How to Solve a Cubic Equation
Part 2 – The $1\overline{1}$ Case

James F. Blinn
Microsoft Research
blinn@microsoft.com

Originally published in
IEEE Computer Graphics and Applications
Jul/Aug 2006, pages 90–100

Historical Background

In my last article we got friendly with cubic polynomials and their discriminants and Hessians. This time we will begin to use this knowledge to find the roots of cubic equations. While the solution to quadratic equations was known to the ancients, the first major breakthrough in the solution of cubic equations didn't happen until about 1515 by Scipione dal Ferro, and independently by Niccolò Tartaglia in 1535. This technique was refined and made public in a book published in 1545 by Gerolamo Cardano (The solution of cubics is often credited only to Cardano. Let this be a lesson to you: whoever publishes gets their name on things). The whole story is recounted in detail in chapter 3 of a wonderful book by Mario Livio [1]. (By the way, the title of this book refers to quintic equations.) Since the 1500's the cubic solution algorithm has been refined and re-derived in many ways. In fact, noted numerical analyst Forman Acton states in his book [2]

> "Every generation in this century, it seems, has developed a cubic solver – or at least espoused one that was proclaimed to be superior to its predecessors. These superiorities, when real, came from a change in the arithmetic tools available: mechanical desk calculators supplanted mental arithmetic and tables of square roots (1930); mechanical square roots were added (1949); and now electronic computers make *iterative* calculation not only efficient but psychologically acceptable. (A person using pencil and paper and multiplying numbers in his head prefers a finite algorithm --- the finiter the better! Iteration, even if actually shorter, had two strikes against it in the nineteenth century.)"

Acton then proceeds to give a fairly straightforward iterative algorithm of his own based on Newton iteration. The main trick is coming up with good initial estimates of the roots.

Nevertheless, I am going to pursue closed form algebraic solutions here, both because of intellectual curiosity and because of an even more recent development in arithmetic tools: the GPU pixel shader. In a recent project here at MSR, Charles Loop and I have been working on rendering cubic surfaces on GPUs [3] which requires solving a cubic equation at each pixel. Since the hardware shaders for a group of pixels all operate in parallel this works best if the algorithm uses a constant number of instructions for each pixel. An iterative solution that, because of the local coefficient values, takes many iterations at one pixel and only a few at the neighboring pixel wouldn't fit too well with the hardware.

So our goal is to find a good practical closed-form solution process for cubic equations. Then, at the end of this discussion, I will go back and make some observations about the relation between iterative and closed-form solutions.

Review and Notation

Last time I introduced a form of the cubic polynomial that is slightly different from what you typically see. The first difference is that the polynomial is homogeneous; homogeneous polynomials crop up in any rendering situation that involves 3D perspective. The second difference is that I defined the coefficients of the middle two terms as having a factor of three separated out from them. The cubic polynomial is, then:

$$f(x,w) = Ax^3 + 3Bx^2w + 3Cxw^2 + Dw^3 \tag{0.1}$$

I find that this notation makes the derivations much neater and will also help us in future generalizations. We will want our solution algorithm for $f(x,w)=0$ to work over a wide range of A, B, C and D values, and in particular when various combinations of them are zero.

Transformations

One of the important tools we will use in solving $f(x,w)=0$ is transformation in parameter space. We define the parameter-space transformation as the 2x2 homogeneous matrix product

$$\begin{bmatrix} x & w \end{bmatrix} = \begin{bmatrix} \tilde{x} & \tilde{w} \end{bmatrix} \underbrace{\begin{bmatrix} t & u \\ s & v \end{bmatrix}}_{\mathbf{T}}$$

(Note that I'm actually defining the transformation that goes backwards from tilde space to non-tilde space. Again, this is just for convenience in later calculations.) Last time we found that the coefficients of the transformed polynomial can be calculated by

$$\begin{bmatrix} \tilde{A} \\ \tilde{B} \\ \tilde{C} \\ \tilde{D} \end{bmatrix} = \begin{bmatrix} t^3 & 3t^2u & 3tu^2 & u^3 \\ t^2s & 2tus + t^2v & u^2s + 2tuv & u^2v \\ ts^2 & us^2 + 2tsv & 2usv + tv^2 & uv^2 \\ s^3 & 3s^2v & 3sv^2 & v^3 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \tag{0.2}$$

We will use this formula to find the effects of a given set of s, t , u, and v values on the polynomial.

Invariants and Covariants

There are two interesting functions of the homogeneous cubic polynomial that we will find useful.

The Hessian

This is a quadratic polynomial whose coefficients are

$$\begin{aligned} \delta_1 &= AC - B^2 \\ \delta_2 &= AD - BC \\ \delta_3 &= BD - C^2 \end{aligned} \tag{0.3}$$

The Hessian polynomial can be written in matrix form:

$$2\left(\delta_1 x^2 + \delta_2 xw + \delta_3 w^2\right) = \begin{bmatrix} x & w \end{bmatrix} \underbrace{\begin{bmatrix} 2\delta_1 & \delta_2 \\ \delta_2 & 2\delta_3 \end{bmatrix}}_{\mathbf{H}} \begin{bmatrix} x \\ w \end{bmatrix}$$

The interesting thing about the Hessian is that, as the cubic is transformed by **T** the Hessian is also transformed by the same **T**. Such a quantity is called a *covariant*. Furthermore, if the Hessian is singular (has a double root) then so is the cubic. And when this happens the double roots for the Hessian and the cubic are the same.

The Discriminant

The condition that the Hessian (and thus the cubic) is singular is that the matrix **H** is singular. The discriminant of the cubic is thus the determinant of **H,** calculated as

$$\begin{aligned} \Delta &= \det \mathbf{H} = 4\delta_1\delta_3 - \delta_2{}^2 \\ &= -A^2D^2 + 6ABCD - 4AC^3 - 4B^3D + 3B^2C^2 \end{aligned} \tag{0.4}$$

The discriminant is a scalar, but as the cubic is transformed by **T** the discriminant transforms according to

