Low-Degree Polynomial Roots

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

Created: July 15, 1999
Last Modified: April 4, 2015

Contents

1 Introduction

Consider a polynomial of degree d of the form

$$p(y) = \sum_{i=0}^{d} p_i y^i \tag{1}$$

where the p_i are real numbers and where $p_d \neq 0$. A *root* of the polynomial is a number r, real or non-real (complex-valued with nonzero imaginary part) such that $p(r) = 0$. The polynomial can be factored as $p(y) = (y - r)^m f(y)$, where m is a positive integer and $f(r) \neq 0$. The power m is the *multiplicity* of the root. If $m = 1$, the root is said to be a *simple root*. If $m \geq 2$, the root is said to be a *multiple root*. The fundamental theorem of algebra states that $p(y)$ has exactly d roots, counting the multiplicities. If the K distinct roots are y_k for $0 \leq k < K$ and if m_k is the multiplicity of root y_k, then $d = \sum_{k=0}^{K-1} m_k$. We are assuming that the p_i are real numbers, so if a root is non-real, say, $r = u + \boldsymbol{\imath} v$, where $v \neq 0$, then $\bar{r} = u - \boldsymbol{\imath} v$ is also a root. The pair $(r, \bar{r})$ is referred to as a *complex conjugate pair of roots*.

The roots of polynomials of degrees 2, 3, or 4 can be written using closed-form algebraic expressions. The theory of polynomials shows us that there cannot be general closed-form algebraic expressions for the roots of $p(y)$ for $d \geq 5$. In practice, numerical root finders must be used to locate the roots. For degrees 2, 3, and 4, one may compute the roots using the closed-form expressions, but this approach is typically ill conditioned when using floating-point arithmetic. The main problem occurs in sign tests used to classify the roots, both in domain (real or non-real) and in multiplicity (determine m).

This document provides a robust approach for computing the roots using the closed-form expressions. The assumption is that the polynomial coefficients are floating-point numbers. In an implementation, they are converted to exact rational representations so that the root classification is performed with exact rational arithmetic and the result is theoretically correct. Naturally, the coefficients themselves might have been computed in a manner that introduced floating-point rounding errors. The discussion here does not take the coefficient errors into account, because only you the programmer have—and can take advantage of—that knowledge.

2 Discriminants

The first step in classification uses a concept called the *discriminant* of a polynomial $p(y)$ and is defined as

$$\Delta = p_d^{2d-2} \prod_{i<j} (r_i - r_j)^2 = (-1)^{d(d-1)/2} p_d^{2d-2} \prod_{i \neq j} (r_i - r_j) \tag{2}$$

where r_0 through r_{d-1} are the roots of $p(y)$ counting the multiplicities: a root r with multiplicity m occurs m times in the set $\{r_0, \ldots, r_{d-1}\}$. In the event a root exists with multiplicity 2 or larger, it is clear that $\Delta = 0$; thus, at its highest level the discriminant lets you know whether or not a polynomial has repeated roots. More information is actually known regarding the sign of Δ,

- $\Delta > 0$: for some integer k with $0 \leq k \leq d/4$, there are $2k$ distinct complex-conjugate pairs of roots and $d - 4k$ distinct real roots. Each root has a multiplicity of 1.
- $\Delta < 0$: for some integer k with $0 \leq k \leq (d-2)/4$, there are $2k + 1$ distinct complex-conjugate pairs of roots and $d - 4k - 2$ distinct real roots. Each root has a multiplicity of 1.

- $\Delta = 0$: at least one root r has multiplicity $m \geq 2$ and r may be either real or non-real. If r is non-real, then its conjugate $\bar{r}$ also has multiplicity m.

The prototypical case is $p(y) = p_0 + p_1 y + p_2 y^2$. The discriminant is $\Delta = p_1^2 - 4p_0p_2$. If $\Delta > 0$, the square root is a positive real number and there are two distinct real roots,

$$r_0 = \frac{-p_1 - \sqrt{\Delta}}{2p_2}, \quad r_1 = \frac{-p_1 + \sqrt{\Delta}}{2p_2}$$

The integer k in the sign tests is 0, so we have 2 distinct real roots and 0 distinct complex-conjugate pairs of roots. If $\Delta < 0$, the square root is a pure imaginary number and the roots are non-real,

$$r_0 = \frac{-p_1 - \boldsymbol{\imath}\sqrt{-\Delta}}{2p_2}, \quad r_1 = \frac{-p_1 + \boldsymbol{\imath}\sqrt{-\Delta}}{2p_2}$$

The integer k in the sign tests is 0. If $\Delta = 0$, we have a repeated real root,

$$r_0 = r_1 = \frac{-p_1}{2p_2}$$

so its multiplicity is $m = 2$.

