How to Solve a Cubic Equation

Part 5 -- Back to Numerics

James F. Blinn
Microsoft Research
blinn@microsoft.com

Originally published in
IEEE Computer Graphics and Applications
May/June 2007 (Vol. 27, No. 3) pages 78-89

The Goal

We've spent the last 4 columns studying the properties of the homogeneous cubic polynomial

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

In particular we've looked at various closed-form ways to find the roots of the equation $f(x,w)=0$, expressed as homogeneous parameter pairs $[x,w]$. In this article I'm going to introduce two new algorithms that, at first, look quite different from what we've done so far. It will turn out, though, that they actually do fit into our solution scheme. In showing this I've taken good ideas from a variety of authors (plus a few of my own) and translated them into a common notation while also converting them to deal with homogeneous polynomials.

Finally, I'll put everything together to come up with a root-finding algorithm that has the best numerical properties that I know of. I am not, however, going to say that the composite algorithm is perfect. There are still some nooks and crannies in the space of cubics where the accuracy is not as good as I would like. I've learned lot in researching and writing this series and I think I could play with this forever, but it's time though for a checkpoint on what we know so far.

You can follow along with the derivation or, if you don't like mysteries, you can jump directly to the Appendix to see the final algorithm. I've expressed the algorithm in more mathematical than computer notation and shaded all the algorithm fragments throughout the article to distinguish them from derivations. Also, just for emphasis, I've sometimes explicitly stated a redundant inverse conditional in the 'else' statements.

Invariants and Covariants

One tool that we've used to analyze the polynomial is the parameter space transformation

$$\begin{bmatrix} x & w \end{bmatrix} = \begin{bmatrix} \tilde{x} & \tilde{w} \end{bmatrix} \begin{bmatrix} t & u \\ s & v \end{bmatrix} \tag{0.2}$$

Classical invariant theory [Hilbert] investigates how this transformation affects a polynomial, and finds other polynomials and scalars whose relation to the original remains unchanged under the transformation. This means that they are effectively geometrically locked to the original and so must express some basic property of it. These auxiliary polynomials are collectively called covariants, and the scalars are called invariants. In the case of a cubic there are three covariants and one invariant. They are as follows:

First covariant

The original function itself $f(x,w)$ is considered a covariant in a somewhat trivial sense. Upon transformation by Equation (0.2) its coefficients change according to the formula

$$\begin{bmatrix}\tilde{A}\\ \tilde{B}\\ \tilde{C}\\ \tilde{D}\end{bmatrix}=\begin{bmatrix}t^3 & 3t^2u & 3tu^2 & u^3\\ t^2s & 2tus+t^2v & u^2s+2tuv & u^2v\\ ts^2 & us^2+2tsv & 2usv+tv^2 & uv^2\\ s^3 & 3s^2v & 3sv^2 & v^3\end{bmatrix}\begin{bmatrix}A\\ B\\ C\\ D\end{bmatrix} \tag{0.3}$$

Second covariant

The second covariant is the so-called Hessian of the cubic, a quadratic polynomial defined as the determinant of the matrix of second derivatives of *f*. The coefficients of this quadratic are

$$\begin{aligned}\delta_1 &= AC - B^2\\ \delta_2 &= AD - BC\\ \delta_3 &= BD - C^2\end{aligned} \tag{0.4}$$

The Hessian can be written in various notations as

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

The Hessian transforms according to the standard formula for transforming quadratics:

$$\begin{bmatrix}2\tilde{\delta}_1 & \tilde{\delta}_2\\ \tilde{\delta}_2 & 2\tilde{\delta}_3\end{bmatrix}=\begin{bmatrix}t & u\\ s & v\end{bmatrix}\begin{bmatrix}2\delta_1 & \delta_2\\ \delta_2 & 2\delta_3\end{bmatrix}\begin{bmatrix}t & s\\ u & v\end{bmatrix} \tag{0.5}$$

Third covariant

Hilbert calls this covariant the skew covariant and gives it the name *J* (I called it $\bar{D}$ in part 4 for reasons which will become clear shortly). It is a cubic with coefficients

$$\begin{aligned}A_J &= +A^2D - 3ABC + 2B^3\\ B_J &= -2AC^2 + ABD + B^2C\\ C_J &= -ACD + 2B^2D - BC^2\\ D_J &= -AD^2 + 3BCD - 2C^3\end{aligned} \tag{0.6}$$

The cubic, *J*, is then

$$J(x,w) = A_J x^3 + 3B_J x^2 w + 3C_J xw^2 + D_J w^3$$

This cubic also transforms according to Equation (0.3)

The Invariant

Finally, the scalar invariant is the determinant of the Hessian matrix **H**.

$$\Delta = \det\mathbf{H} = 4\delta_1\delta_3 - \delta_2^{\,2} \tag{0.7}$$

This is also known as the discriminant of the cubic.

The syzygy

We showed in part 3 that these three polynomials and one scalar together identically satisfy the equation

$$J^2 + \tfrac{1}{2}H^3 = -\Delta f^2 \tag{0.8}$$

This equation is called a syzygy:

Root Finding

The basic root finding algorithm requires four steps: depressing, scaling, solving and undepressing.

General algorithm

Depress

Transform the cubic according to equation (0.2) and (0.3) to make $\tilde{B} = 0$. The most general transform that does this is to pick any two values for (*t*,*u*) and then set

$$\begin{aligned} s &= -\kappa\left(t^2 B + 2tuC + u^2 D\right) \\ v &= +\kappa\left(t^2 A + 2tuB + u^2 C\right) \end{aligned} \tag{0.9}$$

The factor κ can be any nonzero quantity. In earlier articles I made this equal to 1, but I will play with it a bit later. We are now solving the polynomial

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

Scale

The new coefficients $\tilde{A}, \tilde{C}, \tilde{D}$ are polynomial functions of degree 3, 5 and 6 in the (*t*,*u*) values we picked in the previous step. We found in part 3 that, surprisingly, these polynomials all have a common factor:

$$\begin{aligned} \tilde{A}(t,u) &= f(t,u) \\ \tilde{C}(t,u) &= f(t,u) \times \tfrac{1}{2} H(t,u) \\ \tilde{D}(t,u) &= f(t,u) \times J(t,u) \end{aligned} \tag{0.10}$$

This means that we can toss out this common factor and instead solve the polynomial

$$\bar{x}^3 + 3\bar{C}\bar{x} + \bar{D} = 0 \tag{0.11}$$

where

$$\begin{aligned} \bar{C} &= \tfrac{1}{2} H(t,u) \\ \bar{D} &= J(t,u) \end{aligned} \tag{0.12}$$

This view of polynomial depression appeals to my sense of aesthetics. I'd always thought the process of depressing a cubic was a bit arbitrary. But equations (0.11) and (0.12) show that the two coefficients of the depressed polynomial are simply evaluations of the cubic's two covariants.

In what follows I'll use the letters *f*, $\frac{1}{2}H$ and *J* to refer to the functions and letters $\tilde{A}, \bar{C}$ and $\bar{D}$ to refer to the values of these functions for a particular choice of (*t*,*u*). In these terms the fundamental syzygy becomes

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

A couple of other notes: Equation (0.11) is not homogeneous. We can get away with this, setting $\tilde{w} = 1$, since Equation (0.11) cannot have a root at infinity. Also Equation (0.11) has the important property that its three roots sum to zero.