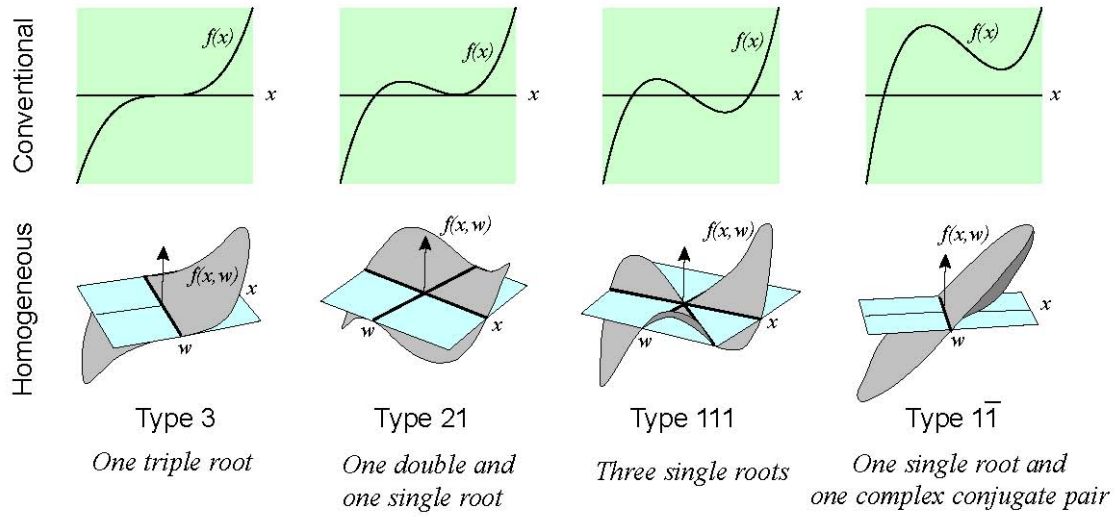
$$\tilde{\Delta} = (\det \mathbf{T})^6 \Delta \tag{0.5}$$

Such a scalar quantity is called an *invariant* of the cubic.

Types of roots

There are four distinct patterns of the roots that can occur with cubic polynomials, all plotted in Figure 1.

Figure 1. The four types of cubic polynomial



In an earlier article [4] I showed the solution to the first two of the following four cases.

One triple root – Type 3

This is indicated by the condition $\delta_1 = \delta_2 = \delta_3 = 0$. The root itself has three homogeneously equivalent ways to calculate it

$$[x \quad w] = [-B \quad A] \text{ or } [-C \quad B] \text{ or } [-D \quad C]$$

In fact, the condition that these three are homogeneously equivalent is just the same as the condition that $\delta_1 = \delta_2 = \delta_3 = 0$. This is because two vectors $[x_1 \quad w_1]$ and $[x_2 \quad w_2]$ are homogeneously equivalent (are a nonzero scalar multiple of each other) if $x_1 w_2 - x_2 w_1 = 0$.

One Double Root and one Single root – Type 21

This is indicated by the condition $\Delta = 4\delta_1\delta_3 - \delta_2^2 = 0$. The double root has two homogeneously equivalent formulations

$$[x \quad w] = [-\delta_2 \quad 2\delta_1] \text{ or } [2\delta_3 \quad -\delta_2] \tag{0.6}$$

Again, the condition that these two formulations are homogeneously equivalent is just that $4\delta_1\delta_3 - \delta_2^2 = 0$.

Three distinct real roots – Type 111

This is indicated by $\Delta > 0$. We will primarily discuss this in Part 4.

One real and one complex conjugate pair roots – Type $1\,\overline{1}$

This is indicated by $\Delta < 0$. This is what we will deal with this time.

The Conventional Solution

If you plug "solve cubic equation" into any Internet search engine you will be rewarded by a whole host of sites giving various versions of the solution. In this section I will provide a distillation of many of these expositions in terms of the notation we are using here. Some parts of the derivation that might seem slightly odd are motivated by generalizations that I'm going to apply later, so bear with me.

Step 1 – Depress the polynomial

The first step in all the solution techniques is to translate the polynomial in parameter space to create a new polynomial (with tilde-ed coefficients) that has $\tilde{B} = 0$. This is known in the business as "depressing" the polynomial (insert joke about depressed polynomials here). The sum of the roots of such a polynomial is zero. Taking a cue from our solution process for quadratics in [5] we can use the following matrix to depress the polynomial

$$\mathbf{T} = \begin{bmatrix} t & u \\ s & v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -B & A \end{bmatrix}$$

Plugging this into equation (0.2) gives us

$$\begin{aligned} \tilde{A} &= A \\ \tilde{B} &= 0 \\ \tilde{C} &= A\left(AC - B^2\right) \\ \tilde{D} &= A\left(2B^3 - 3ABC + A^2 D\right) \end{aligned}$$

Now we note that each coefficient contains a factor of *A*. As long as $A \neq 0$ we can toss it out as a homogeneous scale factor. This is tantamount to a uniform scale of the $[\tilde{x}, \tilde{w}]$ space by $A^{-1/3}$. The solutions remain the same but we can continue with the following coefficients

$$\begin{aligned} \overline{A} &= 1 \\ \overline{B} &= 0 \\ \overline{C} &= AC - B^2 \\ \overline{D} &= 2B^3 - 3ABC + A^2 D \end{aligned}$$

It will be useful to express these last two quantities in terms of the Hessian components. Referring to their definition (0.3) we have

$$\begin{aligned} \overline{C} &= \delta_1 \\ \overline{D} &= -2B\delta_1 + A\delta_2 \end{aligned} \tag{0.7}$$

So our polynomial is

$$\tilde{x}^3 + 3\overline{C}\tilde{x}\tilde{w}^2 + \overline{D}\tilde{w}^3 = 0$$

Now without loss of generality we can fix $\tilde{w} = 1$ since this cubic cannot have a solution where $\tilde{w} = 0$. This leaves us with the simplified cubic to solve

$$\tilde{x}^3 + 3\overline{C}\tilde{x} + \overline{D} = 0 \tag{0.8}$$

Now what?