Of course we usually do not know the roots of $p(y)$, so Equation (2) is not immediately helpful. However, the discriminant is related to another concept called the *resultant* of two polynomials, which is the product of the differences between the roots of the polynomials. The first derivative of $p(y)$ is $p'(y) = \sum_{i=0}^{d-1}(i+1)p_{i+1}y^i$ and the resultant of $p(y)$ and $p'(y)$ is denoted $R(p, p')$. It is the determinant of a $(2d-1) \times (2d-1)$ matrix called the *Sylvester matrix*,

$$R(p,p') = \det \begin{bmatrix} p_d & p_{d-1} & p_{d-2} & \cdots & p_1 & p_0 & 0\cdots & \cdots & 0 \\ 0 & p_d & p_{d-1} & p_{d-2} & \cdots & p_1 & p_0 & 0\cdots & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & \cdots & 0 & p_d & p_{d-1} & p_{d-2} & \cdots & p_1 & p_0 \\ dp_d & (d-1)p_{d-1} & (d-2)p_{d-2} & \cdots & p_1 & 0 & \cdots & \cdots & 0 \\ 0 & dp_d & (d-1)p_{d-1} & (d-2)p_{d-2} & \cdots & p_1 & 0 & \cdots & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & 0 & \cdots & 0 & dp_d & (d-1)p_{d-1} & (d-2)p_{d-2} & \cdots & p_1 \end{bmatrix} \tag{3}$$

The discriminant is then

$$\Delta = (-1)^{d(d-1)/2} R(p,p')/p_d \tag{4}$$

The discriminant for the degree-2 (quadratic) polynomial is

$$\begin{aligned} \Delta &= \tfrac{-1}{p_2} \det \begin{bmatrix} p_2 & p_1 & p_0 \\ 2p_2 & p_1 & 0 \\ 0 & 2p_2 & p_1 \end{bmatrix} \qquad (5) \\ &= p_1^2 - 4p_0p_2 \end{aligned}$$

The discriminant for the degree-3 (cubic) polynomial is

$$\begin{aligned}\Delta &= \tfrac{-1}{p_3}\det\begin{bmatrix} p_3 & p_2 & p_1 & p_0 & 0 \\ 0 & p_3 & p_2 & p_1 & p_0 \\ 3p_3 & 2p_2 & p_1 & 0 & 0 \\ 0 & 3p_3 & 2p_2 & p_1 & 0 \\ 0 & 0 & 3p_3 & 2p_2 & p_1 \end{bmatrix} \\ &= -27p_0^2p_3^2 + 18p_0p_1p_2p_3 - 4p_0p_2^3 - 4p_1^3p_3 + p_1^2p_2^2 \end{aligned} \tag{6}$$

The discriminant for the degree-4 (quartic) polynomial is

$$\begin{aligned}\Delta &= \tfrac{1}{p_4}\det\begin{bmatrix} p_4 & p_3 & p_2 & p_1 & p_0 & 0 & 0 \\ 0 & p_4 & p_3 & p_2 & p_1 & p_0 & 0 \\ 0 & 0 & p_4 & p_3 & p_2 & p_1 & p_0 \\ 4p_4 & 3p_3 & 2p_2 & p_1 & 0 & 0 & 0 \\ 0 & 4p_4 & 3p_3 & 2p_2 & p_1 & 0 & 0 \\ 0 & 0 & 4p_4 & 3p_3 & 2p_2 & p_1 & 0 \\ 0 & 0 & 0 & 4p_4 & 3p_3 & 2p_2 & p_1 \end{bmatrix} \\ &= 256p_0^3p_4^3 - 192p_0^2p_1p_3p_4^2 - 128p_0^2p_2^2p_4^2 + 144p_0^2p_2p_3^2p_4 - 27p_0^2p_3^4 + 144p_0p_1^2p_2p_4^2 - 6p_0p_1^2p_3^2p_4 \\ &\quad - 80p_0p_1p_2^2p_3p_4 + 18p_0p_1p_2p_3^3 + 16p_0p_2^4p_4 - 4p_0p_2^3p_3^2 - 27p_1^4p_4^2 + 18p_1^3p_2p_3p_4 - 4p_1^3p_3^3 \\ &\quad - 4p_1^2p_2^3p_4 + p_1^2p_2^2p_3^2 \end{aligned} \tag{7}$$

For larger d, the discriminant expressions become quite complicated. These occur in a simpler format for preprocessed polynomials.

3 Preprocessing the Polynomials

Preprocessing steps may be applied that make the root classification and construction simpler. The steps are error free when performed with exact rational arithmetic. In the analysis of $p(y)$, the polynomial is made *monic* by dividing through by its leading coefficient to obtain

$$q(y) = \sum_{i=0}^{d-1} q_i y^i + y^d \tag{8}$$

where $q_i = p_i/p_d$ for $0 \le i \le d-1$. The polynomials $p(y)$ and $q(y)$ have the same set of roots and multiplicities. The monic polynomial is made *depressed* by eliminating the y^{d-1} term using a simple translation, $y =$

$x - q_{d-1}/d$, to obtain

$$c(x) = \sum_{i=0}^{d-2} c_i x^i + x^d \tag{9}$$

The coefficients c_i are obtained from binomial expansions of powers of $(x - q_{d-1}/d)$. These will be listed for each low-degree polynomial discussed in later sections. It is possible that low-order coefficients of $c(x)$ are zero, so the analysis must allow for this. The roots of $c(x)$ of Equation (9) are computed and then inverse transformed to obtain roots to $p(y)$. Specifically, if $\hat{x}$ is a root of $c(x)$, the corresponding root of $p(y)$ is $\hat{y} = \hat{x} - q_{d-1}/d$.

An important observation about depressed polynomials helps us construct roots. The depressed polynomial $c(x)$ with roots r_0 through r_{d-1} (counting multiplicities) factors as

$$c(x) = \prod_{i=0}^{d-1} (x - r_i) \tag{10}$$

The coefficients c_i are related to sums of products of roots. These formulas are referred to as the *elementary symmetric polynomials* of the roots. In particular, $c_{d-1} = -\sum_{i=0}^{d-1} r_i$, which is the negation of the sum of the roots, and $c_0 = \prod_{i=0}^{d-1} r_i$, which is the product of the roots. The depressed polynomial has the constraint that $c_{d-1} = 0$, so the roots of the original polynomial $p(y)$ have been translated by the change of variables $y = x - q_{d-1}/d$ so that the sum of the roots of $c(x)$ is zero.