Solve

For $\Delta > 0$ we have three real roots, found by

$$\begin{aligned}
\theta &= \tfrac{1}{3}\operatorname{atan}_2\left(\tilde{A}\sqrt{\Delta}, -\bar{D}\right) \\
\tilde{x}_1 &= 2\sqrt{-\bar{C}}\left(\cos\theta\right) \\
\tilde{x}_2 &= 2\sqrt{-\bar{C}}\left(-\tfrac{1}{2}\cos\theta + \tfrac{\sqrt{3}}{2}\sin\theta\right) \\
\tilde{x}_3 &= 2\sqrt{-\bar{C}}\left(-\tfrac{1}{2}\cos\theta - \tfrac{\sqrt{3}}{2}\sin\theta\right)
\end{aligned} \tag{0.14}$$

For $\Delta < 0$ there is one real root and complex conjugate pair of roots, found by

$$\begin{aligned}
p, q &= \left(\frac{-\bar{D} \pm \tilde{A}\sqrt{-\Delta}}{2}\right)^{\frac{1}{3}} \\
\tilde{x}_1 &= p + q \\
\tilde{x}_2 &= -\tfrac{1}{2}(p+q) + \tfrac{\sqrt{3}}{2}(p-q)i \\
\tilde{x}_3 &= -\tfrac{1}{2}(p+q) - \tfrac{\sqrt{3}}{2}(p-q)i
\end{aligned} \tag{0.15}$$

The two values *p* and *q* come from the two choices for $\pm$ on the right side of the first equation. I've left aside numerical considerations for the moment to emphasize the nice symmetry between these two sets of expressions for the $\tilde{x}_i$'s. And as a check note that, sure enough, $\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_3 = 0$.

Un-depress

Finally we get from tilde'ed space back to the original polynomial by plugging the vectors $\begin{bmatrix}\tilde{x}_i & 1\end{bmatrix}$ into equation (0.2).

Degeneracies

It's worthwhile at this point to take a short look at what happens for the degenerate cases.

For $\Delta = 0$ we have a double root (Type 21). In this case equation (0.15) would get *p*=*q*. So the complex parts of $\tilde{x}_2$ and $\tilde{x}_3$ are zero and $\tilde{x}_2 = \tilde{x}_3$. Similarly, for $\Delta = 0$ equation (0.14) would find that θ was one of the three values: $0, -60°, +60°$ (depending on the sign of $\bar{D}$ and how the two-parameter arctangent function interprets edge cases). Equation (0.14) then results in one of $\tilde{x}_2 = \tilde{x}_3$, $\tilde{x}_3 = \tilde{x}_1$, or $\tilde{x}_1 = \tilde{x}_2$. The punchline is that a type 21 cubic nicely fits in as a special case of both the $\Delta > 0$ and the $\Delta < 0$ case.

For a triple-root polynomial (Type 3) we have all of $\delta_1 = \delta_2 = \delta_3 = \Delta = \bar{C} = \bar{D} = 0$. Equation (0.15) would get *p*=*q*=0 and so $\tilde{x}_1 = \tilde{x}_2 = \tilde{x}_3 = 0$. Equation (0.14) on the other hand is going to have a bit of trouble, as both arguments to the arctangent are zero. But whatever the arctangent returns in this case, we will still get $\tilde{x}_1 = \tilde{x}_2 = \tilde{x}_3 = 0$ because $\bar{C} = 0$. Ultimately the un-depressing transform then moves the triple root at zero to where it should be in un-tilde'ed space.

Special case algorithms

There are two special cases of this algorithm that are important.

Algorithm A is the "classic" algorithm that you see in most cubic solutions. It corresponds to (*t*,*u*)=(1,0) and the other values become

$$\text{Algorithm A}\begin{cases}\begin{bmatrix} t & u \\ s & v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -B & A \end{bmatrix} \\ \tilde{A}_A = A \\ \bar{C}_A = \delta_1 \\ \bar{D}_A = -2B\delta_1 + A\delta_2 \end{cases} \tag{0.16}$$

Algorithm D effectively reverses the order of the coefficients, applies algorithm A to solve for $1/\tilde{x}_i$ and then inverts the result. The intermediate values are

$$\text{Algorithm D}\begin{cases}\begin{bmatrix} t & u \\ s & v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -D & C \end{bmatrix} \\ \tilde{A}_D = D \\ \bar{C}_D = \delta_3 \\ \bar{D}_D = -D\delta_2 + 2C\delta_3 \end{cases} \tag{0.17}$$

When necessary I will put subscripts on the $\tilde{A}, \bar{C}, \bar{D}$ to emphasize that they came from these specific algorithms.

Another Approach

There is yet another algorithm that, at first, looks rather different from what we have seen so far. But when seen correctly it actually is a special case of our general algorithm. Let's look at two derivations

Hilbert's Algorithm

Hilbert [Hilbert] (on pages 69–71) describes an algorithm that, translated into our notation and glossing over a few details, goes as follows. We start with the syzygy of equation (0.8) and rearrange it into

$$+\tfrac{1}{2}H^3 = -\Delta f^2 - J^2$$

We can now factor the right side

$$\underbrace{\tfrac{1}{2}H^3}_{\text{order 6}} = \underbrace{\left(\sqrt{-\Delta}\,f - J\right)}_{\text{order 3}}\underbrace{\left(\sqrt{-\Delta}\,f + J\right)}_{\text{order 3}}$$

I've labeled the factors with their polynomial order to emphasize that these are polynomial functions. Looking at the square root we see that this formula is most practical when $\Delta < 0$ but it will work (using complex arithmetic) for all cubics.

Next we look at the quadratic *H*. It can be factored into the product of two linear functions

$$H = lm$$

Note that the factors *l* and *m* are not the roots of *H*; they are two linear functions $l(x,w) = l_0 x + l_1 w$ and $m(x,w) = m_0 x + m_1 w$ whose roots, [-l_1, l_0] and [-m_1, m_0], are the two roots of *H*. This perhaps subtle point is important to keep in mind to avoid confusion later.

Combining these last two equations gives

$$\tfrac{1}{2}l^3 m^3 = \left(\sqrt{-\Delta}\,f - J\right)\left(\sqrt{-\Delta}\,f + J\right)$$

so we can allocate the factors according to

$$\begin{aligned} \tfrac{1}{\sqrt{2}}\left(\sqrt{-\Delta}\,f - J\right) &= l^3 \\ \tfrac{1}{\sqrt{2}}\left(\sqrt{-\Delta}\,f + J\right) &= m^3 \end{aligned} \tag{0.18}$$

In other words the left hand sides of equation (0.18) equal triple-root (type 3) cubics that are the cubes of the two linear factors of *H*. We've actually seen this before, in a slightly different guise, in part 1 and part 4 of this series: The cubics *f* and *J* both share the same Hessian, as do all linear combinations $\alpha f + \beta J$. In cubic-curve space (shown in figures 3, 4 and 5 of part 1) the line consisting of all such cubics intersects the triple-root curve twice. These intersections are at points where the triple root is the cube of a root of the Hessian. In other words, for certain values of (α,β) the cubic $\alpha f + \beta J$ will be a type 3 cubic. We've just found that the magic values are $(\alpha,\beta) = \left(\sqrt{-\Delta}, \pm 1\right)$.

Anyway, let's now add the two equations of (0.18) together:

$$\sqrt{-2\Delta}\,f = l^3 + m^3 \tag{0.19}$$

Moving the square root to the other side and factoring this gives

$$f = \sqrt{-\tfrac{1}{2\Delta}}\,(l+m)\left(l^2 - lm + m^2\right)$$

So, one factor of the cubic *f* is simply the sum of the factors of the quadratic *H*. No cube roots seem to be necessary. This seems too good to be true, and it is. What Hilbert overlooked is that the separation of *H* into factors *l* and *m* is not unique. For any nonzero homogeneous factor *h*, the polynomials *hl* and *m*/*h* also work. You have to find the value of *h* that makes the *J* cancel out in equation (0.19). To do this you must do what Hilbert mentioned as an alternative computation: Find *l* and *m* as the cube roots of the type 3 polynomials $\sqrt{-\Delta}\,f - J$ and $\sqrt{-\Delta}\,f + J$. But there's another way to look at the problem that makes this easier.

The $\bar{C} = 0$ Algorithm

Let's back up a bit and take another approach. Look at equations (0.11) and (0.12) and ask yourself, what value of (*t*,*u*) can we pick that will make equation (0.11) as easy as possible to solve. How about picking (*t*,*u*) to be a root of the quadratic *H*? We know how to solve quadratics. This will make $\bar{C} = 0$ and our cubic depresses to

$$\tilde{x}^3 + \bar{D} = 0$$

So the calculation would be

$$\begin{aligned} &\begin{bmatrix} x_H & w_H \end{bmatrix} = \text{a root of } H \\ &\bar{D} = J\left(x_H, w_H\right) \\ &\tilde{x} = -\sqrt[3]{\bar{D}} \\ &s = -{x_H}^2 B - 2x_H w_H C - {w_H}^2 D \\ &v = +{x_H}^2 A + 2x_H w_H B + {w_H}^2 C \\ &\begin{bmatrix} x & w \end{bmatrix} = \begin{bmatrix} \tilde{x} & 1 \end{bmatrix} \begin{bmatrix} x_H & w_H \\ s & v \end{bmatrix} \end{aligned}$$

No funny *p*'s and *q*'s to mind.

