You Could Learn a Lot from a Quadratic: II. Digital Dentistry

Author: *Henry G. Baker, http://home.pipeline.com/~hbaker1/home.html; hbaker1@pipeline.com*

In our last episode, our hero was trying desperately to slay a quadratic by strangling it with his mouse-cord, but instead he tripped over a root. Yes, I know, this is a surd...

One *graphical* solution for a quadratic $x^2+Bx+C=0$ is attributed to Thomas Carlyle [Barbeau89]. Construct the line segment from the point $(0,1)$ and the point $(-B,C)$. Construct the circle through these two points having this line segment as its diameter. Then x_1, x_2, such that the points $(x_1,0)$, $(x_2,0)$ are the points of intersection of this circle with the x-axis, are the roots of the given quadratic equation. Here are the details:

$$\left(x+\frac{B}{2}\right)^2+\left(y-\frac{C+1}{2}\right)^2=\frac{B^2}{4}+\left(\frac{C+1}{2}-1\right)^2$$

$$x^2+Bx+y^2-(C+1)y=-C-1+1$$

$$x^2+Bx+C=(C+1)y-y^2$$

Setting $y=0$ gives us the points of intersection with the x-axis, which is the equation $x^2+Bx+C=0$.

Suppose now that we wish to solve the quadratic equation $x^2+Bx+C=0$, but don't know the quadratic formula. Or perhaps we know the formula, but don't know how to find square roots. Out of idle curiosity we start playing with *iterative* processes to see if we can find roots of the quadratic in this way.

One possibility that might occur to us is to divide the whole equation by x, to produce the equation $x+B+C/x=0$. We can then put x by itself on the left, giving the equation $x=-B-C/x$. If we now make an initial guess for x, say x_0, we can produce a (hopefully improved) x_1 by choosing $x_1=-B-C/x_0$, or more generally, given a guess x_i, we can produce the next guess $x_{i+1}=-B-C/x_i$. Obviously, if $x_0\neq 0$ already *is* a root, then $x_1=x_0$, so the sequence immediately converges. Of course, this will not work with an initial guess of $x_0=0$. In the case where one of the roots is zero, however, $C=0$, so that our iteration immediately produces the other root $x_1=-B-C/x_0=-B-0/x_0=-B$. On the other hand, if $B=0$, then $x_{i+1}=-C/x_i$, so we get the alternating sequence

$$x_0,-C/x_0,-C/(-C/x_0)=x_0,-C/x_0,\text{ etc.}$$

that never converges.

In order to get more experience with this iteration process, we try this process on the equation $(x-1)(x-2)=x^2-3x+2=0$, and start choosing initial guesses at random. Since $x_{i+1}=3-2/x_i$, we can readily calculate the following sequences with a pocket calculator.

$$3, 2.33, 2.14, 2.07, 2.03, 2.02, 2.01, ...$$

$$4, 2.5, 2.2, 2.1, 2.04, ...$$

$$0.5, -1, 5, 2.6, 2.2, 2.1, ...$$

$$1.5, 1.7, 1.8, 1.9, 1.94, 1.97, ...$$

$$-5, 3.4, 2.4, 2.17, 2.08, 2.04, ...$$

$$-0.5, 7, 2.7, 2.26, ...$$

So, in all these cases, our iteration *does* converge, and to the larger root ($x=2$). This is a bit peculiar, since if we happen to pick the smaller root ($=1$), the iteration converges on this smaller root. This leads us to investigate what happens if we pick a number very close to the smaller root — e.g., $x_0=1+\epsilon$. We get

$$x_1=3-\frac{2}{1+\epsilon}\approx 3-2(1-\epsilon)=1+2\epsilon$$

Aha! We now see that if we start even a little bit away from the smaller root, then we will move *twice* as far away on the very first iteration. In other words, for this iteration, the starting point $x_0=1$ is a *meta*stable state, whereas the starting point $x_0=2$ is apparently a *stable* state.

We are now ready to investigate the *general* case, to try to characterize under what conditions and how fast this iterative process will converge on a root.