4 Quadratic Polynomials

The quadratic polynomial of interest is $p(y) = p_2 y^2 + p_1 y + p_0$, where $p_2 \neq 0$. The monic polynomial is $q(y) = y^2 + q_1 y + q_0$, where $q_1 = p_1/p_2$ and $q_0 = p_0/p_2$. The depressed polynomial is obtained by transforming $y = x - q_1/2$ to obtain $c(x) = x^2 + c_0$, where $c_0 = q_0 - q_1^2/4$. The discriminant for $c(x)$ is obtained from Equation (5) by substituting $p_0 = c_0$, $p_1 = 0$, and $p_2 = 1$,

$$\Delta = -4c_0 \tag{11}$$

If $\Delta > 0$, $c(x)$ has two simple real-valued roots, $r = \pm\sqrt{-c_0}$. If $\Delta < 0$, $c(x)$ has a complex-conjugate pair of roots, $r = \pm i\sqrt{c_0}$. If $\Delta = 0$, $c(x)$ has a real root $r = 0$ of multiplicity 2.

4.1 A Floating-Point Implementation

The straightforward code for computing the real-valued roots is shown next, where Real refers to a floating-point type such as float or double. The map stores pairs (r, m), where r is a real-valued root and m is its multiplicity. The precondition is that $p_2 \neq 0$.

```
void SolveQuadratic(Real p0, Real p1, Real p2, map<Real,int>& rmMap)
{
    Real q0 = p0 / p2;  // potential rounding error
    Real q1 = p1 / p2;  // potential rounding error
    Real q1half = q1 / 2;  // potential rounding error
    Real c0 = q0 - q1half * q1half;  // potential rounding error
    Real delta = -4 * c0;

    // Potential misclassification if theoretical delta is nearly zero.
```

```
    if (delta > 0)
    {
        // Two simple roots.
        Real r1 = sqrt(delta);  // usually inexact result
        Real r0 = -r1;
        rmMap.insert(r0 - q1half, 1);  // potential rounding error
        rmMap.insert(r1 - q1half, 1);  // potential rounding error
    }
    else if (delta == 0)
    {
        // One double root.
        Real r0 = 0;
        rmMap.insert(r0 - q1half, 2);
    }
    else  // delta < 0
    {
        // A complex conjugate pair of roots.
        // Complex z0 = -q1/2 - i*sqrt(-delta);
        // Complex z1 = -q1/2 + i*sqrt(-delta);
    }
}
```

The comments indicate the places where floating-point rounding errors can occur. As you can see, nearly every line of code has potential problems. The main problem is the misclassification of the roots when the sign of Δ is not computed exactly. Even when the quadratic formula is used directly for $p(y)$, where the discriminant is $\Delta = p_1^2 - 4p_0p_2$, you still have potential misclassification when Δ is nearly zero and the floating-point errors lead to a theoretically incorrect sign.

4.2 A Mixed-Type Implementation

For rational p_0, p_1, and p_2, we can use exact rational arithmetic to compute q_0, q_1, c_0, and Δ without errors. The classification of roots using Δ is therefore theoretically correct, so the following listing is more robust than the floating-point-only version. The code for computing the roots of the depressed quadratic is factored into a separate function, allowing it to be shared by the cubic root finder.

```
void SolveQuadratic(Real fp0, Real fp1, Real fp2, map<Real,int>& rmMap)
{
    Rational p0 = fp0, p1 = fp1, p2 = fp2;  // conversions to rationals, no error
    Rational q0 = p0 / p2;  // no error
    Rational q1 = p1 / p2;  // no error
    Rational q1half = q1 / 2;  // no error
    Rational c0 = q0 - q1half * q1half;  // no error

    map<Rational,int> rmLocalMap;
    SolveDepressedQuadratic(c0, rmLocalMap);

    for_each (rm in rmLocalMap)
    {
        // The conversion from Rational to Real has potential rounding errors.
        Rational root = rm.first - q1half;
        rmMap.insert((Real)root, rm.second);
    }
}

void SolveDepressedQuadratic(Rational c0, map<Rational,int>& rmMap)
{
    if (c0 < 0)  // delta > 0
    {
        // Two simple roots.
        Rational r1 = (Rational)sqrt(-(double)c0);  // potential rounding errors
        Rational r0 = -r1;
        rmMap.insert(r0, 1);
        rmMap.insert(r1, 1);
    }
```

```
        else if (c0 == 0)  // delta = 0
        {
            // One double root (r0 = 0).
            rmMap.insert(0, 2);
        }
        else  // delta < 0
        {
            // A complex conjugate pair of roots.
            // Complex z0 = -q1/2 - i*sqrt(c0);
            // Complex z1 = -q1/2 + i*sqrt(c0);
        }
    }
```

The root approximations can be improved by implementing an approximation to the sqrt function using only rational arithmetic and producing an output with enough precision so that the roots are correctly rounded for the precision supplied by Real. In this sense, SolveQuadratic would conform to the IEEE 754-2008 requirements of correct rounding that it imposes on the standard mathematics functions commonly supported on floating-point hardware.