Evaluating *J* at an arbitrary parameter involves finding all the coefficients from equation (0.6), so the arithmetic starts stacking up. But there's remarkable simplification. To show it I will start by stating an interesting identity, true for all values (*t*,*u*) and for (*s*,*v*) defined by equation (0.9) with $\kappa = 1$. The identity is

$$H(s,v) = \tfrac{1}{2}\left(H(t,u)\right)^2$$

You can prove this by direct substitution (hard) or by using the diagram techniques discussed in part 3 (easier). The identity means that if (*t*,*u*) is a root of *H*, then so is (*s*,*v*). It is, in fact… *the other root of H*. Constructing a transformation from the two roots of *H* makes the Hessian itself transform via equation (0.5) into:

$$\begin{bmatrix} t & u \\ s & v \end{bmatrix}\begin{bmatrix} 2\delta_1 & \delta_2 \\ \delta_2 & 2\delta_3 \end{bmatrix}\begin{bmatrix} t & s \\ u & v \end{bmatrix} = \begin{bmatrix} 0 & \tilde{\delta}_2 \\ \tilde{\delta}_2 & 0 \end{bmatrix}$$

We saw the result of this transformation a bit more geometrically in figure 1 of Part 1 where we showed that the intersection of the two surfaces $\tilde{\delta}_1 = 0, \tilde{\delta}_3 = 0$ consists of all cubics with $\tilde{B} = \tilde{C} = 0$.

So, we want the two rows of the ideal transformation to be the two roots of the quadratic *H*. We have a nice numeric formula for solving quadratics from one of my previous columns [Blinn], but it is a bit more general than we need. Relaxing the requirement that the formulas generate roots in sorted order gives a very pretty formula for the desired transformation

$$\begin{bmatrix} t & u \\ s & v \end{bmatrix} = \begin{bmatrix} -\delta_2 \pm \sqrt{-\Delta} & 2\delta_1 \\ 2\delta_3 & -\delta_2 \pm \sqrt{-\Delta} \end{bmatrix}$$

In this case [*s*,*v*] still obeys equation (0.9) but with a kappa value that is not 1. Either choice of the $\pm$ sign will work, having the effect of exchanging the roots. But we will make the sign choice to match the sign of $-\delta_2$; this ensures a pleasant addition. Some authors express this choice using the sign function: sgn.

$$t = v = -\delta_2 - \operatorname{sgn}(\delta_2)\sqrt{-\Delta} \qquad \text{NO!}$$

But this is absolutely NOT what we want to do, because the sgn function typically returns zero for an argument of zero. This seems to make sense, in the abstract, but it will get the wrong answer here if $\delta_2 = 0$. In that case we want *t* and *v* to be $\sqrt{-\Delta}$, not zero. I will instead define a variant of the sign function

$$\sigma(x) \equiv \text{if } (x < 0) \text{ then } -1 \text{ else } +1$$

This is actually simpler to implement in hardware as it just picks the sign bit from the parameter, with no testing for zero. In these terms the depression matrix is

$$\begin{bmatrix} t & u \\ s & v \end{bmatrix} = \begin{bmatrix} -\delta_2 - \sigma(\delta_2)\sqrt{-\Delta} & 2\delta_1 \\ 2\delta_3 & -\delta_2 - \sigma(\delta_2)\sqrt{-\Delta} \end{bmatrix} \qquad (0.20)$$

The cubic depressed according to the matrix in equation (0.20) has coefficients

$$\tilde{A} = f(t,u)$$
$$\tilde{D} = f(s,v)$$

Using a procedure similar to that used to prove equation (0.10) it is possible to show the nontrivial fact that these coefficients have a common factor of -4Δ; they are in fact:

$$\tilde{A} = -4\Delta\left(-\bar{D}_A - \sigma(\delta_2)A\sqrt{-\Delta}\right)$$
$$\tilde{D} = -4\Delta\left(-\bar{D}_D + \sigma(\delta_2)D\sqrt{-\Delta}\right)$$

So tossing out the common factor, the depressed cubic becomes

$$\bar{A}_H\tilde{x}^3 + \bar{D}_H\tilde{w}^3 = 0$$

with coefficients

$$\bar{A}_H = -\bar{D}_A - \sigma(\delta_2) A\sqrt{-\Delta}$$
$$\bar{D}_H = -\bar{D}_D + \sigma(\delta_2) D\sqrt{-\Delta}$$

and the solution is

$$\begin{bmatrix}\tilde{x} & \tilde{w}\end{bmatrix} = \begin{bmatrix}-\sqrt[3]{\bar{D}_H} & \sqrt[3]{\bar{A}_H}\end{bmatrix}$$

which un-depresses to

$$\begin{bmatrix}x & w\end{bmatrix} = \begin{bmatrix}-\sqrt[3]{\bar{D}_H} & \sqrt[3]{\bar{A}_H}\end{bmatrix}\begin{bmatrix}t & u\\ s & v\end{bmatrix}$$
$$= \underbrace{-\sqrt[3]{\bar{D}_H}\begin{bmatrix}t & u\end{bmatrix}}_{\text{root of } l} + \underbrace{\sqrt[3]{\bar{A}_H}\begin{bmatrix}s & v\end{bmatrix}}_{\text{root of m}}$$

The two vectors being added are just the roots of *H* (and thus the roots of *l* and *m*) appropriately scaled to make their sum be the root of *f*. So we can see that Hilbert's algorithm is just our general (*t,u*) depression algorithm with a special choice for (*t,u*), that being a root of *H*. Now let's see how this fares numerically.

The Numerical Situation

All these solution techniques work fine symbolically. Our next goal is to make sure these algorithms work well numerically. The main thing we need to do is find dangerous additions/subtractions and replace them with safe ones. A dangerous subtraction is one between two nearly equal quantities. The high order bits cancel out and give a result with only a few valid bits. A safe subtraction is one between two quantities of opposite sign, as the net result is an addition. The reverse condition characterizes safe/dangerous additions.

Un-Depression as a Source of Error

In part 2 we looked at round off error for type $1\,\bar{1}$ cubics and found a result that will turn out to be basic principle for all types: The un-depression calculation itself – the calculation that allowed us to solve the cubic in the first place – being the final bit of arithmetic in the algorithm, is the most dangerous numerical operation of all. We can see this geometrically in Figure 1 which shows the roots in the complex plane. After we solve the depressed cubic, where the sum of the roots is zero, we must un-depress them. For algorithm A this involves a translation by *–B* and a scale by 1/*A*. After homogeneous division the roots are *a*, *b+ic*, and *b-ic* where

$$a = \frac{x_1}{w_1},\ b = \mathrm{Re}\frac{x_2}{w_2},\ c = \mathrm{Im}\frac{x_2}{w_2}$$

You can see that if original cubic had a smallish *b* we would be calculating it by subtracting a nearly equal value during the un-depressing step: dangerous. The calculation of the largish *a* is OK though. The reverse happens for a cubic with a smallish *a* and largish *b*. This shows why algorithm A is generally good at computing roots with large values, and bad at computing roots with small values. In fact, in the limit, as *a* becomes infinitely large (and the coefficient *A* goes to zero) the un-depression transform becomes singular; it maps (almost) all [*x,w*] vectors to [1,0]. This is great for the root $a = \infty$. But, as we saw in Part 2, the singular matrix maps the other two roots to [0,0]. In other words, a depression transform that works optimally well for one root works optimally poorly for the other two roots. And this also applies to type111 cubics and, for that matter, to quadratics.