If we take the iterative formula $x_{i+1}=-B-C/x_i$ and arrange the right-hand side as a fraction, we get $x_{i+1}=(-Bx_i-C)/x_i$. This suggests that we generalize the process slightly to produce not just a new x_{i+1}, but a new *ratio* $x_{i+1}/y_{i+1}=-B-C(y_i/x_i)$, i.e.,

$$\frac{x_{i+1}}{y_{i+1}}=\frac{-Bx_i-Cy_i}{x_i}$$

Since both the top and bottom of the right-hand side of this equation are linear functions of x_i and y_i, we are led to consider the *matrix* equation

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} -B & -C \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

An enormous amount is known about matrices [Golub96], and we can bring it all to bear on our problem. Those of you who have had linear algebra and have sharp eyes will instantly recognize our square matrix (call it 'M') as the *companion matrix* of the polynomial $x^2 + Bx + C$. The companion matrix of a polynomial is a matrix that is trivially constructed from the given polynomial, such that the 'characteristic polynomial' of the matrix is equal to that given polynomial. Thus, the characteristic polynomial of our 2×2 square matrix M is $x^2 + Bx + C$.

After reformulating our iteration process as a *matrix* iteration process, we see that we are looking for a *vector*

$$V = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

such that $MV = \lambda V$, i.e.,

$$\begin{pmatrix} -B & -C \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \approx \lambda \begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} \lambda x_n \\ \lambda y_n \end{pmatrix}$$

This *scalar* number $\lambda \neq 0$ will cancel from both the numerator λx_n and the denominator λy_n, leaving us with a 'stationary' value x_n/y_n. Such a vector V is called an *eigenvector* (rough German translation: 'own vector'), so the solution to our iteration problem is an eigenvector for the matrix M.

The iteration process described above has the effect of computing the n-th *power* of our 2×2 matrix M and applying it to the initial guess x_0/y_0. How can we characterize the M^n—the n-th power of this matrix M? We have the full power of linear algebra at our fingertips.

Suppose, for the moment, that our matrix M is 'diagonalizable'—i.e., there exists an invertible matrix T such that $T^{-1}MT = D$, and D is *diagonal*.[1] Then $M^n = (TDT^{-1})^n = T(D^n)T^{-1}$, so if $D = \text{diag}(\lambda_1, \lambda_2)$, then $D^n = \text{diag}(\lambda_1^n, \lambda_2^n)$. In other words,

$$\begin{aligned} M^n &= (T \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} T^{-1})^n \\ &= T \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}^n T^{-1} \\ &= T \begin{pmatrix} \lambda_1^n & 0 \\ 0 & \lambda_2^n \end{pmatrix} T^{-1} \end{aligned}$$

Let us now make sure that the determinant of T is 1, i.e., $|T| = 1$, which we can always arrange by dividing any other diagonalizing T' by $|T'|$, i.e., $T = T'/|T'|$. Now let the elements of T be a, b, c, d, i.e.,

$$T = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \qquad \text{and} \qquad |T| = ad - bc.$$

Since $|T| = 1$, the inverse of T is thus

$$T^{-1} = \frac{1}{|T|} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

and $M^n = TD^nT^{-1}$ can be written out as:

$$\begin{aligned} M^n &= \begin{pmatrix} -B & -C \\ 1 & 0 \end{pmatrix}^n \\ &= TD^nT^{-1} \\ &= \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \lambda_1^n & 0 \\ 0 & \lambda_2^n \end{pmatrix} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \\ &= \begin{pmatrix} a\lambda_1^n & b\lambda_2^n \\ c\lambda_1^n & d\lambda_2^n \end{pmatrix} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \\ &= \begin{pmatrix} ad\lambda_1^n - bc\lambda_2^n & -ab\lambda_1^n + ab\lambda_2^n \\ cd\lambda_1^n - cd\lambda_2^n & -bc\lambda_1^n + ad\lambda_2^n \end{pmatrix} \\ &= \begin{pmatrix} ad\lambda_1^n - bc\lambda_2^n & -ab(\lambda_1^n - \lambda_2^n) \\ cd(\lambda_1^n - \lambda_2^n) & -bc\lambda_1^n + ad\lambda_2^n \end{pmatrix} \end{aligned}$$