Step 2 – A Miracle Occurs

Here is where dal Ferro and Tartaglia earned their lunch. In the words of the cartoonist Sidney Harris (http://www.sciencecartoonsplus.com/gallery.htm) "a miracle occurs". Behold the following identity (true for any values *p*, *q*)

$$(p+q)^3 - 3pq(p+q) - (p^3+q^3) = 0$$

If you compare this with equation (0.8) you can see the following match ups

$$\underbrace{(p+q)}_{\tilde{x}}^3 \underbrace{-3pq}_{3\bar{C}}\underbrace{(p+q)}_{\tilde{x}} \underbrace{-(p^3+q^3)}_{\bar{D}} = 0$$

so if we can find *p* and *q* that satisfy

$$\begin{aligned} -pq &= \bar{C} \\ -p^3 - q^3 &= \bar{D} \end{aligned} \tag{0.9}$$

then our answer will be

$$\tilde{x} = p + q$$

So solve the first equation of (0.9) for *q* and plug it into the second.

$$-p^3 - \left(-\frac{\bar{C}}{p}\right)^3 = \bar{D}$$

Clearing fractions and rearranging gives us a sixth degree polynomial in *p*.

$$p^6 + \bar{D}p^3 - \bar{C}^3 = 0 \tag{0.10}$$

This sounds way worse, but look! It's actually a quadratic polynomial in p^3, and we know how to solve quadratics. The quadratic equation gives us

$$p^3 = \frac{-\bar{D} \pm \sqrt{\bar{D}^2 + 4\bar{C}^3}}{2}$$

Given p^3 we get q^3 from the second line of equation (0.9). This works out to be

$$\begin{aligned} q^3 &= -p^3 - \bar{D} \\ &= \frac{-\bar{D} \mp \sqrt{\bar{D}^2 + 4\bar{C}^3}}{2} \end{aligned}$$

In other words q^3 is the other root of the quadratic. It doesn't really matter which is which since we will ultimately be only looking at the sum and difference of p and q, so I will pick the topmost of the $\pm$ and $\mp$ signs and take cube roots to get

$$\begin{aligned} p &= \sqrt[3]{\frac{-\bar{D} + \sqrt{\bar{D}^2 + 4\bar{C}^3}}{2}} \\ q &= \sqrt[3]{\frac{-\bar{D} - \sqrt{\bar{D}^2 + 4\bar{C}^3}}{2}} \end{aligned} \tag{0.11}$$

Step 3 – Use the discriminant

The quantity under the square root sign of equation (0.11) is actually rather interesting. If you look at the definition for the discriminant in equation (0.4) and apply it to the over-bar polynomial, which has $\bar{A} = 1,\ \bar{B} = 0$ you see that what remains of the big expression is

$$\bar{\Delta} = -\bar{D}^2 - 4\bar{C}^3 \tag{0.12}$$

Then equation (0.5) gives the relation between $\bar{\Delta}$ and Δ, the discriminant of the original untransformed polynomial. We first need the net transformation from the original cubic to the over-bar cubic:

$$\begin{bmatrix} 1 & 0 \\ -B & A \end{bmatrix} \begin{bmatrix} A^{-1/3} & 0 \\ 0 & A^{-1/3} \end{bmatrix}$$

The determinant of this matrix product is $A^{1/3}$. So we have

$$\bar{\Delta} = \left(A^{1/3}\right)^6 \Delta = A^2 \Delta$$

Combining this with equation (0.12) gives us

$$\bar{D}^2 + 4\bar{C}^3 = -A^2 \Delta$$

Another, more laborious, way to see this is to plug in the definitions of $\bar{C}$ and $\bar{D}$ from equation (0.7) and expand out.

I'm now going to do something that looks like a no-op but that will eliminate some clutter in later derivations. Since we have $A = \tilde{A}$ I will use the tilde-ed version from now on, giving us our final relation.

$$\bar{D}^2 + 4\bar{C}^3 = -\tilde{A}^2 \Delta \tag{0.13}$$

This is actually a special case of an important relation we will refer to a lot in later discussions.

Combining equation (0.11) and (0.13) gives us

$$p = \sqrt[3]{\frac{-\bar{D} + \tilde{A}\sqrt{-\Delta}}{2}}, \quad q = \sqrt[3]{\frac{-\bar{D} - \tilde{A}\sqrt{-\Delta}}{2}} \tag{0.14}$$

Step 4 – Account for complex roots

Now let's look at equation (0.14) in the light of the sign of Δ. If $\Delta < 0$ then the square roots are real and *p* and *q* are real. This is the situation for a type $1\bar{1}$ with one real and two complex conjugate roots.

On the other hand if $\Delta > 0$ we have already stated that there are three distinct real roots. But according to equation (0.14) a positive Δ will yield a complex value under the cube root signs. That's fine, but it means that we need to take a complex-valued cube root to find *p* and *q;* tricky but do-able. Ultimately it will turn out that the imaginary values cancel out and we have three real roots. We started with four real coefficients (*A,B,C,D*) and we wound up with three real roots. But along the way we needed to do calculations with complex quantities as intermediate values. This situation is what originally prompted mathematicians to take complex arithmetic seriously. Fortunately there is a way to do this more simply; we'll get into that in Part 4.

For the remainder of this article we will deal with the type $1\bar{1}$ case. The fact that *p* and *q* are real means that their sum is real which gives us the real root, but how do we get the complex conjugate roots? We remember that in the complex world there are actually three cube roots of any number. For example the three cube roots of 1 are 1, and $-\frac{1}{2} + \frac{\sqrt{3}}{2} i$ (which is usually named ω) and $-\frac{1}{2} - \frac{\sqrt{3}}{2} i$ (which is equal to ω^2). This means that there are three variants each for *p* and *q* which satisfy the second row of equation (0.9)

$$p^3 = (\omega p)^3 = \left(\omega^2 p\right)^3$$
$$q^3 = (\omega q)^3 = \left(\omega^2 q\right)^3$$

But to satisfy the first row of equation (0.9) we must match these up in the following three pairings:

$$-pq = \bar{C}$$
$$-(\omega p)(\omega^2 q) = \bar{C}$$
$$-(\omega^2 p)(\omega q) = \bar{C}$$

This gives us the three roots to the depressed cubic as

$$\tilde{x}_1 = p + q$$
$$\tilde{x}_2 = \omega\ p + \omega^2 q$$
$$\tilde{x}_3 = \omega^2\ p + \omega\ q$$

Plugging in the definition of ω and shuffling a bit gives us

$$\begin{aligned} \tilde{x}_1 &= p + q \\ \tilde{x}_2 &= -\tfrac{1}{2}(p+q) + \tfrac{\sqrt{3}}{2}(p-q)\,i \\ \tilde{x}_3 &= -\tfrac{1}{2}(p+q) - \tfrac{\sqrt{3}}{2}(p-q)\,i \end{aligned} \qquad (0.15)$$

Notice that, as expected, the sum $\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_3 = 0$.