5 Cubic Polynomials

The cubic polynomial of interest is $p(y) = p_3y^3 + p_2y^2 + p_1y + p_0$, where $p_3 \neq 0$. The monic polynomial is $q(y) = y^3 + q_2y^2 + q_1y + q_0$, where $q_2 = p_2/p_3$, $q_1 = p_1/p_3$ and $q_0 = p_0/p_3$. The depressed polynomial is obtained by transforming $y = x - q_2/3$ to obtain $c(x) = x^3 + c_1x + c_0$, where $c_0 = q_0 - q_1q_2/3 + 2q_2^3/27$ and $c_1 = q_1 - q_2^2/3$. The discriminant for $c(x)$ is obtained from Equation (6) by substituting $p_0 = c_0$, $p_1 = c_1$, $p_2 = 0$, and $p_3 = 1$,

$$\Delta = -(4c_1^3 + 27c_0^2) \tag{12}$$

According to the high-level sign tests for the discriminant, if $\Delta > 0$, there are three simple real roots. If $\Delta < 0$, there is a simple real root and a complex conjugate pair of roots. If $\Delta = 0$, there must be a root of multiplicity at least 2. Because non-real roots occur only as complex conjugate pairs, this forces either a single real root of multiplicity 3 or two distinct real roots, one with multiplicity 1 and one with multiplicity 2. A more detailed analysis is required in order to implement an algorithm.

5.1 Real Roots of Multiplicity Larger Than One

Real roots of multiplicity larger than 1 are easily constructed by examining the derivatives of $c(x)$. The first derivative is $c'(x) = 3x^2 + c_1$ and the second derivative is $c''(x) = 6x$.

A real root of multiplicity 3 must be a solution to $c(x) = 0$, $c'(x) = 0$, and $c''(x) = 0$. This forces $x = 0$, $c_1 = 0$, and $c_0 = 0$; that is, $c(x) = x^3$.

A real root of multiplicity 2 must be a solution $c(x) = 0$ and $c'(x) = 0$ subject to $c''(x) \neq 0$. The last condition implies $x \neq 0$. The equation $c'(x) = 0$ then requires $c_1 < 0$, and $x^2 = -c_1/3$. Substituting this into $c(x) = 0$, we obtain $0 = c(x) = 2c_1x/3 + c_0$ which implies a double root $r_0 = -3c_0/(2c_1)$ with $c_0 \neq 0$. Let the other root be named r_1. The zero root sum condition for the depressed polynomial is $2r_0 + r_1 = 0$, which implies $r_1 = -2r_0$.

5.2 One Simple Real Root

Consider the case of one simple real root and a complex-conjugate pair of roots. The real root can be constructed using the following observation. A real number may be written as the sum of two real numbers, so let the root be $r_0 = u + v$, where u and v are to be determined. Observe that $r_0^2 = u^2 + 2uv + v^2$ and

$$r_0^3 = u^3 + 3u^2v + 3uv^2 + v^3 = 3uv(u + v) + u^3 + v^3 = (3uv)r_0 + (u^3 + v^3) \tag{13}$$

Because r_0 is a root of $c(x)$,

$$0 = r_0^3 + c_1 r_0 + c_0 = (3uv + c_1)r_0 + (u^3 + v^3 + c_0) \tag{14}$$

It is sufficient to compute u and v for which $3uv = -c_1$ and $u^3 + v^3 = -c_0$. Cubing the first expression, we have $u^3v^3 = (-c_1/3)^3$. Define $\alpha = u^3$ and $\beta = v^3$; then, $\alpha\beta = (-c_1/3)^3$ and $\alpha + \beta = -c_0$. Solving the second equation for β and substituting in the first,

$$\alpha^2 + c_0\alpha + (-c_1/3)^3 = 0 \tag{15}$$

This is a quadratic equation with roots

$$\alpha = \frac{-c_0 - \sqrt{-\Delta/27}}{2}, \quad \beta = \frac{-c_0 + \sqrt{-\Delta/27}}{2} \tag{16}$$

where Δ is the discriminant defined by Equation (12). We know that $\Delta < 0$ for the case under consideration, so α and β are real and the (principal) cube roots are real. We conclude that

$$r_0 = u + v = \alpha^{1/3} + \beta^{1/3} = \left(\frac{-c_0 - \sqrt{-\Delta/27}}{2}\right)^{1/3} + \left(\frac{-c_0 + \sqrt{-\Delta/27}}{2}\right)^{1/3} \tag{17}$$

The non-real roots are obtained by factoring the polynomial knowing the simple real root,

$$x^3 + c_1 x + c_0 = (x - r_0)(x^2 + r_0 x + (r_0^2 + c_1)) = 0 \tag{18}$$

The quadratic factor has roots

$$\frac{-r_0 \pm \imath\sqrt{3r_0^2 + 4c_1}}{2} \tag{19}$$

where $3r_0^2 + 4c_1 > 0$.

To minimize operations, the case $c_0 \neq 0$ and $c_1 = 0$ can be handled separately. The real root is simply $r_0 = (-c_0)^{1/3}$, where the cube root is the real-valued one. The non-real roots are provided by Equation (19), $r_1 = r_0(-1 - \imath\sqrt{3})/2$ and $r_2 = r_0(-1 + \imath\sqrt{3})/2$.