Now consider the absolute values $|\lambda_1|$, $|\lambda_2|$ of λ_1, λ_2. If $|\lambda_1| > |\lambda_2|$, then $|\lambda_1^n| >> |\lambda_2^n|$ for sufficiently large n, so that the terms involving λ_1^n will completely dominate those terms involving λ_2^n. Let n be sufficiently large. Then

$$\begin{aligned} M^n &= \begin{pmatrix} -B & -C \\ 1 & 0 \end{pmatrix}^n \\ &= \begin{pmatrix} ad\lambda_1^n - bc\lambda_2^n & -ab(\lambda_1^n - \lambda_2^n) \\ cd(\lambda_1^n - \lambda_2^n) & -bc\lambda_1^n + ad\lambda_2^n \end{pmatrix} \\ &\approx \begin{pmatrix} ad\lambda_1^n & -ab\lambda_1^n \\ cd\lambda_1^n & -bc\lambda_1^n \end{pmatrix} \\ &= \lambda_1^n \begin{pmatrix} ad & -ab \\ cd & -bc \end{pmatrix} \end{aligned}$$

[1] Note that we are not interested in actually *computing* this 'factored' form of M, but only in using it to better understand the meaning of the matrix power M^n.

We can now *ignore* the factor λ_1^n, because when computing x_n/y_n this factor will cancel out. Let us denote M^n/λ_1^n by M_∞, i.e., $M_\infty = M^n/\lambda_1^n$. Then

$$M_\infty = \begin{pmatrix} ad & -ab \\ cd & -bc \end{pmatrix}$$

Now applying M_∞ to an initial guess x_0/y_0, we have

$$\begin{aligned} \begin{pmatrix} x_n \\ y_n \end{pmatrix} &= M_\infty \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \\ &= \begin{pmatrix} ad & -ab \\ cd & -bc \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \\ &= \begin{pmatrix} adx_0 - aby_0 \\ cdx_0 - bcy_0 \end{pmatrix} \\ &= \begin{pmatrix} a(dx_0 - by_0) \\ c(dx_0 - by_0) \end{pmatrix} \\ &= (dx_0 - by_0) \begin{pmatrix} a \\ c \end{pmatrix} \end{aligned}$$

But the number $dx_0 - by_0$ is cancelled out in the ratio x_n/y_n, and thus $x_n/y_n = a/c$, independent of the initial guess x_0/y_0! In short, the iterative process we developed is a way to compute the ratio a/c, where a and c are the first column entries in the invertible matrix T. But what is this ratio a/c?

Since we have 5 equations in the 4 unknowns a, b, c, d (4 equations from $M = TDT^{-1}$ plus the equation $|T| = 1$), and one of these equations is redundant, we can solve for the entries a, b, c, d of T as follows:[2]

$$T = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & \frac{\lambda_1\lambda_2}{\lambda_1 - \lambda_2} \\ \frac{1}{\lambda_1} & \frac{\lambda_1}{\lambda_1 - \lambda_2} \end{pmatrix}$$

In other words, $a/c = 1/(1/\lambda_1) = \lambda_1$, so our iteration does indeed produce the root of larger absolute value λ_1.[3] Note that nowhere did we actually *compute* the factorization TDT^{-1} of M by producing T and D, but we determined that we could extract the ratio of two entries of T by computing sufficiently large powers of the matrix M.

We also note that since the ratio a/c is *independent* of the initial guess, we need not explicitly make an initial guess at all, but merely compute the matrix powers M^i.[4] We can thus compute λ_1 as $\lambda_1 = a/c = ad/cd = (-ab)/(-bc)$—i.e., the ratio of either column of M_∞.