Finally we must un-depress the cubic by transforming back to the original parameter space using

$$\begin{bmatrix} x & w \end{bmatrix}_i = \begin{bmatrix} \tilde{x}_i & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -B & A \end{bmatrix}$$

We can call this the Prozac transformation (sorry, I couldn't help myself). Putting all these calculations together gives Algorithm A0

Algorithm A0

$$\delta_1 = AC - B^2$$
$$\delta_2 = AD - BC$$
$$\delta_3 = BD - C^2$$
$$\Delta = 4\delta_1\delta_3 - \delta_2{}^2$$

$$\bar{D} = -2B\delta_1 + A\delta_2$$

$$p = \sqrt[3]{\frac{-\bar{D} + \tilde{A}\sqrt{-\Delta}}{2}}$$
$$q = \sqrt[3]{\frac{-\bar{D} - \tilde{A}\sqrt{-\Delta}}{2}}$$

$$\tilde{x}_1 = p + q$$
$$\tilde{x}_2 = -\tfrac{1}{2}(p+q) + \tfrac{\sqrt{3}}{2}(p-q)\,i$$
$$\tilde{x}_3 = -\tfrac{1}{2}(p+q) - \tfrac{\sqrt{3}}{2}(p-q)\,i$$

$$\begin{bmatrix} x & w \end{bmatrix}_i = \begin{bmatrix} \tilde{x}_i - B & A \end{bmatrix}$$

Checking the Double Root Case

In the case where $\Delta = 0$ we expect that this algorithm devolves nicely to give us a double root. Sure enough, for $\Delta = 0$ we get p and q being equal:

$$p = q = \sqrt[3]{\frac{-\bar{D}}{2}}$$

Equation (0.15) then gives

$$\tilde{x}_1 = 2p$$
$$\tilde{x}_2 = -p$$
$$\tilde{x}_3 = -p$$

$$\begin{bmatrix} x & w \end{bmatrix}_1 = \begin{bmatrix} 2p - B & A \end{bmatrix}$$
$$\begin{bmatrix} x & w \end{bmatrix}_{2,3} = \begin{bmatrix} -p - B & A \end{bmatrix}$$

But that's up with the cube root? Equation (0.6) gives a formulation for the double root which is way simpler. How do we reconcile these? Equation (0.13) saves us. For $\Delta = 0$ it gives us

$$\bar{D}^2 = -4\bar{C}^3$$

If $\bar{D} = 0$ then p=0 and we have a triple root. But as long as $\bar{D} \neq 0$ (and therefore $\bar{C} = \delta_1 \neq 0$) we can massage the innards of the cube root as follows

$$p = q = \sqrt[3]{\frac{-\bar{D}}{2}} = \sqrt[3]{\frac{-\bar{D}^3}{2\bar{D}^2}} = \sqrt[3]{\frac{-\bar{D}^3}{-8\bar{C}^3}} = \frac{\bar{D}}{2\bar{C}}$$
$$= \frac{-2B\delta_1 + A\delta_2}{2\delta_1}$$

Voila. No more cube root. The double root is now

$$\begin{bmatrix} x & w \end{bmatrix}_{2,3} = \begin{bmatrix} -\dfrac{-2B\delta_1 + A\delta_2}{2\delta_1} - B & A \end{bmatrix}$$
$$= \frac{A}{2\delta_1}\begin{bmatrix} -\delta_2 & 2\delta_1 \end{bmatrix}$$

This is (a homogeneous scale times) the result in equation (0.6). Whew! This also gets us a nice formulation for the single root:

$$\begin{bmatrix} x & w \end{bmatrix}_1 = \begin{bmatrix} 2p - B & A \end{bmatrix}$$
$$= \begin{bmatrix} \dfrac{-2B\delta_1 + A\delta_2}{\delta_1} - B & A \end{bmatrix}$$
$$= \frac{1}{\delta_1}\begin{bmatrix} -3B\delta_1 + A\delta_2 & A\delta_1 \end{bmatrix}$$

This means that for the two degenerate cases, Type 21 and Type 3, we can find the (multiple) solutions without any nasty cube or square roots. This result, however, is mainly of theoretical interest. In the real numerical world it's probably not a good idea to test for Δ exactly equal to zero and branch off to alternate code. Rather we would just take the cube roots in equation (0.14) and if, as a result, we happen to get p=q we will then numerically get $\tilde{x}_2 = \tilde{x}_3$.

What could possibly go wrong?

Two words: round off error.

The main source of round off error in any numerical calculation is the subtraction of two nearly-equal numbers with finite precision arithmetic (or equivalently the addition of two numbers that have opposite signs but are nearly equal in magnitude). The high order bits from each operand will cancel out leaving a result with fewer bits of precision. To properly analyze where this could happen in an algorithm you must

examine each addition or subtraction to see under what conditions the two sides of a subtraction are nearly equal (or one side of an addition nearly equals minus the other). The problem is that there are a lot of them. For algorithm A0, there are eleven unique addition/subtractions. Each one is a potential time bomb for our accuracy.

But here's some good news. Suppose you have a bad addition/subtraction, giving you a smallish value that is mostly numerical noise. All is not lost. If, later in the algorithm, you happen to add some large value to it the good digits in this large value can swamp out your numerical noise.

So our next task is to figure out as best we can where the trouble spots might be and see what we can do about them. I don't have a complete analysis in this part, but I do have some very useful results that can improve our accuracy tremendously.

I have not seen many articles addressing numerical errors in closed form solutions of cubic equations. The most popular are [6], [7], and [8]. These all address the same source of error which we will expand upon next.

Properly Solving the Embedded Quadratic

Equation (0.10) is a quadratic equation, and we already know from [5] the source of numerical problems with quadratics and how to fix them. In the notation of the present discussion we look at the two calculations $-\bar{D}+\tilde{A}\sqrt{-\Delta}$ and $-\bar{D}-\tilde{A}\sqrt{-\Delta}$. Depending on the signs of $\bar{D}$ and $\tilde{A}$ one or the other of these will be a potentially bad addition, but the other one will be guaranteed safe (adding two postitives or two negatives). So we can safely calculate only one of *p* or *q* using equation (0.14). But that's ok, since the second line of equation (0.9) shows us, given one of *p* or *q*, how to get the other using just a numerically safe division. As a first pass of turning this into an algorithm we have

if $\bar{D}$ and $\tilde{A}$ have different signs

$$p=\sqrt[3]{\frac{-\bar{D}+\tilde{A}\sqrt{-\Delta}}{2}},\ q=-\frac{\bar{C}}{p}$$

else // $\bar{D}$ and $\tilde{A}$ have the same sign

$$q=\sqrt[3]{\frac{-\bar{D}-\tilde{A}\sqrt{-\Delta}}{2}},\ p=-\frac{\bar{C}}{q}$$

We can simplify this a bit by noting that the values of *p* and *q* can be exchanged without affecting the algorithm. We then can arrange the calculation as shown in Algorithm A.