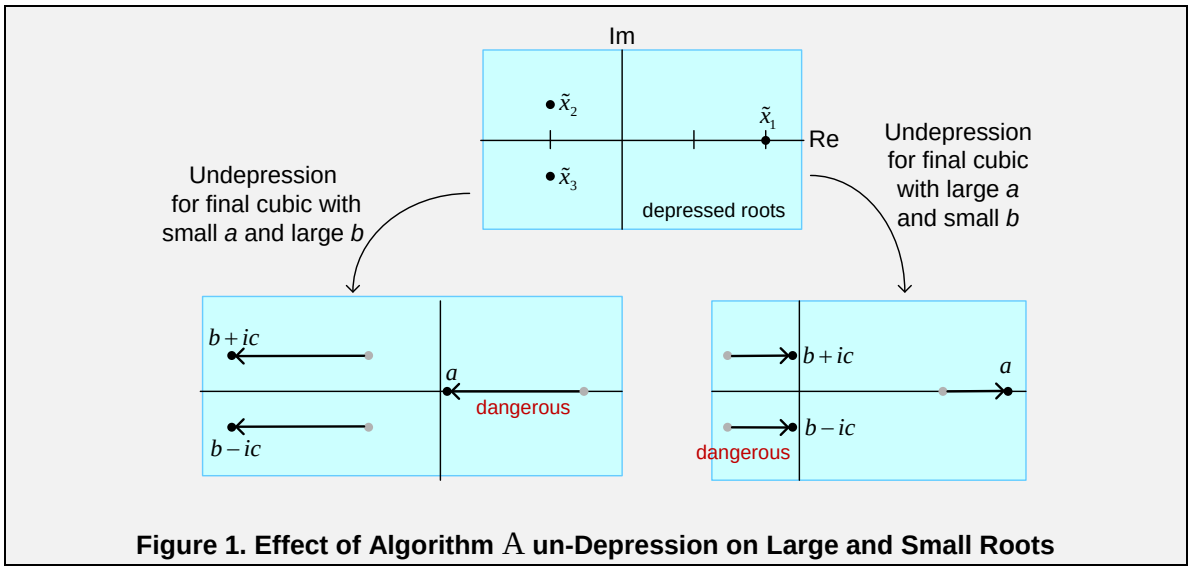

Figure 1. Effect of Algorithm A un-Depression on Large and Small Roots

Numerical Test Bed

True understanding comes from a combination of theoretical ideas and practical experience. So, to see how things really work, I have put together a numerical test bed that works as follows. I start with three desired roots and calculate the coefficients *A, B, C, D* of the cubic that has those roots. The tricky part is that there might be round off error in simply constructing this polynomial. I mitigate this by doing a brute force numerical refinement of the expected root values based on the actual polynomial represented by the calculated *A, B, C, D* coefficients. Then I pass the cubic through the root-finding algorithm and compare the results to the adjusted root values. Then I display the results as

$$bits = -\log_2\left(\left|\frac{a_{correct} - a_{calculated}}{a_{correct}}\right|\right).$$

This is a somewhat easier to understand error metric from what I used in part 3 (using base 10 logs). Using base two logs roughly gives the number of bits in the result that are correct. For single precision floating point, the smallest nonzero error (for an $a_{calculated}$ that is off in just the lowest bit) will generate a *bits* value between 23 and 24. An error of zero would give a value of infinity so I clamp it to 25 for display purposes.

Plots of error vs. (t,u)

Now let's see how varying the values of (t,u) affect the actual error. Figures 2 and 3 are plots of the numbers of correct bits returned from running our algorithm on two sample cubics while varying (t,u). I generate (t,u) in terms of an angle

$$t = \sin\psi$$
$$u = \cos\psi$$

and vary ψ from -90 to +90 degrees. (The range from +90 to +270 just generates the same (t,u) with signs flipped, so it's pretty much a repeat of the first range.) Algorithm A corresponds to a value of $\psi = 90°$ and Algorithm D to a value of $\psi = 0°$.

Figure 2 shows an example type111 cubic with roots in order from largest to smallest (in magnitude)

$$a_L = \tan(85°) = +11.430052$$
$$a_M = \tan(-45°) = -\ \ 1.000000$$
$$a_S = \tan(5°) = +\ \ 0.087489$$

This gives a more quantitative version of the phenomenon shown in figure 1. You can see that the algorithm works best to find a particular root when the values (*t*,*u*) match that root, but is terrible for the other two roots. This at first seems to imply that you need to know the answer in order to find the answer, not very useful. But you can see by figure 2 (and the zillions of others I have looked at) that one of either algorithm A or D has a pretty high correct-bit count. Algorithm A is perfect at finding a root at infinity, but still pretty good at largish roots. Algorithm D is perfect at finding a root at zero, but still pretty good at smallish roots. We'll build on this in our final algorithms.

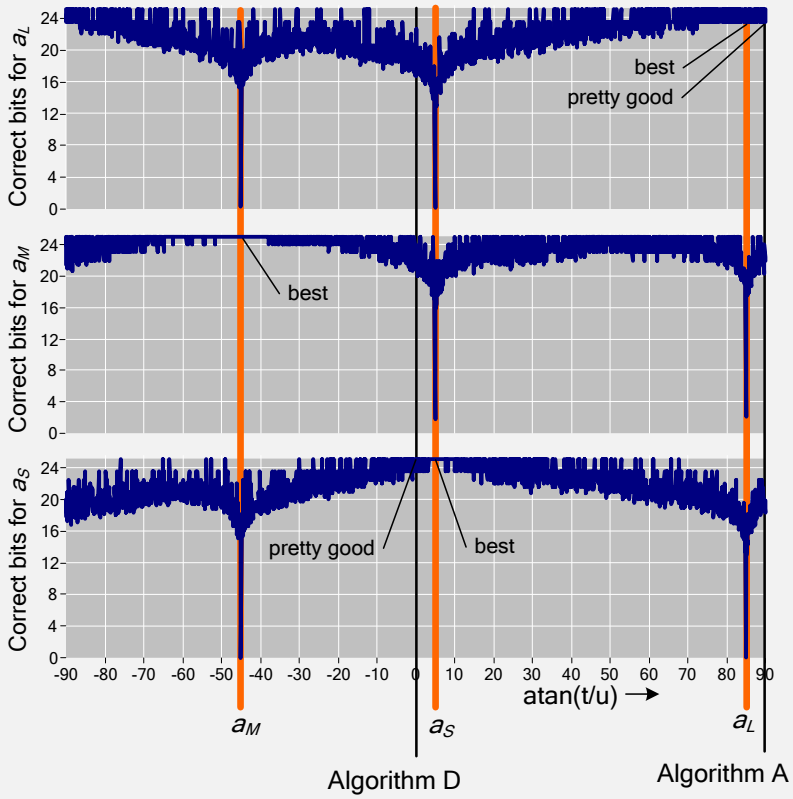

Figure 2 – Correct bits from varying the (t,u) parameter for Type 111 cubic.

Figure 3 shows a similar plot for a cubic with real root *a*=tan(-15)=-.26795, and complex conjugate roots $b \pm ic = \tan(60°) \pm i \times 0.2$. The figure also marks where (*t*,*u*) matches the roots of the Hessian, which are at 2.175=tan(65.3) and 1.453=tan(55.5). This torpedoes our hopes about an algorithm using (*t*,*u*) as a root of *H*. Since the roots of *H* are close to *b* when *c* is smallish, this choice of (*t*,*u*) will be far from the desired root *a* and will give bad numerical results.

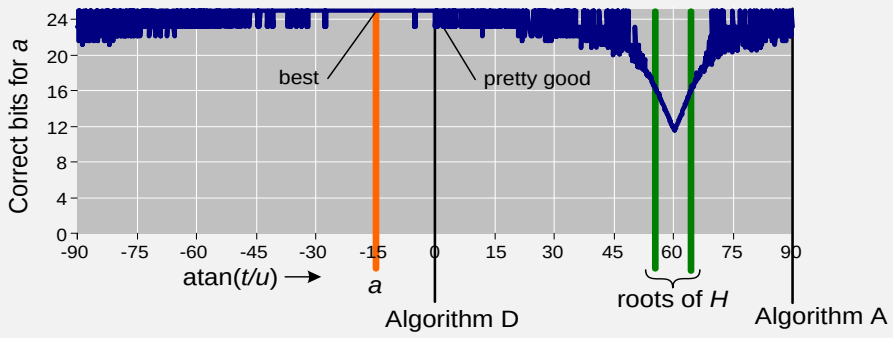

Figure 3 – Correct bits from varying the (t,u) parameter for Type 11bar cubic.