We have thus succeeded in modelling our simple iterative arithmetic process as a matrix power. This allowed us to characterize the conditions under which the simple iterative process would converge, and to what value.

As we noticed when we performed the sequence of iterative calculations on the calculator, this iterative process doesn't converge very fast. Empirically, the number of correct digits in the result seems to be linearly related to the number of iterations. We would like to find an iterative process which converges more quickly than this.

The budding computer scientist will instantly suggest that instead of *iteratively* computing the matrix powers, we would be better off successively *squaring* the matrix M, thus producing the powers M^{2^k}. This process should get us to the answer we desire much more quickly. Indeed, with each squaring step, we might get *twice* as much precision as the previous step.

The sequence of squarings for the companion matrix of $x^2 - 3x + 2$ is:

$$\begin{pmatrix} 3 & -2 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 7 & -6 \\ 3 & -2 \end{pmatrix}, \begin{pmatrix} 31 & -30 \\ 15 & -14 \end{pmatrix}, \begin{pmatrix} 511 & -510 \\ 255 & -254 \end{pmatrix}, \begin{pmatrix} 131071 & -131070 \\ 65535 & -65534 \end{pmatrix}, \ldots$$

and the ratios a/c for these matrices are:

$$3, 2.333, 2.0667, 2.0039, 2.0000, \ldots$$

which does converge significantly faster than the iteration $x = 3 - 2/x$.

Let us now see what happens when the roots are identical—i.e., $x^2 - 2\lambda x + \lambda^2 = 0$. In this case,

$$M = \begin{pmatrix} 2\lambda & -\lambda^2 \\ 1 & 0 \end{pmatrix}$$

so

$$M^{2^i} = \begin{pmatrix} (2^i + 1)\lambda^{2^i} & -2^i\lambda^{2^i+1} \\ 2^i\lambda^{2^i-1} & (1 - 2^i)\lambda^{2^i} \end{pmatrix}$$

Thus, even when the roots are identical, the ratio of the entries in the first column of the n-th squaring will *still* converge to the value of the root. In such a case, however, the matrix squaring convergence will only be

[2] This solution also proves that T and the matrix factorization exist, so long as $\lambda_1 \neq \lambda_2$.

[3] Note that the root of *smaller* absolute value can also be trivially extracted from M_∞ as $\lambda_2 = b/d = -(-ab)/(ad) = -(-bc)/(cd)$, which is minus the ratio of the elements of either row, or as $\lambda_2 = C/\lambda_1 = Cc/a$.

[4] We have thus 'M-powered' our companion. Note that this has the effect of implicitly choosing the initial guess as $-B/1 = -B = (\lambda_1 + \lambda_2)/2$—i.e., the *mean average* of the roots—since the first column of M is $\begin{pmatrix} -B \\ 1 \end{pmatrix}$.

linear, as the ratio $(2^i+1)/2^i = 1+2^{-i}$ converges to 1 only a single bit per iteration.

Let us now see what happens when one root is the negative of the other—i.e., $\lambda_1 = -\lambda_2$, or $x^2 - \lambda^2 = 0$. In this "square root" case,

$$M^2 = \begin{pmatrix} 0 & \lambda^2 \\ 1 & 0 \end{pmatrix}^2 = \begin{pmatrix} \lambda^2 & 0 \\ 0 & \lambda^2 \end{pmatrix}$$

so that the i-th iteration is

$$M^{2^i} = \begin{pmatrix} \lambda^{2^i} & 0 \\ 0 & \lambda^{2^i} \end{pmatrix}$$

In this case, the ratio of the entries in the first column is not possible to compute, but this will be obvious from the appearance of the matrix M^{2^i}. Although it works on many quadratic equations, this matrix-squaring process does not work for simple square roots.