Algorithm A

Everything is the same as Algorithm A0 except the calculation of p and q is replaced by

$$\text{if } \bar{D} < 0 \text{ then } p = \sqrt[3]{\frac{-\bar{D} + |\tilde{A}|\sqrt{-\Delta}}{2}}$$

$$\text{else } p = \sqrt[3]{\frac{-\bar{D} - |\tilde{A}|\sqrt{-\Delta}}{2}}$$

$$q = -\frac{\bar{C}}{p}$$

Algorithm A is better, but not by as much as you might hope. The compensating force is the fact that we are adding the (cube roots of) the two roots of the quadratic together. Going back to the original calculation, equation (0.14), whenever the error on, say, q is noticeable it's because it is close to zero. Its value is then necessarily a lot smaller than p so that the addition $\tilde{x} = p + q$ largely swamps the error. Nevertheless, the improved calculation is definitely worthwhile, and it also has the pleasant effect that now we have to take the cube root of only one quantity.

Looking inside the box

Now let's take a closer look at algorithm A to see where else we might have problems. We'll do the following algebraic experiment. We construct a cubic having three known roots: a real root at a, and two complex conjugate roots at $b+ic$ and $b-ic$. This looks like

$$\begin{aligned} f(x,w) &= (x - aw)\big(x - (b+ic)w\big)\big(x - (b-ic)w\big) \\ &= x^3 + (-a-2b)x^2w + \left(+2ab + b^2 + c^2\right)xw^2 - a\left(b^2 + c^2\right)w^3 \end{aligned} \tag{0.16}$$

Extracting out the coefficients we have

$$\begin{aligned} A &= 1 \\ B &= \tfrac{1}{3}(-a-2b) \\ C &= \tfrac{1}{3}\left(+2ab + b^2 + c^2\right) \\ D &= -a\left(b^2 + c^2\right) \end{aligned} \tag{0.17}$$

Let's run these quantities through algorithm A. This will give us some insight into how the algorithm works and enable us to identify some danger zones. Now remember that when we normally run algorithm A starting with just the coefficients A, B, C, D we don't know what any of the stuff on the right side is yet. All the right hand sides here are "hidden" inside the numerical intermediate values we would be calculating.

We start by plugging the coefficients from equation (0.17) into algorithm A giving, after simplification.

$$\begin{aligned} \delta_1 &= \tfrac{1}{9}\left(-(a-b)^2 + 3c^2\right) \\ \delta_2 &= \tfrac{1}{9}\left(+2b(a-b)^2 + 2(-4a+b)c^2\right) \\ \delta_3 &= \tfrac{1}{9}\left(-b^2(a-b)^2 + \left(+3a^2 + 2ab - 2b^2\right)c^2 - c^4\right) \end{aligned} \tag{0.18}$$

The form for δ_1 has an interesting geometric interpretation. If $\delta_1 = 0$ we must have $|a-b| = |\sqrt{3}c|$. This means that the three roots, plotted in complex space, lie on the vertices of an equilateral triangle.

Next we calculate the discriminant, which boils down to

$$\Delta = -\frac{12}{81}c^2\left((a-b)^2 + c^2\right)^2$$

This shows that Δ is always negative for this type of cubic, and that we can take the square root symbolically.

$$\sqrt{-\Delta} = \frac{2}{9}\sqrt{3}\ c\left((a-b)^2 + c^2\right)$$

The next step is to evaluate $\bar{D}$ which, after some simplification, becomes

$$\bar{D} = -\tfrac{2}{27}(a-b)\left((a-b)^2 + 9c^2\right)$$

Now we need the quantities under the cube roots: $\left(-\bar{D} \pm A\sqrt{-\Delta}\right)/2$. After plugging in the earlier results and fooling around this simplifies to something really nice.

$$\frac{-\bar{D} \pm A\sqrt{-\Delta}}{2} = \left(\frac{a-b\pm\sqrt{3}c}{3}\right)^3$$

This is ever so convenient; we can take the cube roots symbolically and get

$$\begin{aligned} p &= \frac{a-b+\sqrt{3}c}{3} \\ q &= \frac{a-b-\sqrt{3}c}{3} \end{aligned} \tag{0.19}$$

I realize that I've used the calculation from algorithm A0 here, but since we are doing this all symbolically we would get the same result if we used algorithm A. In fact if you look at the expression for δ_1 in equation (0.18) you will notice that it can be factored into

$$\delta_1 = -\left(\frac{+(a-b)+\sqrt{3}c}{3}\right)\left(\frac{+(a-b)-\sqrt{3}c}{3}\right)$$

This is just the first row of equation (0.9)

We're near the end. Let's slow down a bit and look carefully at the final two operations in algorithm A. I've put a box around the key subtractions, the parenthesized expressions on either side are the values we are subtracting to get the result

$$\begin{aligned} \tilde{x} &= p + q \\ &= \tfrac{1}{3}\left(a-b+\sqrt{3}c\right)\boxed{-}\tfrac{1}{3}\left(-a+b+\sqrt{3}c\right) \\ &= \left(\tfrac{2}{3}a - \tfrac{2}{3}b\right) \end{aligned} \tag{0.20}$$

and the final anti-depressant subtraction

$$\begin{aligned} x &= \tilde{x} - B \\ &= \left(\tfrac{2}{3}a - \tfrac{2}{3}b\right)\boxed{-}\left(-\tfrac{1}{3}a - \tfrac{2}{3}b\right) \\ &= a \end{aligned} \tag{0.21}$$

These two subtractions are the most important numerically since any round-off errors don't have a chance to be mitigated by later additions. Now let's see what these hidden values tell us.

Trouble Zones for Algorithm A

From looking at equation (0.21) we can see that we are likely to be in trouble if the value of |*b*| is very much larger than |*a*|. The two sides of the main subtraction are going to be nearly equal to $-\frac{2}{3}b$ with the small residue of *a* tossed in. But those small amounts of *a* are all that remains after the subtraction, at considerable loss in accuracy. Similarly, looking at equation (0.20) we can see that we will have trouble if the value of |*c*| is very much larger than |*a*-*b*|.

This is easy to test using an implementation of algorithm A that computes answers to single precision floating point accuracy. We feed it with polynomials constructed from equation (0.16). Plug in (*a*,*b*,*c*)=(1,1,1000000), construct the polynomial, solve it, and you get back… drum roll… *a*=1.45833. This is not a very good approximation to 1. Or plug in (*a*,*b*,*c*)=(1,1000000,1000000) and you get back *a*=0.5. This is especially a burn since we often are interested in roots that are near zero as they are the ones just in front of us on the screen. In this case the calculation of the single real root has been poisoned by the existence of two other very large roots that are not even real. And you can crank up *b* and *c* to make the error arbitrarily bad until numbers start overflowing (I'm only going to worry about problems with no overflow in the internal calculations for now.)