The La Porte Story

In my earlier articles I said that I had found very little non-trivial discussion of numerical fixes to closed form cubic root finders. Since then I have found one, in the work of French mathematician M. La Porte [LaPorte]. I have not been able to get a hold of the original paper, but the results of this work (without derivations) were reproduced as an example in page 240-243 of a paper by his colleague Jean Vignes [Vignes]. Basically, La Porte looked at each addition/subtraction in the conventional root-finding algorithm and, by purely algebraic means that Vignes describes as "non-trivial", found alternate formulations for almost all of the operations. These calculations generally take the form of

$$\begin{aligned} &\text{if} \quad (value1 > 0) \quad value2 = formula2a \\ &\text{else} \qquad\qquad\quad value2 = formula2b \end{aligned}$$

Here *formula2a* and *formula2b* both involve *value1* and are algebraically equivalent, but they have dangerous additions/subtractions in mutually exclusive situations. This is actually a direct extension of what we do in the quadratic case, where we effectively have

$$\begin{aligned} &\text{if} \quad (B > 0) \quad root = C/\left(-B - \sqrt{B^2 - AC}\right) \\ &\text{else } (B < 0) \quad root = \left(-B + \sqrt{B^2 - AC}\right)/A \end{aligned}$$

One reason La Porte's formulas might not have caught on is that they are wrong. Either due to typographical errors or other mistakes several of the code paths described in [Vignes] don't work or make sense. But the expressions looked tantalizingly familiar and I have spent many hours reverse engineering them and relating them to my notation and solution scheme. (Unfortunately we cannot consult LaPorte directly since he died in 1975 in a traffic accident). In the process I have gotten many good ideas from [Vignes] and have found where La Porte's formulations went wrong in one case (again perhaps due to a typographical error). I will bring these ideas in as they come up.

Numerics of Type 11bar

If $\Delta < 0$ we have one real root, *a*, and a complex conjugate pair $b \pm ic$. The numerically nice solution I arrived at in Part 2 was

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

Two strategies for real roots

In computer graphics we are usually only interested in the real roots of a polynomial. In this case the algorithm simplifies to the following, which I will call Algorithm AD

$$\begin{aligned}&\text{if } B^3D \geq AC^3\\&\quad\text{use algorithm A to calculate } a\\&\text{else}\\&\quad\text{use algorithm D to calculate } a\end{aligned} \tag{0.21}$$

La Porte's algorithm goes about this in another way. Translated into our notation it looks something like the following, which I'll call Algorithm L. (The test condition is slightly different which I'll address in a bit).

$$\begin{aligned}&\text{if } (test)\\&\quad\text{use algorithm A to calculate } a\\&\text{else}\\&\quad\text{use algorithm A to calculate } b \text{ and } c\\&\quad\text{use } b \text{ and } c \text{ to calculate } a\end{aligned} \tag{0.22}$$

The 'else' clause works because the product of minus the roots of a cubic equals the constant coefficient, D. For a homogeneous polynomial with a leading coefficient A that isn't necessarily 1, this translates into

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

So if we can calculate b and c safely, we can get a safely from

$$a = -\frac{D}{A\left(b^2+c^2\right)} \tag{0.23}$$

I'll expand on this later. First let's look at the calculations from equation (0.15) that the two strategies have in common.

Solve the Quadratic Really Properly

The values p^3 and q^3 are the two solutions to the quadratic equation $p^6 + \bar{D}p^3 - \bar{C}^3 = 0$, and in part 2 I showed the formula to calculate p and q numerically correctly. We chose p as the $\pm$ choice from equation (0.15) that corresponds to the safe addition. We can write this using our new sigma function as

$$\begin{aligned}p &= -\sigma\left(\bar{D}\right)\sqrt[3]{\frac{\left|\bar{D}\right| + \left|\tilde{A}\right|\sqrt{-\Delta}}{2}}\\ q &= -\sigma\left(\bar{D}\right)\sqrt[3]{\frac{\left|\bar{D}\right| - \left|\tilde{A}\right|\sqrt{-\Delta}}{2}}\end{aligned} \tag{0.24}$$

This choice for *p* and *q* means that we will always have $|p| \geq |q|$. It also shows that *p* will have the opposite sign of $\bar{D}$ as long as $\bar{D} \neq 0$. We then got rid of the unsafe subtraction for *q* by applying the identity $pq = -\bar{C}$. This gives the final calculation from in part 2:

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

This works almost all the time. The landmine here is when *p*=0. Equation (0.24) shows us that this can happen only if both $\bar{D} = 0$ and $\tilde{A}\sqrt{-\Delta} = 0$. And, from the syzygy in equation (0.13) this means $\bar{C} = 0$, and we are actually calculating $q = -\bar{C}/p = 0/0$. This seems even scarier. But this is a situation our quadratic solution algorithm from [Blinn] already takes care of; we're just not using it in its entirety. That algorithm had separate cases for $\bar{D}$ being positive, negative or zero. We've combined the positive and negative with our use of sigma, but including the zero case properly gives us

$$\begin{aligned} &\text{if} \quad (\bar{D} \neq 0) \quad p = \sqrt[3]{\frac{-\bar{D} - \sigma(\bar{D})|\tilde{A}|\sqrt{-\Delta}}{2}}, q = -\frac{\bar{C}}{p} \\ &\text{else} \ (\bar{D} = 0) \quad p = \sqrt[3]{\frac{|\tilde{A}|\sqrt{-\Delta}}{2}}, q = -p \end{aligned} \tag{0.25}$$

No more unpleasant divisions. But how close to zero does $\bar{D}$ have to be to trigger the final 'else' in equation (0.25)? I'm leery of testing floats for exactly zero; if it's simply very close to zero we could still have an overflow in $-\bar{C}/p$. Instead, I have had good luck with testing whether $\bar{D}$ is so small that it doesn't change the value of $|\tilde{A}|\sqrt{-\Delta}$ when added to it. This will make the 'else' clause kick in approximately when $|\bar{D}| < 2^{-24}|\tilde{A}|\sqrt{-\Delta}$. This can also happen when $p \neq 0$ but that's OK; we still will have *q* = -*p* in that case. The algorithm is

$$\begin{aligned} &T_0 = -\sigma(\bar{D})|\tilde{A}|\sqrt{-\Delta} \\ &T_1 = -\bar{D} + T_0 \\ &p = \sqrt[3]{\frac{T_1}{2}} \\ &\text{if} \quad (T_1 = T_0) \quad q = -p \\ &\text{else} \qquad\quad q = -\frac{\bar{C}}{p} \end{aligned} \tag{0.26}$$

This is one of the rare situations where testing for equality of two nonzero floating point numbers makes sense.