We have shown a *root-squaring* process for finding the roots of some quadratics, because at every step, we *square* the roots from the preceding step:

$$\begin{aligned} M^i M^i &= (TD^iT^{-1})(TD^iT^{-1}) \\ &= TD^{2i}T^{-1} \\ &= T\begin{pmatrix} \lambda_1^{2i} & 0 \\ 0 & \lambda_2^{2i} \end{pmatrix} T^{-1} \\ &= M^{2i} \end{aligned}$$

But if we are merely interested in squaring roots, we might do it more directly as

$$\begin{aligned} (x-\lambda_1)(x-\lambda_2)(x+\lambda_1)(x+\lambda_2) &= (x^2-\lambda_1^2)(x^2-\lambda_2^2) \\ &= (y-\lambda_1^2)(y-\lambda_2^2) \end{aligned}$$

where $y = x^2$. I.e., given the equation $p(x) = x^2 + Bx + C = 0$, we can compute a new equation whose roots are the *squares* of the roots of the given equation by computing

$$\begin{aligned} p(x)p(-x) &= (x^2+Bx+C)(x^2-Bx+C) \\ &= (x^2+C+Bx)(x^2+C-Bx) \\ &= (x^2+C)^2 - (Bx)^2 \\ &= (y+C)^2 - B^2y \\ &= y^2 + 2Cy + C^2 - B^2y \\ &= y^2 + (2C-B^2)y + C^2 \end{aligned}$$

This so-called *Graeffe* process produces a new quadratic equation in the variable y whose roots are the *squares* of the previous quadratic equation in the variable x. The point of root squaring is that the linear term $2C - B^2$ of the new equation is minus the sum of the *squares* of the roots, i.e., $-\lambda_1^2 - \lambda_2^2$. If the absolute values of the roots differ, and this process is repeated,, then the root with larger absolute value will eventually dominate in the sums of the squares of the roots. Furthermore, if we continue this root-squaring process, the 2^k-th power of the larger root will so dominate the 2^k-th power of the smaller root in the coefficient of the linear term of the k-th iteration, so that this coefficient *is* the 2^k-th power of the root of larger absolute value. Then by taking the 2^k-th root of the absolute value of this number, we can finally find the absolute value of the larger root.[5]

While the root-squaring method works well for roots whose absolute values are different, it cannot handle the case where the absolute values are identical, which is always the case when the roots are complex conjugates of one another.

Another way to empower the roots of the quadratic polynomial $p(x) = x^2 + Bx + C$ is by means of the 'logarithmic derivative' power series expansion

$$\begin{aligned} (\log p(x))' &= \frac{p'(x)}{p(x)} \\ &= \frac{2x+B}{x^2+Bx+C} \\ &= \frac{(x-\lambda_2)+(x-\lambda_1)}{(x-\lambda_1)(x-\lambda_2)} \\ &= \frac{1}{x-\lambda_1} + \frac{1}{x-\lambda_2} \\ &= \frac{-1/\lambda_1}{1-x/\lambda_1} + \frac{-1/\lambda_2}{1-x/\lambda_2} \\ &= -\left(s_1 + s_2x + s_3x^2 + s_4x^3 + ...\right) \end{aligned}$$

and here $s_i = \lambda_1^{-i} + \lambda_2^{-i}$. The point of this expansion is to show that the coefficients of the power series for the ratio of polynomials $p'(x)/p(x)$ are the sums of the powers of the root inverses. In other words, the coefficient of x^i in this power series expansion is

$$-\left(\frac{1}{\lambda_1^{i+1}} + \frac{1}{\lambda_2^{i+1}}\right)$$

If $|\lambda_1| > |\lambda_2|$, then $1/|\lambda_1| < 1/|\lambda_2|$, so that as i increases, $1/\lambda_2^i$ will dominate $1/\lambda_1^i$ in the sum $1/\lambda_1^i + 1/\lambda_2^i$. Thus, if we pick two successive coefficients large enough,

[5] We can also use this Graeffe method to produce a tight upper bound on the size (absolute value) if the largest root [Zippel93,11.2].

their ratio will tend towards λ_2, i.e.,

$$\begin{aligned}\frac{s_i}{s_{i+1}} &\approx \frac{\lambda_2^{-i}}{\lambda_2^{-i-1}} \\ &= \frac{\lambda_2^{i+1}}{\lambda_2^i} \\ &= \lambda_2\end{aligned}$$

