf{R1}} (3245,1515)\\
&\xrightarrow{\mathbf{R1}} (1730,1525) \xrightarrow{\mathbf{R1}} (205,1535) \xrightarrow{\mathbf{R2}} (20500,15305)\\
&\xrightarrow{\mathbf{R1}} (5195,15315) \xrightarrow{\mathbf{R2}} (519500,153105) \longrightarrow \cdots
\end{aligned}$$

and the correct value for $\sqrt{2.345}$ is $1.53133928\ldots$.

Although the method works perfectly well for any positive number, it is usually easiest to begin by multiplying or dividing the given number by 100 as often as is necessary until it lies between 1 and 100. For example, given the number 23450, we would begin by dividing by 100 twice to get 2.345, and then continuing as above. Since we have divided by 100 twice, we must multiply by 10 twice to get the correct square root: thus, the square root of 23450 is 153.133928....

Finally, let's see how the algorithm copes with a perfect square. Let's work out the square root of 16. First, multiply by 5, and set $a = 80$. As usual, $b = 5$. Then, applying our rules gives;

$$(80, 5) \xrightarrow{\mathbf{R1}} (75, 15) \xrightarrow{\mathbf{R1}} (60, 25) \xrightarrow{\mathbf{R1}} (35, 35)$$
$$\xrightarrow{\mathbf{R1}} (0, 45) \xrightarrow{\mathbf{R2}} (0, 405) \xrightarrow{\mathbf{R2}} (0, 4005) \xrightarrow{\mathbf{R2}} (0, 40005)$$

since multiplying 0 by 100 leaves it unchanged. In practice, one can stop as soon as $a = 0$, and identify the square root by removing the final digit from b.

It is an amusing exercise to program a computer to do this algorithm, at least if n is an integer, and because only integer additions and subtractions are used, there are no errors due to floating-point arithmetic. On the other hand, it is necessary to use more complicated storage techniques (strings or arrays) to store the values of a and b as they get larger and larger, and the algorithm will get slower and slower at producing successive digits.

Explanation of the algorithm

It's a bit tricky to explain that the algorithm does indeed produce the correct answer. Before we embark on the demonstration, let's make a couple of observations.

Note that the number of times rule **R1** is applied in between the applications of rule **R2** ought to give the sequence of digits in b. We can check that all these digits lie between 0 and 9. Indeed, let's choose a point at which **R2** is applied. So we begin with $a < b$, we change a to $100a$, and then subtract successively $10b - 45$ (this is what you get by putting 0 before the final 5 of b), $10b - 35$, $10b - 25$ and so on. We can do no more than 9 of these subtractions; to subtract 10 times would take away 10 numbers averaging exactly $10b$ from $100a$ – i.e., to subtract $100b$ from $100a$, but since $a < b$, the first number of the pair would become negative, which is not allowed. It follows that we can apply rule **R1** at most 9 times between any two applications of rule **R2**, and so the digits appearing in b really are exactly the number of times **R1** is applied in between applications of **R2**.

As remarked above, we'll assume that our initial number lies between 1 and 100, so that its square root lies between 1 and 10. Thus we suppose that b is equal to $b_0.b_1b_2b_3\ldots$, where all of the b_i lie between 0 and 9. Let's show that $n = b^2$. We shall consider a modified version of the algorithm which incorporates the decimal point:

Let $a = n/2$, and put $b = 0.5$.

Repeated steps

(R1′) If $a \geq b$, replace a with $a - b$, and increase the next-to-last decimal digit of b by 1;

(R2′) If $a < b$, add two zeroes to the end of a, divide b by 10, and add a zero to b just before the final decimal digit (which will always be '5').

Let's see how this works in the example before, where $n = 2$. We begin by setting $a = n/2 = 1$, and $b = 0.5$. Next, we run through the steps:

$$\begin{aligned}
&(1, 0.5) \xrightarrow{\mathbf{R1'}} (0.5, 1.5) \xrightarrow{\mathbf{R2'}} (0.5, 0.105) \xrightarrow{\mathbf{R1'}} (0.395, 0.115) \xrightarrow{\mathbf{R1'}} (0.280, 0.125)\\
&\xrightarrow{\mathbf{R1'}} (0.155, 0.135) \xrightarrow{\mathbf{R1'}} (0.020, 0.145) \xrightarrow{\mathbf{R2'}} (0.02000, 0.01405)\\
&\xrightarrow{\mathbf{R1'}} (0.00595, 0.01415) \xrightarrow{\mathbf{R2'}} (0.0059500, 0.0014105)\\
&\xrightarrow{\mathbf{R1'}} (0.0045395, 0.0014115) \xrightarrow{\mathbf{R1'}} (0.0031280, 0.0014125)\\
&\xrightarrow{\mathbf{R1'}} (0.0017155, 0.0014135) \xrightarrow{\mathbf{R1'}} (0.0003020, 0.0014145)\\
&\xrightarrow{\mathbf{R2'}} (0.000302000, 0.000141405) \xrightarrow{\mathbf{R1'}} (0.000160595, 0.000141415)\\
&\xrightarrow{\mathbf{R1'}} (0.000019180, 0.000141425) \xrightarrow{\mathbf{R2'}} (0.00001918000, 0.00001414205) \longrightarrow \cdots
\end{aligned}$$

(Note that the numbers which appear are exactly the same as in the earlier example; we have merely multiplied all of them by a power of 10.)

We begin with $a = n/2$, and, using rule **R1′**, we have to subtract various numbers (which we occasionally make smaller with rule **R2′**), we eventually get that the first number in the pair approaches 0. It follows that a is the sum of all the numbers which we take away from it, to get 0.

Before the first application of **R2′**, we make b_0 applications of rule **R1′**, subtracting $b_0/2$ on average each time. The total amount subtracted in these steps is therefore $b_0^2/2$.

Between the first and second applications of **R2′**, we make b_1 applications of rule **R1′**, subtracting $b_0/10 + b_1/200$ on average each time. The total amount subtracted in these steps is therefore $b_0b_1.10^{-1} + b_1^2.10^{-2}/2$.

Between the second and third applications of **R2′**, we make b_2 applications of rule **R1′**, subtracting $b_0/100 + b_1/1000 + b_2/20000$ on average each time.

The total amount subtracted in these steps is therefore $b_0b_2.10^{-2}+b_1b_2.10^{-3}+b_2^2.10^{-4}/2$.

In general, between the kth and $k+1$st applications of **R2′**, we make b_k applications of rule **R1′**, subtracting $b_0.10^{-k}+b_1.10^{-k-1}+\cdots+b_{k-1}.10^{1-2k}+b_k.10^{-2k}/2$ on average each time. The total amount subtracted in these steps is therefore $b_0b_k.10^{-k}+b_1b_k.10^{-k-1}+\cdots+b_{k-1}b_k.10^{1-2k}+b_k^2.10^{-2k}/2$

But remember that $n/2$ must be the sum of all the terms we are subtracting. Thus

$$\begin{aligned} n/2 \;=\; & b_0^2/2 \\ & + b_0b_1.10^{-1}+b_1^2.10^{-2}/2 \\ & + b_0b_2.10^{-2}+b_1b_2.10^{-3}+b_2^2.10^{-4}/2 \\ & + \cdots \\ & + b_0b_k.10^{-k}+b_1b_k.10^{-k-1}+\cdots+b_{k-1}b_k.10^{1-2k}+b_k^2.10^{-2k}/2 \\ & + \cdots \end{aligned}$$

the total subtracted before the first **R2′**, between the first and second, between the second and third, and so on. Note that the coefficient of b_ib_j is 10^{-i-j} if i and j are different, and is $10^{-i-j}/2$ if $i=j$. However, if we expand

$$(b_0+b_1.10^{-1}+b_2.10^{-2}+b_3.10^{-3}+\cdots)^2,$$

then the coefficient of b_ib_j is 2.10^{-i-j} if i and j are different, and is 10^{-i-j} if $i=j$. It follows that the sum making $n/2$ above is exactly

$$(b_0+b_1.10^{-1}+b_2.10^{-2}+b_3.10^{-3}+\cdots)^2/2,$$

and so

$$n=(b_0+b_1.10^{-1}+b_2.10^{-2}+b_3.10^{-3}+\cdots)^2,$$

so that

$$\sqrt{n}=b_0+b_1.10^{-1}+b_2.10^{-2}+b_3.10^{-3}+\cdots,$$

and the digits b_i coming out of the algorithm are therefore exactly the digits in the decimal expansion of $\sqrt{n}$, as required.

Frazer Jarvis *is a member of the Department of Pure Mathematics at the University of Sheffield, and his research interests are centred on algebraic number theory. Outside mathematics, he is a keen pianist and sings with the Sheffield Philharmonic Chorus.*