Add the Roots Nicely

The next step is the calculation

$$\tilde{x}_1 = p + q$$

If equation (0.26) gave us *q* = -*p* then we will have $\tilde{x}_1 = 0$ exactly. Otherwise the sum of *p* and *q* will be

$$\tilde{x}_1 = p - \frac{\bar{C}}{p}$$

This will be safe if $\bar{C} < 0$ but dangerous if $\bar{C} > 0$. Here is La Porte's first contribution -- a safe alternative calculation. First look at the identity

$$(p+q)\left(p^2 - pq + q^2\right) = p^3 + q^3$$

Recall the defining equations for *p* and *q* from part 2.

$$-pq = \bar{C}$$
$$-p^3 - q^3 = \bar{D}$$

Mash these together and get the algebraic identity

$$p + q = \frac{-\bar{D}}{p^2 + \bar{C} + q^2}$$

The right side is going to be safe if $\bar{C} > 0$ since all the quantities in the denominator are positive. The algorithm is

$$\begin{array}{lll} \text{if} & \left(\bar{C} \le 0\right) & \tilde{x}_1 = p + q \\ \text{else} & \left(\bar{C} > 0\right) & \tilde{x}_1 = \dfrac{-\bar{D}}{p^2 + q^2 + \bar{C}} \end{array} \qquad (0.27)$$

The 'else' option of equation (0.27) is actually much less important for Algorithm AD than it is for algorithm L. To see why let's look at values of *p* and *q* as we did in Part 2, in terms of the roots *a* and $b \pm ic$. We found that when we are running algorithm A we have:

$$\bar{C}_A = \tfrac{1}{9}\left(-(a-b)^2 + 3c^2\right)$$
$$p = \tfrac{1}{3}\left(a - b + \sqrt{3}c\right)$$
$$q = \tfrac{1}{3}\left(a - b - \sqrt{3}c\right)$$

The simple sum of *p* and *q* will give us the most numerical trouble when

$$|a-b| \ll \left|\sqrt{3}c\right|$$

Figure 4 shows the roots in the complex plane. The danger zone is where the roots $b \pm ic$ are in the area shaded in red (which is stretched horizontally for better visualization). All cubics in this region will trigger the 'else' clause of equation (0.27). This is necessary for algorithm L since it always uses Algorithm A. But algorithm AD only uses algorithm A when $a^2 > b^2 + c^2$. This corresponds to the green area inside the circle in figure 4. The intersection of the circle with the red area is not zero, but it is extremely small. So simply using $\tilde{x}_1 = p + q$ for algorithm AD works almost all the time.

Figure 4

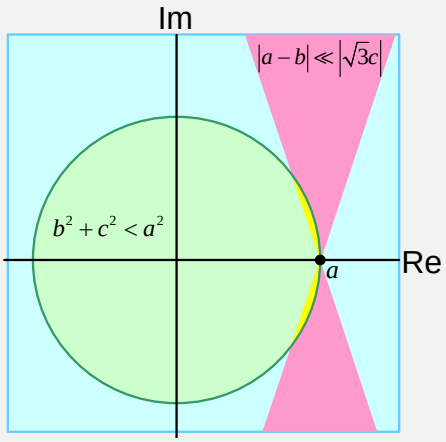

Figure 4. Region (yellow) where Algorithm A needs the else clause of equation (0.27)

Un-depress

Next, look at the un-depress step.

Algorithm AD

For algorithm AD the un-depression depends on our decision variable for which of algorithms A or D we used, and thus which un-depression matrix to use. We have

$$\begin{aligned} &\text{if} \quad \left(B^3 D \geq A C^3\right) \quad [x \quad w]_1 = [\tilde{x}_1 - B \quad A] \\ &\text{else} \qquad\qquad\quad [x \quad w]_1 = [-D \quad \tilde{x}_1 + C] \end{aligned} \tag{0.28}$$

Our A vs. D decision test picks the expression with the safe addition.

Algorithm L

For algorithm L our strategy from equations (0.22) and (0.23) gives

$$\begin{aligned} &\text{if} \quad (test) \quad [x \quad w]_1 = [\tilde{x}_1 - B \quad A] \\ &\text{else} \qquad\quad [x \quad w]_1 = \left[-D \quad A\left(b^2 + c^2\right)\right] \end{aligned}$$

Now the un-depressed values b and c are

$$b = \frac{-\frac{1}{2}(p+q) - B}{A}$$

$$c = \frac{\frac{\sqrt{3}}{2}(p-q)}{A}$$

so with a little algebra we get

$$b^2 + c^2 = \frac{B^2 + (p+q)B + \left(p^2 - pq + q^2\right)}{A^2}$$

We've seen the two parenthesized expressions before. Only one of them is safe numerically but we've already done the work to figure out which one when we calculated $\tilde{x}_1$. We can simply use

$$b^2 + c^2 = \frac{B^2 + \tilde{x}_1 B - \dfrac{\bar{D}}{\tilde{x}_1}}{A^2}$$

The first term on the numerator is obviously positive. The last term is positive too (since both $\tilde{x}_1 = p + q$ and $|p| \geq |q|$ together mean that the sign of $\tilde{x}_1$ is the same as the sign of *p*, and this is opposite to the sign of $\bar{D}$.) The middle term is positive when $\tilde{x}_1$ and *B* have the same signs. This becomes our test condition and Algorithm L finishes up with an un-depression with all operations being safe:

$$\text{if} \quad (B\tilde{x}_1 \leq 0) \quad a = \frac{\tilde{x}_1 - B}{A}$$

$$\text{else} \quad (B\tilde{x}_1 > 0) \quad a = -\frac{AD}{B^2 + \tilde{x}_1 B - \dfrac{\bar{D}}{\tilde{x}_1}}$$

This is correct and works. The printed version of the algorithm in [Vignes] erroneously shows the middle term on the denominator as (in our notation) $\bar{D}B$ instead of $\tilde{x}_1 B$.

Homogenized, the algorithm fragment would look like

$$\text{if} \quad (B\tilde{x}_1 \leq 0)$$

$$\begin{bmatrix} x & w \end{bmatrix}_1 = \begin{bmatrix} \tilde{x}_1 - B & A \end{bmatrix}$$

$$\text{else} \quad (B\tilde{x}_1 > 0)$$

$$\begin{bmatrix} x & w \end{bmatrix}_1 = \begin{bmatrix} -AD\tilde{x}_1 & B^2\tilde{x}_1 + \tilde{x}_1{}^2 B - \bar{D} \end{bmatrix}$$

Which one is better?

After a substantial numerical testing I've found that both algorithm AD and algorithm L seem to work equally well. As long as the cubic is not close to being of type 3, both algorithms calculate the real root with 23 or more accurate bits. But given that algorithm L is a bit more complicated arithmetically I'm currently leaning toward algorithm AD.

Numerics of Type111

To address the numerics of Type111 cubics we begin by recalling that Algorithm A is good at finding roots that are large (close to infinity) and Algorithm D is good at finding roots that are small (close to zero). Our basic strategy will be

Use Algorithm A to find the root $\begin{bmatrix} x_L & w_L \end{bmatrix}$
  where $x_L / w_L = a_L$ is the largest
  of the three roots in absolute value
Use Algorithm D to find the root $\begin{bmatrix} x_S & w_S \end{bmatrix}$
  where $x_S / w_S = a_S$ is the smallest
  of the three roots in absolute value
Use these two roots to find $\begin{bmatrix} x_M & w_M \end{bmatrix}$
  the root in the middle

Trigonometry and Root Ranges

Most published algorithms calculate the angle θ from equation (0.14) as

$$\theta = \tfrac{1}{3}\cos^{-1}\left(\frac{\bar{D}}{2\bar{C}\sqrt{-\bar{C}}}\right)$$

LaPorte's suggestion for Type111 cubics—and it's a good one—is to instead use the arctangent.

$$\theta = \tfrac{1}{3}\text{atan2}\left(\tilde{A}\sqrt{\Delta}, -\bar{D}\right)$$

This is numerically nicer for two reasons. First, the arccosine function has a very high derivative near argument values of -1 and +1. This means that theta will be very sensitive to inaccuracies in the inputs near those values. Secondly, as I mentioned in part 4, round off error in calculating the argument to the arccosine may produce a value slightly above +1 or below -1 and the arccosine function dies. The arctangent has neither of these problems. It is, however, worthwhile to test this numerically to make sure any advantage due to the arctangent isn't swamped by other errors. I've done that and convinced myself that the arctangent is definitely better than the arccosine. Not miraculously better, but better.