Ratios of the form $p'(x)/p(x)$ are particularly interesting when finding roots, because any repeated roots will cancel out. Thus, if $p(x) = (x-\lambda)^2$, then $p'(x) = 2(x-\lambda)$, so

$$\begin{aligned}\frac{p'(x)}{p(x)} &= \frac{2(x-\lambda)}{(x-\lambda)^2} \\ &= \frac{2}{x-\lambda} \\ &= \frac{-2/\lambda}{1-x/\lambda} \\ &= -2\left(1/\lambda + x/\lambda^2 + x^2/\lambda^3 + x^3/\lambda^4 + ...\right)\end{aligned}$$

and the ratio of successive coefficients converges to (actually it already *is*) λ.

Isaac Newton developed a clever and quite general technique for finding roots. Suppose that $f(x)$ is a function with a power series $f(x) = f(0) + f'(0)x + f''(0)x^2/2!+...$. Then if x_0 is an initial guess for a root of $f(x)$, we can expand $f(x)$ in another power series around the point $x = x_0$:

$$f(x-x_0) = f(x_0)+(x-x_0)f'(x_0)+(x-x_0)^2f''(x_0)/2!+...$$

If we now ignore the terms beyond the linear terms, then $f(x-x_0) \approx f(x_0) + (x-x_0)f'(x_0)$. Since we are looking for a *root*, where $f(x) = 0$, we assume that our approximation is reasonable, and solve it for x:

$$\begin{aligned}f(x_0) + (x-x_0)f'(x_0) &= 0 \\ \frac{f(x_0)}{f'(x_0)} + x - x_0 &= 0 \\ x &= x_0 - \frac{f(x_0)}{f'(x_0)}\end{aligned}$$

In general, of course, we are looking for an improvement x_{i+1} of x_i, so we make a sequence of linear approximations:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Let us now consider Newton's method for the quadratic equation $p(x) = x^2 - N = 0$, i.e., for finding the *square root* $\sqrt{N}$ of N. In this case,

$$\begin{aligned}x_{i+1} &= x_i - \frac{p(x_i)}{p'(x_i)} \\ &= x_i - \frac{x_i^2 - N}{2x_i} \\ &= \frac{2x_i^2 - x_i^2 + N}{2x_i} \\ &= \frac{x_i^2 + N}{2x_i} \\ &= \frac{x_i + N/x_i}{2}\end{aligned}$$

In short, a better approximation to the square root of N can be had by *averaging* the current approximation with the quotient of N by the current approxmation.[6]

Newton's method usually converges very fast. Suppose, for example, that we have a current approximation $\sqrt{N}(1+\epsilon)$ to the square root of N. Then

$$\begin{aligned}x_{i+1} &= \frac{x_i + N/x_i}{2} \\ &= \frac{\sqrt{N}(1+\epsilon) + N/\sqrt{N}(1+\epsilon)}{2} \\ &= \frac{N(1+\epsilon)^2 + N}{2\sqrt{N}(1+\epsilon)} \\ &= \sqrt{N}\,\frac{(1+\epsilon)^2 + 1}{2(1+\epsilon)} \\ &= \sqrt{N}\,\frac{1 + 2\epsilon + \epsilon^2 + 1}{2(1+\epsilon)} \\ &= \sqrt{N}\left(1 + \frac{\epsilon^2}{2(1+\epsilon)}\right) \\ &\approx \sqrt{N}\left(1 + \epsilon^2/2\right)\end{aligned}$$

Thus, if ϵ is small, then ϵ^2 is considerably smaller, and we have a *quadratically* convergent algorithm for the square root—i.e., the number of accurate bits in the result *doubles* with each iteration.