What can we do?

Taking a cue from what we did when solving quadratic equations, let's look at the following parameter space transformation:

$$\begin{bmatrix} t & u \\ s & v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -D & C \end{bmatrix}$$

This is basically the same as reversing the order of coefficients, and then depressing these new coefficients. Effectively what we are doing is forming a new cubic (by exchanging the homogeneous *x* and *w* components) whose real root is 1/*a* instead of *a*. We then solve this cubic and invert the result to get *a*. All this is encoded in this new transformation matrix.

Applying the matrix to equation (0.2) gives us our new depressed coefficients

$$\begin{aligned} \tilde{A} &= D \\ \tilde{B} &= 0 \\ \tilde{C} &= D\left(BD - C^2\right) \\ \tilde{D} &= D\left(-D^2A + 3DCB - 2C^3\right) \end{aligned}$$

This time we have a homogeneous factor of *D* to toss out which leaves

$$\begin{aligned} \bar{A} &= 1 \\ \bar{B} &= 0 \\ \bar{C} &= BD - C^2 = \delta_3 \\ \bar{D} &= -D^2A + 3DCB - 2C^3 \\ &= -D\delta_2 + 2C\delta_3 \end{aligned}$$

The basic identity of equation (0.13), $\bar{D}^2 + 4\bar{C}^3 = -\tilde{A}^2\Delta$, remains true for our new version of $\bar{C}$ and $\bar{D}$ since we now have $\tilde{A} = D$ instead of $\tilde{A} = A$.

Finally, at the end the calculation we un-depress the roots by the transformation

$$\begin{bmatrix} x & w \end{bmatrix} = \begin{bmatrix} \tilde{x} & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -D & C \end{bmatrix}$$

Putting this all together we get Algorithm D.

Algorithm D

Calculate $\delta_1, \delta_2, \delta_3, \Delta$ as in algorithm A0

$$\tilde{A} = D$$
$$\bar{C} = \delta_3$$
$$\bar{D} = -D\delta_2 + 2C\delta_3$$

Calculate $p, q, \tilde{x}_1, \tilde{x}_2, \tilde{x}_3$ as in algorithm A

Finally un-depress the results by

$$\begin{bmatrix} x & w \end{bmatrix}_i = \begin{bmatrix} -D & \tilde{x}_i + C \end{bmatrix}$$

Trying this out on our bad test cases (*a*,*b*,*c*)=(1,1,1000000) and (*a*,*b*,*c*)=(1,1000000,1000000) we get back, in each case, *a*=1.0, right on the nose. Yay.

Which one to chose?

So we have two solution schemas. Both get the same result algebraically, but they differ numerically on various inputs. How do we know which one to use when? It's time for some numerical experiments. I started with the test bed that constructs a polynomial from equation (0.16) with known real root *a*. I then had it use both algorithm A and algorithm D to solve for *a* and compared the result with the known value of *a*. I display the error in the calculated version, a_{calc} as the base ten logarithm of the absolute relative error

$$error = \log_{10}\left(\left|\frac{a - a_{calc}}{a}\right|\right)$$

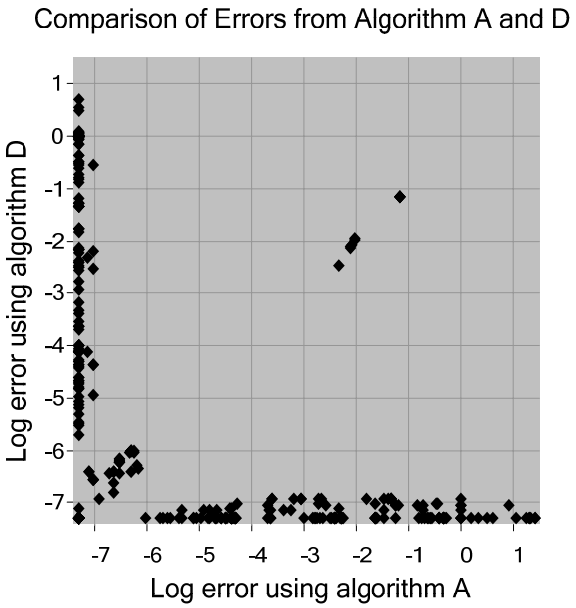
For reference, the least significant bit in a single precision IEEE float value in the range 1 to 2 has the value 1.1920929×10^{-7}. The smallest possible nonzero error would come from a value just below 2 which would show an error of

$$\log_{10}\left(\left|\frac{2 - 1.999999881}{2}\right|\right) = -7.224720001$$

For calculations that are spot on, with $a = a_{calc}$ exactly, this formula would give minus infinity; I have clamped that to -7.3 for display purposes.

Given this machinery I plugged in a set of values for (*a*,*b*,*c*) that cover the range from 10^{-8} to 10^{+8} in steps of 10^2. The results appear in the scatter plot of figure 2. Dots near the left side are when algorithm A gives a very small error; dots near the bottom are when algorithm D gives a very small error. The bottom left is where both errors are small. The encouraging thing is that, over this large range of inputs, at least one of the two techniques has a very small error (Except for a few cases in the middle, which happen at $a \approx b$ and $c \approx 0$. This is nearly a triple root situation. We'll look into this in more detail later.)

Figure 2.



So we know that (for the most part) one or the other of the algorithms will be good numerically. The question is: how do we know which one?

Let's look at the two candidate parameter space transformations

$$\begin{bmatrix} 1 & 0 \\ -B & A \end{bmatrix} vs. \begin{bmatrix} 0 & 1 \\ -D & C \end{bmatrix}$$

I originally thought as follows: if |*A*| is small the first matrix will be almost singular. Singular is bad. If |*D*| is small the second matrix will be almost singular. Singular is bad. Therefore we pick the matrix that is the least singular by comparing |*A*| with |*D*|. This works way better than just using algorithm A alone, but it's not quite right. Often a polynomial with a very small *A* gets a smaller error from algorithm A than from algorithm D.