5.3 Three Simple Real Roots

The construction in the previous section for a real root $r_0 = u + v$ still applies, but the values α and β of Equation (16) are now non-real,

$$\alpha = \frac{-c_0 - \imath\sqrt{\Delta/27}}{2}, \quad \beta = \frac{-c_0 + \imath\sqrt{\Delta/27}}{2} \tag{20}$$

where $\Delta > 0$. The polar representation in the complex plane of the second root is

$$\beta = \left(-c_0 + \imath\sqrt{\Delta/27}\right)/2 = \rho(\cos\theta + \imath\sin\theta) = \rho\exp(\imath\theta) \tag{21}$$

where ρ is the length of β as a 2-tuple in the complex plane and θ is the counterclockwise angle formed by β with the positive real axis of the complex plane. Specifically,

$$\rho = \sqrt{(-c_0/2)^2 + (\sqrt{\Delta/27}/2)^2}, \quad \theta = \mathsf{atan2}(\sqrt{\Delta/27}/2, -c_0/2) \tag{22}$$

There are three cube roots,

$$\beta^{1/3} = \rho^{1/3}\exp(\imath\theta/3), \quad \rho^{1/3}\exp(\imath(\theta/3 + 2\pi/3)), \quad \rho^{1/3}\exp(\imath(\theta/3 - 2\pi/3)) \tag{23}$$

The cube roots of α are the complex conjugates of the cube roots of β,

$$\alpha^{1/3} = \rho^{1/3}\exp(-\imath\theta/3), \quad \rho^{1/3}\exp(-\imath(\theta/3 + 2\pi/3)), \quad \rho^{1/3}\exp(-\imath(\theta/3 - 2\pi/3)) \tag{24}$$

The real-valued polynomial roots are the sums of the cube roots and their complex conjugates,

$$\begin{aligned}
r_0 &= \rho^{1/3}\exp(\imath\theta/3) + \rho^{1/3}\exp(-\imath\theta/3) \\
&= 2\rho^{1/3}\cos(\theta/3) \\
\\
r_1 &= \rho^{1/3}\exp(\imath(\theta/3 + 2\pi/3)) + \rho^{1/3}\exp(-\imath(\theta/3 + 2\pi/3)) \\
&= 2\rho^{1/3}\cos(\theta/3 + 2\pi/3) \\
&= \rho^{1/3}(-\cos(\theta/3) - \sqrt{3}\sin(\theta/3)) \\
\\
r_2 &= \rho^{1/3}\exp(\imath(\theta/3 - 2\pi/3)) + \rho^{1/3}\exp(-\imath(\theta/3 - 2\pi/3)) \\
&= 2\rho^{1/3}\cos(\theta/3 - 2\pi/3) \\
&= \rho^{1/3}(-\cos(\theta/3) + \sqrt{3}\sin(\theta/3))
\end{aligned} \tag{25}$$

As expected for the depressed cubic polynomial, $r_0 + r_1 + r_2 = 0$.

5.4 A Mixed-Type Implementation

The straightforward floating-point implementation suffers the same problem as for the quadratic polynomial. Floating-point rounding errors can cause a misclassification of the root domain or multiplicity. The implementation listed next uses exact rational arithmetic to provide a correct classification. The precondition is that $p_3 \neq 0$. The code for computing the roots of the depressed quadratic is factored into a separate function, allowing it to be shared by the quartic root finder.

```
void SolveCubic(Real fp0, Real fp1, Real fp2, Real fp3, map<Real, int>& rmMap)
{
    Rational p0 = fp0, p1 = fp1, p2 = fp2, p3 = fp3;
    Rational q0 = p0 / p3;
    Rational q1 = p1 / p3;
```

```
    Rational q2 = p2 / p3;
    Rational q2third = q2 / 3;
    Rational c0 = q0 - q2third * (q1 - 2 * q2third * q2third);
    Rational c1 = q1 - q2 * q2third;

    map<Rational, int> rmLocalMap;
    SolveDepressedCubic(c0, c1, rmLocalMap);

    for_each (rm in rmLocalMap)
    {
        Rational root = rm.first - q2third;
        rmMap.insert((Real)root, rm.second);
    }
}

void SolveDepressedCubic(Rational c0, Rational c1, map<Rational,int>& rmMap)
{
    // Handle the special case of c0 = 0, in which case the polynomial reduces
    // to a depressed quadratic.
    if (c0 == 0)
    {
        SolveDepressedQuadratic(c1, rmMap);
        map_iterator iter = rmMap.find(0);
        if (iter != rmMap.end())
        {
            // The quadratic has a root of zero, so increase its multiplicity.
            ++iter->second;
        }
        else
        {
            // The quadratic does not have a root of zero.  Insert one for the cubic.
            rmMap.insert(0, 1);
        }
        return;
    }

    // Handle the special case of c0 != 0 and c1 = 0.
    if (c1 == 0)
    {
        Rational r0;
        if (c0 > 0)
        {
            r0 = (Rational)-pow((double)c0, 1.0/3.0);
        }
        else
        {
            r0 = (Rational)pow(-(double)c0, 1.0/3.0);
        }
        rmMap.insert(std::make_pair(r0, 1));

        // One complex conjugate pair.
        // Complex z0 = r0*(-1 - i*sqrt(3))/2;
        // Complex z1 = r0*(-1 + i*sqrt(3))/2;
        return;
    }

    // At this time, c0 != 0 and c1 != 0.
    Rational delta = -(4 * c1 * c1 * c1 + 27 * c0 * c0);
    if (delta > 0)
    {
        // Three simple roots.
        Rational deltaDiv27 = delta / 27;
        Rational betaRe = -c0 / 2;
        Rational betaIm = sqrt((double)deltaDiv27) / 2;
        Rational theta = atan2((double)betaIm, (double)betaRe);
        Rational thetaDiv3 = theta / 3;
        double angle = (double)thetaDiv3;
        Rational cs = (Rational)cos(angle);
        Rational sn = (Rational)sin(angle);
        Rational rhoSqr = betaRe * betaRe + betaIm * betaIm;
        Rational rhoPowThird = (Rational)pow((double)rhoSqr, 1.0/6.0);
        Rational temp0 = rhoPowThird * cs;
```

```
            Rational temp1 = rhoPowThird * sn * (Rational)sqrt(3.0);
            Rational r0 = 2 * temp0;
            Rational r1 = -temp0 - temp1;
            Rational r2 = -temp0 + temp1;
            rmMap.insert(r0, 1);
            rmMap.insert(r1, 1);
            rmMap.insert(r2, 1);
        }
        else if (delta < 0)
        {
            // One simple root.
            Rational temp0 = -c0 / 2;
            Rational deltaDiv27 = delta / 27;
            Rational temp1 = (Rational)sqrt(-(double)deltaDiv27) / 2;
            Rational temp2 = temp0 - temp1;
            Rational temp3 = temp0 + temp1;
            if (temp2 >= 0)
            {
                temp2 = (Rational)pow((double)temp2, 1.0/3.0);
            }
            else
            {
                temp2 = (Rational)-pow(-(double)temp2, 1.0/3.0);
            }
            if (temp3 >= 0)
            {
                temp3 = (Rational)pow((double)temp3, 1.0/3.0);
            }
            else
            {
                temp3 = (Rational)-pow(-(double)temp3, 1.0/3.0);
            }
            Rational r0 = temp2 + temp3;
            rmMap.insert(r0, 1);

            // One complex conjugate pair.
            // Complex z0 = (-r0 - i*sqrt(3*r0*r0+4*c1))/2;
            // Complex z1 = (-r0 + i*sqrt(3*r0*r0+4*c1))/2;
        }
        else  // delta = 0
        {
            // One simple root and one double root.
            Rational r0 = -3 * c0 / (2 * c1);
            r1 = -2 * r0;
            rmMap.insert(r0, 2);
            rmMap.insert(r1, 1);
        }
    }
```