Now let's see how to find the largest root after un-depression via algorithm A. When we calculated θ using the arccosine we always got $0° \le \theta \le 60°$. Figure 5a shows the range that the $\tilde{x}_i$ values could take. When interpreting this figure, remember that we can also write the calculation of the $\tilde{x}_i$ values from equation (0.14) as

$$\tilde{x}_1 = 2\sqrt{-\bar{C}}\cos(\theta)$$
$$\tilde{x}_2 = 2\sqrt{-\bar{C}}\cos(\theta - 120°)$$
$$\tilde{x}_3 = 2\sqrt{-\bar{C}}\cos(\theta + 120°)$$

After algorithm A's un-depression (basically a translation) you can see that the largest un-depressed root must come from either $\tilde{x}_1$ or $\tilde{x}_3$. But if you calculate θ using the arctangent you might also get it in the range $-60° \le \theta \le 0°$, which places the roots in the ranges shown in figure 5b. Here the outside two roots are $\tilde{x}_1$ and $\tilde{x}_2$. Looking back at equation (0.14) you can see that flipping the sign of θ simply interchanges $\tilde{x}_2$ and $\tilde{x}_3$. So we can make our calculations more convenient by simply taking the absolute value of θ and only looking at $\tilde{x}_1$ and $\tilde{x}_3$.

Figure 5.

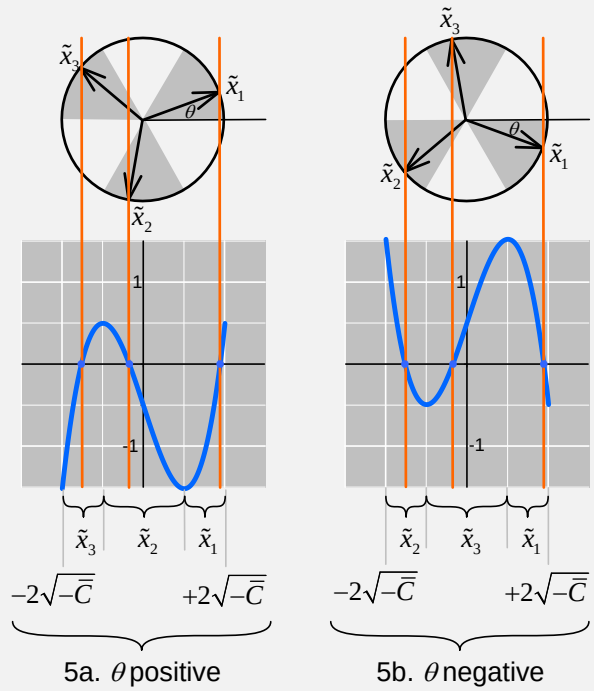

Figure 5. Ranges of depressed roots for positive and negative theta

Finding the largest/smallest roots

Now we must decide, when using algorithm A, which of $\tilde{x}_1$ or $\tilde{x}_3$ will ultimately un-depress into the root with maximum magnitude. Our decision criterion is

$$\left|\frac{\tilde{x}_1 - B}{A}\right| > \left|\frac{\tilde{x}_3 - B}{A}\right|$$

We can simplify this criterion a bit by squaring both sides and tossing out the positive factor A^2 and the positive term B^2 from both sides giving

$$\tilde{x}_1^{\,2} - 2\tilde{x}_1 B > \tilde{x}_3^{\,2} - 2\tilde{x}_3 B$$

Move the B terms to the right and the squared terms to the left. Then factor and get

$$(\tilde{x}_1 + \tilde{x}_3)(\tilde{x}_1 - \tilde{x}_3) > 2B(\tilde{x}_1 - \tilde{x}_3)$$

and since $\tilde{x}_1 - \tilde{x}_3 > 0$ we can toss out that common factor and come up with the final algorithm for finding the root with the largest magnitude.

$$\begin{aligned}
&\theta_A = \tfrac{1}{3}\left|\operatorname{atan}_2\left(A\sqrt{\Delta}, -\bar{D}_A\right)\right| \\
&\tilde{x}_{1A} = 2\sqrt{-\bar{C}_A}\left(\cos\theta_A\right) \\
&\tilde{x}_{3A} = 2\sqrt{-\bar{C}_A}\left(-\tfrac{1}{2}\cos\theta_A - \tfrac{\sqrt{3}}{2}\sin\theta_A\right) \\
&\text{if } \left(\tilde{x}_{1A} + \tilde{x}_{3A} > 2B\right) \\
&\qquad \text{then} \quad \tilde{x}_L = \tilde{x}_{1A} \\
&\qquad \text{else} \quad \tilde{x}_L = \tilde{x}_{3A} \\
&\left[x_L \quad w_L\right] = \left[\tilde{x}_L - B \quad A\right]
\end{aligned} \tag{0.29}$$

A similar analysis for the algorithm D pass, to get the smallest root, gives

$$\begin{aligned}
&\theta_D = \tfrac{1}{3}\left|\operatorname{atan}_2\left(D\sqrt{\Delta}, -\bar{D}_D\right)\right| \\
&\tilde{x}_{1D} = 2\sqrt{-\bar{C}_D}\left(\cos\theta_D\right) \\
&\tilde{x}_{3D} = 2\sqrt{-\bar{C}_D}\left(-\tfrac{1}{2}\cos\theta_D - \tfrac{\sqrt{3}}{2}\sin\theta_D\right) \\
&\textit{if}\ \left(\tilde{x}_{1D} + \tilde{x}_{3D} < 2C\right) \\
&\qquad \textit{then}\quad \tilde{x}_S = \tilde{x}_{1D} \\
&\qquad \textit{else}\quad \tilde{x}_S = \tilde{x}_{3D} \\
&\begin{bmatrix} x_S & w_S \end{bmatrix} = \begin{bmatrix} -D & \tilde{x}_S + C \end{bmatrix}
\end{aligned} \tag{0.30}$$

I've put lots of subscript *A*'s and *D*'s on the various intermediate values here to emphasize that they are different and to show that both these code blocks could be calculated in parallel.

The Third Root

In the non-homogeneous case we now have the largest root $a_L = x_L / w_L$ and the smallest root $a_S = x_S / w_S$. What is the middle root, a_M ? Again, minus the product of the roots equals *D*/*A* so

$$a_M = -\frac{D}{A a_L a_S}$$

In the homogeneous case we need to do the following: We express the cubic as the product of the three linear factors $\left(xw_L - wx_L\right)$, $\left(xw_M - wx_M\right)$ and $\left(xw_S - wx_S\right)$. Expand this product out, collect like terms and match up with equation (0.1) to get

$$\begin{aligned}
A &= w_L w_M w_S \\
3B &= -w_L w_M x_S - w_L x_M w_S - x_L w_M w_S \\
3C &= +x_L x_M w_S + x_L w_M x_S + w_L x_M x_S \\
D &= -x_L x_M x_S
\end{aligned} \tag{0.31}$$

The simplest way to proceed at this point is to take the first and last equations and get

$$\begin{bmatrix} x_M & w_M \end{bmatrix} = \begin{bmatrix} -\dfrac{D}{x_L x_S} & \dfrac{A}{w_L w_S} \end{bmatrix}$$

Eliminating the divisions, this is homogeneously equivalent to

$$\begin{bmatrix} x_M & w_M \end{bmatrix} = \begin{bmatrix} -D w_L w_S & A x_L x_S \end{bmatrix} \tag{0.32}$$

This works almost all the time (there's that phrase again). It has problems in two situations. If the largest root is infinity then w_L=0 and also *A*=0 so we get

$$\begin{bmatrix} x_M & w_M \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

This is not good. Similarly if the smallest root is zero we would have x_L=0, *D*=0 and we also get [0,0]. What can we do?