Yet another way to appreciate Newton's iterative square root is to consider how it operates when finding the square root of $N = 1$, considering each guess as a

[6] [Garver32] attributes this square root method to *Heron of Alexandria* circa 200 A.D.

ratio:

$$\begin{aligned}\frac{x_{i+1}}{y_{i+1}} &= \frac{x_i/y_i + y_i/x_i}{2} \\ &= \frac{x_i^2 + y_i^2}{2x_iy_i} \\ &= \frac{\cosh^2\alpha_i + \sinh^2\alpha_i}{2\cosh\alpha_i\sinh\alpha_i} \\ &= \frac{\cosh 2\alpha_i}{\sinh 2\alpha_i} \\ &= \coth 2\alpha_i\end{aligned}$$

In other words, if

$$x_i/y_i = \frac{\cosh\alpha_i}{\sinh\alpha_i} = \coth\alpha_i,$$

then

$$x_{i+1}/y_{i+1} = \coth 2\alpha_i,$$

or, for general N,

$$x_i/y_i = \sqrt{N}\coth 2^i\alpha_0,$$

where

$$\alpha_0 = \text{atanh}\left(\sqrt{N}/(x_0/y_0)\right).$$

(α_0 is real only when $|x_0/y_0| > \sqrt{N}$.) Since $|\coth 2^i\alpha| = 1/|\tanh 2^i\alpha|$ approaches 1 *very* quickly with increasing i (assuming that $\alpha \neq 0$), we have another proof that Newton's iteration converges to the square root. This derivation also shows that if $x_0 > \sqrt{N}$, then $x_i > \sqrt{N}$ for all i, i.e., x_i converges *monotonically* towards $\sqrt{N}$.

Not only is Newton's method particularly pretty ([Peitgen86]), but it is also enormously efficient. [Paterson72] shows that Newton's (Heron's) method has optimal efficiency for quadratic equations, where by "efficiency" he means the number of bits of precision gained per iteration relative to the number of operations performed per iteration. Herein we have show why Heron is our quadratic hero.

References

[Barbeau89] Barbeau, E.J. *Polynomials*. Springer-Verlag, New York, 1989.

[Frame45] Frame, J.S. "Machines for Solving Algebraic Equations." *Math. Tables and other Aids to Comput.* **I**, 9 (Jan. 1945), 337-353.

[Garver32] Garver, R. "A Square Root Method and Continued Fractions." *Amer. Math. Monthly* **39**, 9 (Nov. 1932), 533-535.

[Golub96] Golub, G.H., and Van Loan, C.F. *Matrix Computations, 3rd Ed*. Johns Hopkins Univ. Press, Baltimore, 1996.

[Knuth81] Knuth, D.E. *Seminumerical Algorithms, 2nd Ed.* Addison-Wesley, Reading, MA, 1981.

[Melzak73] Melzak, Z.A. *Companion to Concrete Mathematics: Mathematical Techniques and Various Applications*. John Wiley & Sons, New York, 1973.

[Paterson72] Paterson, M.S. "Efficient Iterations for Algebraic Numbers." In Miller, R.E., and Thatcher, J.W., (eds.) *Complexity of Computer Computations*. Plenum Press, New York, 1972.

[Peitgen86] Peitgen, H.-O., and Richter, P.H. *The Beauty of Fractals: Images of Complex Dynamic Systems*. Springer-Verlag, Berlin, 1986.

[Press86] Press, W.H., *et al*. *Numerical Recipes*. Cambridge Univ. Press, 1986, ISBN 0-521-30811-9.

[Turnbull46] Turnbull, H.W. *Theory of Equations*. Oliver and Boyd, Edinburgh and London; New York: Interscience Publishers, Inc., 1946.

[Young72] Young, D.M., and Gregory, R.T. *A Survey of Numerical Mathematics, Vol. I*. Dover Publ., New York, 1972.

[Zippel93] Zippel, R. *Effective Polynomial Computation*. Kluwer Academic Publishers, Boston, 1993.

Henry Baker has been diddling bits for 35 years, with time off for good behavior at MIT and Symbolics. In his spare time, he collects garbage and tilts at windbags. *This column appeared in ACM Sigplan Notices 33,2 (Feb 1998), 34-39.*