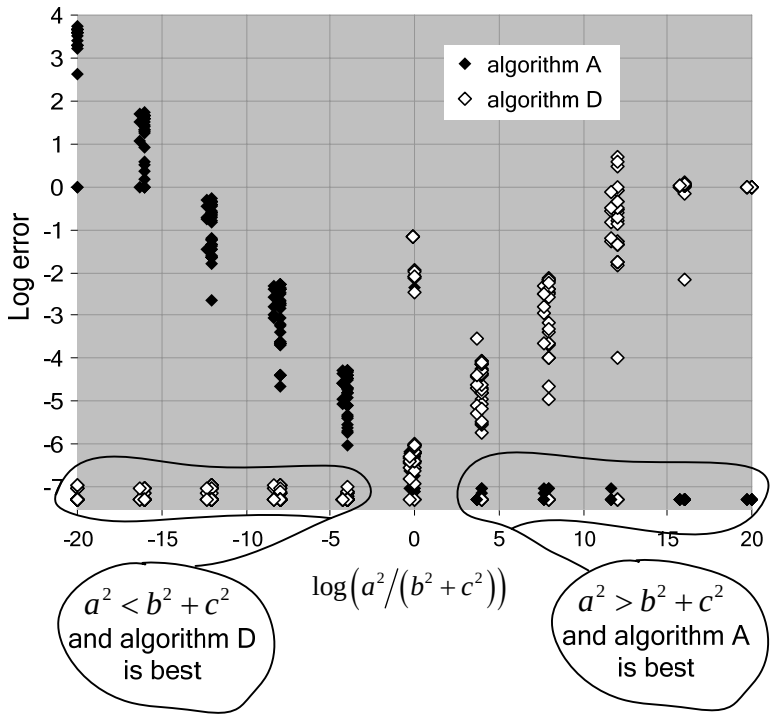
The correct criterion

After playing with a lot of plots and looking for patterns I finally came up with the correct criterion. And it's sort of obvious in retrospect (But I guess everything is obvious in retrospect.) The correct decision criterion is

if $a^2 > b^2 + c^2$ use Algorithm A,
otherwise use Algorithm D.

We can empirically see that this works by looking at figure 3. Here I have plotted the error due to algorithms A and D against the value $\log\left(a^2/\left(b^2+c^2\right)\right)$. Points to the left of zero have $a^2 < b^2 + c^2$ and points to the right of zero have $a^2 > b^2 + c^2$. The pattern is clear.

Figure 3. Correllation of Error with $a^2 - b^2 - c^2$



Why does this work?

But why does this work so well? There are two observations to make here.

First, reversing the order of the coefficients (as we are doing in algorithm D) inverts the sense of the test. That is, the polynomial with the coefficients reversed has the three roots

$$\left(\frac{1}{a}, \frac{1}{b+ic}, \frac{1}{b-ic}\right) = \left(\frac{1}{a}, \frac{b-ic}{b^2+c^2}, \frac{b+ic}{b^2+c^2}\right)$$

Solving this polynomial would return (a_r, b_r, c_r) where

$$a_r = \frac{1}{a},\ b_r = \frac{b}{b^2+c^2},\ c_r = \frac{c}{b^2+c^2}$$

Now play a bit and you can see that

$$\begin{aligned} {a_r}^2 - {b_r}^2 - {c_r}^2 &= \frac{1}{a^2} - \frac{b^2}{\left(b^2+c^2\right)^2} - \frac{c^2}{\left(b^2+c^2\right)^2} \\ &= \frac{b^2+c^2-a^2}{a^2\left(b^2+c^2\right)} \end{aligned}$$

This symmetry between the algorithms' transformation and the inversion of the choice test shows that we must be on the right track.

Second we look at the last two additions in algorithm A and algorithm D. For algorithm A we have seen that this is

$$\tilde{x} = \tfrac{1}{3}\left(a - b + \sqrt{3}c\right) \boxed{+} \tfrac{1}{3}\left(a - b - \sqrt{3}c\right)$$
$$x = \left(\tfrac{2}{3}a - \tfrac{2}{3}b\right) \boxed{-} \left(-\tfrac{1}{3}a - \tfrac{2}{3}b\right)$$

We have already noted that these two operations work best when *a* is large and *b*, *c* are small. Going through the same analysis for algorithm D yields, after some work

$$p = \frac{b^2 + c^2 - ab + \sqrt{3}ac}{3}$$
$$q = \frac{b^2 + c^2 - ab - \sqrt{3}ac}{3}$$
$$\begin{aligned}\tilde{x} &= p + q \\ &= \tfrac{1}{3}\left(-ab + b^2 + c^2 + \sqrt{3}ac\right) \\ &\quad \boxed{+} \tfrac{1}{3}\left(-ab + b^2 + c^2 - \sqrt{3}ac\right)\end{aligned}$$
$$\begin{aligned}w &= \tilde{x} + C \\ &= \tfrac{1}{3}\left(-2ab + 2b^2 + 2c^2\right) \boxed{+} \tfrac{1}{3}\left(+2ab + b^2 + c^2\right)\end{aligned}$$

We can see that the two boxed additions in the calculation of $\tilde{x}$ and *w* will work best when $b^2 + c^2$ is large and *a* is small.

One teensy tiny little detail

Well, this is very nice except for one thing. We are starting with *A,B,C,D* and we don't know the values of *a,b,c* until we solve for them. But we need them to determine which algorithm to use to solve for them. What we really need is some relatively simple function of *A,B,C,D* that has the same sign as $a^2 - b^2 - c^2$. How can we get this? Let's take a look at the boundary condition $a^2 = b^2 + c^2$. Plug this into equation (0.17) and you get

$$A = 1$$
$$B = -\tfrac{1}{3}\left(a + 2b\right)$$
$$C = \tfrac{1}{3}a\left(a + 2b\right)$$
$$D = -a^3$$

We can see that

$$\frac{A}{D} = -\frac{1}{a^3}, \quad \frac{B}{C} = \frac{-\tfrac{1}{3}\left(a + 2b\right)}{\tfrac{1}{3}a\left(a + 2b\right)} = -\frac{1}{a}$$

So

$$\frac{A}{D} = \frac{B^3}{C^3}$$

A good candidate for the decision test is, then

$$B^3D - AC^3$$

This works (but with a caveat). To see why it works, plug in the general internal structure values from equation (0.17). After some algebraic bashing you can factor it into

$$\begin{aligned}B^3D - AC^3 &= \tfrac{1}{27}\left(a^2 - b^2 - c^2\right) \\ &\quad \times\left(+a^2\left(b^2 + c^2\right) + 2ab\left(3c^2 - b^2\right) + \left(b^2 + c^2\right)^2\right)\end{aligned}$$

The first factor is our desired test quantity, good. What about the second factor? It's positive for *a*=0, but we'd like to know if it is positive for all values of *a,b,c*. Look at it as a quadratic polynomial in the variable *a* and calculate the discriminant. This works out to be $-c^2\left(3b^2-c^2\right)^2$. Since this can never be positive the second factor can never cross zero into negative values. The worst that can happen is that the second factor can be zero. This can happen in three situations: $\left(b=c=0\right)$, $\left(a=b,\ c=0\right)$ and $\left(a=-2b,\ c=\pm\sqrt{3}b\right)$.