6 Quartic Polynomials

The quartic polynomial of interest is $p(y) = p_4y^4 + p_3y^3 + p_2y^2 + p_1y + p_0$, where $p_4 \neq 0$. The monic polynomial is $q(y) = y^4 + q_3y^3 + q_2y^2 + q_1y + q_0$, where $q_3 = p_3/p_4$, $q_2 = p_2/p_4$, $q_1 = p_1/p_4$, and $q_0 = p_0/p_4$. The depressed polynomial is obtained by transforming $y = x - q_3/4$ to obtain $c(x) = x^4 + c_2x^2 + c_1x + c_0$, where $c_0 = q_0 - q_3q_1/4 + q_3^2q_2/16 - 3q_3^4/256$, $c_1 = q_1 - q_3q_2/2 + q_3^3/8$, and $c_2 = q_2 - 3q_3^2/8$. The discriminant for $c(x)$ is obtained from Equation (7) by substituting $p_0 = c_0$, $p_1 = c_1$, $p_2 = c_2$, $p_3 = 0$, and $p_4 = 1$,

$$\Delta = 256c_0^3 - 128c_2^2c_0^2 + 144c_2c_1^2c_0 - 27c_1^4 + 16c_2^4c_0 - 4c_2^3c_1^2 \tag{26}$$

Some auxiliary quantities for classification are

$$a_0 = 12c_0 + c_2^2, \quad a_1 = 4c_0 - c_2^2 \tag{27}$$

The constructions of the roots in the next sections use derivatives of $c(x)$. The first derivative is $c'(x) = 4x^3 + 2c_2x + c_1$, the second derivative is $c''(x) = 12x^2 + 2c_2$, and the third derivative is $c'''(x) = 24x$. Keep in mind that any of the cases with a repeated root yield $\Delta = 0$.

The classification is shown in Table 1. The table is exhaustive, listing all possible combinations of root domain and multiplicity. A real root is named r_i. A non-real root is named z_i and its complex conjugate $\bar{z}_i$ is also a non-real root. The names are for distinct roots. A *multiplicity vector* is used to describe the multiplicities of the roots. A vector is of the form $(m_0, m_1, m_2, m_3) \in \{(4,0,0,0), (3,1,0,0), (2,2,0,0), (2,1,1,0), (1,1,1,1)\}$. The construction is presented in later sections, each section describing a particular multiplicity vector. A classification summary without details is found at the Wikipedia *quartic function* page.

Table 1. Classification of the roots for a depressed quartic polynomial.