What we are really doing here is polynomial division. We want to divide the cubic by the quadratic formed by the two known roots. Homogeneous polynomial division is a bit different than regular polynomial division and is a major discussion in it own right. I'll just pick out the essentials that are useful for this specific case. Let's give names to the coefficients of the quadratic

$$Ex^2 + 2Fxw + Gw^2 = \left(xw_L - wx_L\right)\left(xw_S - wx_S\right)$$

which gives us

$$E = w_L w_S$$
$$F = -x_L w_S - w_L x_S$$
$$G = x_L x_S$$

we can then write equation (0.31) as

$$\underbrace{\begin{bmatrix} x_M & w_M & -1 \end{bmatrix}}_{WANTED} \underbrace{\begin{bmatrix} 0 & -E & -F & -G \\ E & F & G & 0 \\ A & 3B & 3C & D \end{bmatrix}}_{KNOWN} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

Linear algebra tells us that this is possible only if the determinants of the four 3x3 submatrices of the 3x4 matrix are zero. If you expand this out you will see that this is just equivalent to the condition that the quadratic is a factor of the cubic. Since it is indeed a factor, we see that we can find the solution vector $\begin{bmatrix} x_M & w_M & -1 \end{bmatrix}$ as the cross product of any two columns of the 3x4 matrix. Our first guess was to take the outer two columns whose cross product is

$$\begin{bmatrix} 0 \\ E \\ A \end{bmatrix} \times \begin{bmatrix} -G \\ 0 \\ D \end{bmatrix} = \begin{bmatrix} DE & -AG & EG \end{bmatrix}$$

Now the third component of this will not generally equal -1; what we actually are solving for is a homogeneous scale of the desired vector: $h\begin{bmatrix} x_M & w_M & -1 \end{bmatrix}$. But what we absolutely must have is that the third component of the cross product *not* be zero. In this case that component is $EG = x_L x_S w_L w_S$ which could be zero in reasonable circumstances, so that choice of cross product is dangerous. But there are six possible pairs of columns that we can use for the cross product. Probably the best choice is the middle two, whose cross product gives us

$$h\begin{bmatrix} x_M & w_M & -1 \end{bmatrix} =$$
$$\begin{bmatrix} 3CF - 3BG & -3BF + 3CE & F^2 - EG \end{bmatrix}$$

We can toss out the common factor of 3 from the first two components and get the net result for the third root as

$$\begin{aligned} E &= w_L w_S \\ F &= -x_L w_S - w_L x_S \\ G &= x_L x_S \\ \begin{bmatrix} x_M & w_M \end{bmatrix} &= \begin{bmatrix} CF - BG & -BF + CE \end{bmatrix} \end{aligned} \qquad (0.33)$$

The only time this will go wrong is when $F^2 - EG = 0$ which is when the quadratic has a double root. In this case — when the largest and smallest roots of the cubic are equal — it means that we must have a triple-root cubic. But in fact, this will only really cause problems if the triple root is at zero or infinity. We will deal with this another time.

La Porte's Algorithm

The use of the arctangent is the only idea I've been able to use from La Porte's algorithm for type 111 cubics. He gives a formula for the three-root case, but it has an undefined variable in it and I have not yet been able to figure out what that was supposed to have been. But I'm not through playing with it. If I come up with something I'll report it later.

Problem Areas

There are three numerical problem areas in finding roots of a cubic. The first of these is when the roots are well separated but one is very small and another is very large. The algorithm presented here pretty much has this one nailed. I've beaten on it with cubics covering a wide range of root values and the only situation where it gets less than 23 correct bits is near the other two problem areas: double roots and triple roots.

Double and triple root cubics actually work fine if Δ exactly equals zero (double root) or if $\delta_1 = \delta_2 = \delta_3$ exactly equals zero (triple root). The problem comes when our cubic is only close to having a double or triple root. The algorithm doesn't completely fall apart here, but the accuracy is not terrific. The main source of error is in a dangerous subtraction in the calculation of Δ (equation (0.7)) and worse yet in $\delta_1, \delta_2, \delta_3$ (equation (0.4)). Being skeptical I actually confirmed this by doing these subtractions in double precision and accuracy was indeed better. An open question is whether it is possible to find near-double and near-triple roots accurately with only single precision arithmetic. This is a topic for later.

Another thing that needs attention is the problem of numerical overflow. Many of the intermediate calculations in root finding can overflow even if the roots themselves do not. Since we are dealing with homogeneous polynomials a simple homogeneous scaling of *A, B , C* and *D* can get rid of this problem. The trick is finding what that scale should be.

Final Thoughts

Getting the right answer takes a bunch more arithmetic than a straightforward implementation of the classic algorithm. The basic idea is to carve up the 4D space of cubic polynomials (coordinatized by *A,B,C,D*) into separate regions. Inside each region one particular formula gives a numerically good calculation of the roots. But there are several cases where a region is very small and so we encounter such cubics extremely rarely. Nevertheless, we still need an extra test or a more complex calculation to identify and handle these cases. Examples are equations (0.26), (0.27) and the trade-off between equations (0.32) and (0.33). In fact, these are all cases where we are close to having a triple root. This somehow seems inefficient, but in an era where a floating point multiplication is often cheaper than a memory access, arithmetic doesn't scare us so much.

References

[Hilbert] D. Hilbert, *Theory of Algebraic Invariants*, Cambridge University Press, 1993. (This is a reprint of a series of lectures given in 1897)

[Blinn] J. Blinn, "How to solve a Quadratic Equation, Part 2," *IEEE CG&A*, vol. 26, no. 2, 2006, p. 82

[Vignes] J. Vignes, "New Methods for Evaluating the Validity of the Results of Mathematical Computations", *Mathematics and Computers in Simulation XX*, North-Holland Publishing Company, 1978, pages 227-249,.

[LaPorte] M. La Porte, "Une formulation Nume'riquement Stable Donnant les Racines Re'elles de l'E'quation du 3'eme Degre', *IFP Report*, no. 21516, October 1973.

Appendix – The algorithm in one piece

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

if $\Delta < 0$

if $\left(B^3D \geq AC^3\right)$

$\tilde{A} = A, \quad \bar{C} = \delta_1, \quad \bar{D} = -2B\delta_1 + A\delta_2$

else

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

$$T_0 = -\sigma\left(\bar{D}\right)\left|\tilde{A}\right|\sqrt{-\Delta}$$
$$T_1 = -\bar{D} + T_0$$
$$p = \sqrt[3]{\frac{T_1}{2}}$$

if $\left(T_1 = T_0\right) \quad q = -p$

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

if $\left(\bar{C} \leq 0\right) \quad \tilde{x}_1 = p + q$

else $\left(\bar{C} > 0\right) \quad \tilde{x}_1 = \frac{-\bar{D}}{p^2 + q^2 + \bar{C}}$

if $\left(B^3D \geq AC^3\right) \quad \left[x \quad w\right]_1 = \left[\tilde{x}_1 - B \quad A\right]$

else $\quad \left[x \quad w\right]_1 = \left[-D \quad \tilde{x}_1 + C\right]$

if $\Delta > 0$

$$\theta_A = \tfrac{1}{3}\left|\operatorname{atan}_2\left(A\sqrt{\Delta}, -\bar{D}_A\right)\right|$$
$$\tilde{x}_{1A} = 2\sqrt{-\bar{C}_A}\left(\cos\theta_A\right)$$
$$\tilde{x}_{3A} = 2\sqrt{-\bar{C}_A}\left(-\tfrac{1}{2}\cos\theta_A - \tfrac{\sqrt{3}}{2}\sin\theta_A\right)$$
if $\left(\tilde{x}_{1A} + \tilde{x}_{3A} > 2B\right)$

then $\quad \tilde{x}_L = \tilde{x}_{1A}$

else $\quad \tilde{x}_L = \tilde{x}_{3A}$

$$\left[x_L \quad w_L\right] = \left[\tilde{x}_L - B \quad A\right]$$

$$\theta_D = \tfrac{1}{3}\left|\operatorname{atan}_2\left(D\sqrt{\Delta}, -\bar{D}_D\right)\right|$$
$$\tilde{x}_{1D} = 2\sqrt{-\bar{C}_D}\left(\cos\theta_D\right)$$
$$\tilde{x}_{3D} = 2\sqrt{-\bar{C}_D}\left(-\tfrac{1}{2}\cos\theta_D - \tfrac{\sqrt{3}}{2}\sin\theta_D\right)$$
if $\left(\tilde{x}_{1D} + \tilde{x}_{3D} < 2C\right)$

then $\quad \tilde{x}_S = \tilde{x}_{1D}$

else $\quad \tilde{x}_S = \tilde{x}_{3D}$

$$\left[x_S \quad w_S\right] = \left[-D \quad \tilde{x}_S + C\right]$$

(the above can be calculated in parallel)

$$E = w_L w_S$$
$$F = -x_L w_S - w_L x_S$$
$$G = x_L x_S$$
$$\left[x_M \quad w_M\right] = \left[CF - BG \quad -BF + CE\right]$$