The last two of these satisfy $a^2-b^2-c^2=0$ so the zeroness of the second factor does not affect the final test quantity. For the *b*=*c*=0 we always want to use Algorithm A, even though our new test $B^3D-AC^3=0$. We can catch this by making the test for algorithm A be $B^3D-AC^3\geq 0$. (Ultimately, though, we will want to look more closely at the situation where $a^2\approx b^2+c^2$ since we sometimes still have numerical problems there as you can see in figures 2 and 3.)

The Limiting Case

I was originally worried about using algorithm A and its transformation matrix when the value of *A* is near zero. But we've seen that we need to use algorithm A whenever the real root *a* is large. In the limit where *a* is infinity we have *A*=0 exactly. If *A*=0 the transformation matrix is totally singular.

$$\begin{bmatrix} 1 & 0 \\ -B & 0 \end{bmatrix}$$

A singular matrix will map whatever value we calculate for $\tilde{x}$ to the same [*x*,*w*] vector. This seems bad until we realize that that vector is in fact the correct answer: [1,0]. So the singularity doesn't bother us when calculating *a*. But there is another problem. If you follow through the calculations for algorithm A when *A*=0 you get.

$$\begin{aligned}
\delta_1 &= -B^2 \\
\delta_2 &= -BC \\
\delta_3 &= BD-C^2 \\
\tilde{A} &= 0 \\
\bar{C} &= -B^2 \\
\bar{D} &= -2B\delta_1 + A\delta_2 = 2B^3
\end{aligned}$$

The values of *p* and *q* are equal

$$p=q=\sqrt[3]{\frac{-\bar{D}+0\sqrt{-\Delta}}{2}}=-B$$

Then the depressed roots are

$$\begin{aligned}
\tilde{x}_1 &= -2B \\
\tilde{x}_2 &= -\frac{1}{2}\left(-2B\right)+\frac{\sqrt{3}}{2}\left(0\right)i = B \\
\tilde{x}_3 &= -\frac{1}{2}\left(-2B\right)-\frac{\sqrt{3}}{2}\left(0\right)i = B
\end{aligned}$$

And the roots in the original space are

$$\begin{aligned}
\begin{bmatrix} x & w \end{bmatrix}_1 &= \begin{bmatrix} -2B-B & 0 \end{bmatrix} \\
\begin{bmatrix} x & w \end{bmatrix}_2 &= \begin{bmatrix} B-B & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix} \\
\begin{bmatrix} x & w \end{bmatrix}_3 &= \begin{bmatrix} B-B & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}
\end{aligned}$$

In other words, when *a* is large algorithm A is real good at calculating *a* but real bad at calculating *b* and *c*. We can see this more generally by combining equation (0.15) for $\tilde{x}_2$ and (0.19) for *p* and *q* to get a sense of the internal values in effect when calculating *b* and *c*.

$$\begin{aligned}\tilde{x}_2 &= -\tfrac{1}{2}\left(\tfrac{1}{3}\left(a-b+\sqrt{3}c\right)\boxplus\tfrac{1}{3}\left(a-b-\sqrt{3}c\right)\right)+\tfrac{\sqrt{3}}{2}\left(\tfrac{1}{3}\left(a-b+\sqrt{3}c\right)\boxminus\tfrac{1}{3}\left(a-b-\sqrt{3}c\right)\right)i\\ &= \left(-\tfrac{1}{3}a+\tfrac{1}{3}b\right) + c\ \ i\end{aligned}$$

From this you can see that the calculation of *c* works best when *c* is large compared with |*a*-*b*|. The final anti-depressant addition is

$$\begin{aligned}x_2 &= \tilde{x}_2 - B\\ &= \left(-\tfrac{1}{3}a+\tfrac{1}{3}b\right)\boxminus\left(-\tfrac{1}{3}a-\tfrac{2}{3}b\right)+ci\\ &= b + ci\end{aligned}$$

From this you can see that the anti-depressant transformation works better when |*b*| is much larger than |*a*|. A symmetrical thing happens with algorithm D.

Final Algorithm

The net result of this is that, if we want to calculate *b* and *c* as well as *a* we need to use the following

> if $B^3D \geq AC^3$
>
> use algorithm A to calculate *a*,
>
> use algorithm D to calculate *b* and *c*
>
> else
>
> use algorithm D to calculate *a*,
>
> use algorithm A to calculate *b* and *c*.

This is very much like what we did to get the numerically stable solution to quadratic equations. We used the result of one transformation for one root and the result of the other transformation for the other root. We might not have seen it in this way, though, since the arithmetic is so much simpler for quadratics.

Next time

Next time we will investigate more general transformations that can depress the cubic polynomial, and later look at good ways to solve the case where $\Delta > 0$.

References

[1] M. Livio, *The Equation That Couldn't Be Solved: How Mathematical Genius Discovered the Language of Symmetry*, Simon and Schuster, 2005.

[2] F. Acton, *Real Computing Made Real*, Princeton University Press, 1996, pages 29 – 32.

[3] C. Loop and J. F. Blinn, *Real-Time GPU Rendering of Piecewise Algebraic Surfaces*, Proceedings of Siggraph 2006, ACM Press

[4] J. F. Blinn, *Polynomial Discriminants Part 1, Matrix Magic*, IEEE CG&A, Nov/Dec 2000, Reprinted in *Jim Blinn's Corner: Notation, Notation, Notation*, Chapter 19, pages 262 – 263, Morgan Kauffman, 2003.

[5] J. F. Blinn, *How to Solve a Quadratic Equation*, IEEE CG&A, Nov/Dec 2005, Volume 25, Number 6, pages 76 – 79

[6] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, 2002, pages 479 – 481.

[7] D. Herbison-Evans, *Solving Quartics and Cubics for Graphics*, in *Graphics Gems V*, A. W. Paeth editor, AP Professional, 1995, pages 11 – 12 .

[8] W. H. Press et al., *Numerical Recepies in C++*, Cambridge University Press, 2002, pages 190 – 191.