multiplicity	factors	discriminant	conditions
$(4,0,0,0)$	$(x-r_0)^4$	$\Delta = 0$	$c_2 = 0 \wedge a_0 = 0 \wedge a_1 = 0$
$(3,1,0,0)$	$(x-r_0)^3(x-r_1)$	$\Delta = 0$	$c_2 < 0 \wedge a_0 = 0 \wedge a_1 < 0$
$(2,2,0,0)$	$(x-r_0)^2(x-r_1)^2$	$\Delta = 0$	$c_2 < 0 \wedge a_1 = 0$
$(2,2,0,0)$	$(x-z_0)^2(x-\bar{z}_0)^2$	$\Delta = 0$	$c_2 > 0 \wedge a_1 = 0$
$(2,1,1,0)$	$(x-r_0)^2(x-r_1)(x-r_2)$	$\Delta = 0$	$c_2 < 0 \wedge a_0 \neq 0 \wedge a_1 < 0$
$(2,1,1,0)$	$(x-r_0)^2(x-z_0)(x-\bar{z}_0)$	$\Delta = 0$	$a_1 > 0 \vee (c_2 > 0 \wedge (a_1 \neq 0 \vee c_1 \neq 0))$
$(1,1,1,1)$	$(x-r_0)(x-r_1)(x-r_2)(x-r_3)$	$\Delta > 0$	$c_2 < 0 \wedge a_1 < 0$
$(1,1,1,1)$	$(x-z_0)(x-\bar{z}_0)(x-z_1)(x-\bar{z}_1)$	$\Delta > 0$	$c_2 > 0 \vee a_1 > 0$
$(1,1,1,1)$	$(x-r_0)(x-r_1)(x-z_0)(x-\bar{z}_0)$	$\Delta < 0$	

I have verified that the conditions in that summary are correct and reproduce the results of the paper: *Graphical Discussion of the Roots of a Quartic Equation*, E. L. Rees, The American Mathematical Monthly, Vol. 29, No. 2, February 1922, pp. 51-55. This paper is based on analyzing the intersections of the auxiliary quartic polynomial $A(x) = x^4 + c_2x^2 + c_0$ and the linear polynomial $L(x) = -c_1x$. The intersections correspond to the roots of $c(x) = A(x) - L(x)$. The analysis of the conditions for $\Delta = 0$ are based on intersections at which the line of $L(x)$ is tangent to the graph of $A(x)$.

Although the Wikipedia page is interesting mathematically, it suffers the fate of many discussions about roots of low-degree polynomials. It is difficult to determine the logic necessary in a computer program to classify the roots and compute the real-valued ones. The conditions in Table 1 appear to have been generated using an optimization tool for Boolean expressions in order to generate a minimal set of tests, but it is not immediately clear in which order these conditions should be tested in a computer program. Also, the variables in some of the algebraic expressions for the roots can be real or non-real, and one must take care to manipulate these appropriately in order to generate only the real-valued roots. For example, the Cubic function page has a section entitled *Vieta's substitution*. This is an extremely concise formulation for the roots of a cubic polynomial. However, the w_k are cube roots that are usually non-real, so you cannot simply code the equations as they are stated when you want real roots. When expanded to allow a computer program to generate real roots, the details are as complicated as those for Cardano's method.

In the process of writing a quartic solver, I stumbled several times and had to revise the logic when test data

showed the code was not correct. One of those failures was based on the conditions for case $(x - r_0)^2(x - z_0)(x - \bar{z}_0)$ being incorrect, but the main contributor to the Wikipedia page was kind enough to fix that quickly. (The Rees' case of $\Delta = 0$, $q > 0$, $s > 0$, and $r \neq 0$ was missing in the pre-fix logic.)

Another failure occurred when using the approach of Ferrari to solve the depressed quartic by completing a square. The process requires choosing a root to a related cubic polynomial, and I recall reading (somewhere) that you can choose any root to the polynomial. I do not believe this is correct. The Wikipedia page, section *Ferrari's solution*, has a cubic equation (4) for generating a root y to complete the square. That process has a term $\beta/(2\sqrt{\alpha + 2y})$, and the author mentions the potential division by zero and "*... one thus need to choose a root of the cubic equation such that* $\alpha + 2y \neq 0$. *This is always possible unless for the depressed equation* $x^4 = 0$." It is not immediately clear that it is possible, but in fact it is—and it is possible to select the correct one without trying all real roots of the cubic. However, it is important to know the sign of $\alpha + 2y$, because when negative, $\sqrt{\alpha + 2y}$ is a non-real result. A computer program that generates the real roots must take this into account. My implementation shows that you can choose a root for which $\alpha + 2y > 0$ as long as $\beta \neq 0$.

The author mentions "*This implies* $\beta = 0$*, and thus that the depressed equation is bi-quadratic, and may be solved by an easier method.*" In fact, Ferrari's approach is symbolic and does not apply when $\beta = 0$; you have no option but to use the biquadratic solution. In turn, this means adding extra logic for the cases of Table 1 when $\Delta \neq 0$ to fall back to the biquadratic solution when $c_1 = 0$. I found that it is simpler (in a program) to handle the case $c_1 = 0$ separately before analyzing the sign of Δ.

That said, it is possible that $\alpha + 2y$ and β are both nearly zero, which can cause a problem numerically. My solution is to replace that term by one not involving a division; see the section on *Multiplicity Vector* $(1, 1, 1, 1)$ for details.

Yet another pitfall with Table 1 is that it hides information about the sign of c_0. I found that is is simpler (in a program) to handle the case $c_0 = 0$ separately, followed by the test for $c_1 = 0$. The if-then-else logic for the other cases was easier to implement.

6.1 Processing the Root Zero

If $c_0 = 0$, the depressed quartic is $c(x) = x(x^3 + c_2x + c_1)$, so we know $x = 0$ is a root. The other roots may be computed using the cubic root finder. That finder tests whether its constant term (c_1) is zero, and if so factors out x and calls the quadratic root finder. This recursive approach generates the appropriate root 0 and corresponding multiplicity.

6.2 The Biquadratic Case

If $c_0 \neq 0$ and $c_1 = 0$, the depressed quartic is $c(x) = x^4 + c_2x^2 + c_0 = (x^2 + c_2/2)^2 + a_1$, where $a_1 = c_0 - c_2^2/4$. The discriminant for the quartic is $\Delta = 16c_0(4c_0 - c_2^2)^2 = 256c_0a_1^2$.

Suppose that $\Delta < 0$; then $c_0 < 0$ and $a_1 < 0$ and we can solve for $x^2 = -c_2/2 \pm \sqrt{-a_1}$. The implication of $-c_2/2 - \sqrt{-a_1} \geq 0$ is that $c_2 < 0$ and $c_0 \geq 0$, but the latter condition is a contradiction to knowing $c_0 < 0$. Therefore, $-c_2/2 - \sqrt{-a_1} < 0$ and we have two non-real roots $z = \pm\imath\sqrt{c_2/2 + \sqrt{-a_1}}$. The conditions $c_0 < 0$ and $a_1 < 0$ guarantee that the graph of $c(x)$ intersects the x-axis in exactly two points, which then implies $-c_2/2 + \sqrt{-a_1} > 0$. We have real roots $r_1 = \sqrt{-c_2/2 + \sqrt{-a_1}}$ and $r_0 = -r_1$.

Suppose that $\Delta = 0$; then $a_1 = 0$. Because $c_0 \neq 0$, we additionally know that $c_2 \neq 0$. If $c_2 > 0$, $x^2 = -c_2/2$ has non-real solutions $z = \pm\imath\sqrt{c_2/2}$, each of multiplicity 2. If $c_2 < 0$, the solutions are real $r_1 = \sqrt{-c_2/2}$ and $r_0 = -r_1$.

Suppose that $\Delta > 0$; then $c_0 > 0$ and $a_1 \neq 0$. We have three cases to consider:

1. $c_2 \geq 0$: The graph of $c(x)$ does not intersect the x-axis. The implication of $-c_2/2 + \sqrt{-a_1} \geq 0$ is $c_0 \leq 0$, which contradicts our knowing $c_0 > 0$. Therefore, the two numbers on the right-hand side of $x^2 = -c_2/2 \pm \sqrt{-a_1}$ are negative and the non-real roots are $z_0 = \imath\sqrt{c_2/2 - \sqrt{-a_1}}$ (and $\bar{z}_0$) and $z_1 = \imath\sqrt{c_2/2 + \sqrt{-a_1}}$ (and $\bar{z}_1$).
2. $c_2 < 0$ and $a_1 > 0$: The graph of $c(x)$ does not intersect the x-axis. Solve $x^2 = -c_2/2 \pm \imath\sqrt{a_1}$; the two numbers on the right-hand side are non-real. We can compute the complex-valued square roots of these to produce the roots: $z_0 = c_0^{1/4}(\cos(\theta/2) + \imath\sin(\theta/2))$ (and $\bar{z}_0$) and $z_1 = c_0^{1/4}(-\cos(\theta/2) + \imath\sin(\theta/2))$ (and $\bar{z}_1$), where $\theta = \text{atan2}(\sqrt{a_1}, -c_2/2)$.
3. $c_2 < 0$ and $a_1 < 0$: The graph of $c(x)$ intersects the x-axis in four distinct points. Solve $x^2 = -c_2/2 \pm \sqrt{-a_1}$ where the two numbers on the right-hand side are positive. The four distinct real roots are $r_1 = \sqrt{-c_2/2 - \sqrt{-a_1}}$, $r_0 = -r_1$, $r_2 = \sqrt{-c_2/2 + \sqrt{-a_1}}$, and $r_3 = -r_2$.

6.3 Multiplicity Vector $(3, 1, 0, 0)$

A real root of multiplicity 3 must be a solution to $c(x) = 0$, $c'(x) = 0$, $c''(x) = 0$, but $c'''(x) \neq 0$. A solution r_0 must then be a root of $f(x) = 3c'(x) - xc''(x) = 4c_2x + 3c_1$; thus, $r_0 = -3c_1/(4c_2)$. The simple root r_1 is obtained from the zero root sum condition for the depressed polynomial, $3r_0 + r_1 = 0$, which implies $r_1 = -3r_0$.

6.4 Multiplicity Vector $(2, 2, 0, 0)$

A root of multiplicity 2 must be a solution to $c(x) = 0$, $c'(x) = 0$, but $c''(x) \neq 0$. A solution r_0 must then be a root of $f(x) = 4c(x) - xc'(x) = 2c_2x^2 + 3c_1x + 4c_0$, where $c_2 \neq 0$. If the two distinct roots are x_0 and x_1, then the zero root sum condition for the depressed polynomial is $2x_0 + 2x_1 = 0$. It must be that $x_1 = -x_0$. The only way the quadratic polynomial $f(x)$ has two roots, one the negative of the other, is when the linear term is zero; that is, $c_1 = 0$. The depressed polynomial is of the form $c(x) = x^4 + c_2x^2 + c_0 = (x^2 + c_2/2)^2 + a_1/4$. To obtain two roots, each of multiplicity 2 and with $x_1 = -x_0$, it is necessary that $a_1 = 0$. If $c_2 < 0$, the roots are real: $r_0 = -\sqrt{-c_2/2}$ and $r_1 = +\sqrt{-c_2/2}$. If $c_2 > 0$, then the roots are non-real: $z_0 = -\imath\sqrt{c_2/2}$ and $z_1 = +\imath\sqrt{c_2/2}$.

6.5 Multiplicity Vector $(2, 1, 1, 0)$

The root of multiplicity 2 is necessarily real; otherwise, a repeated non-real root forces the multiplicity vector to be $(2, 2, 0, 0)$, which is a case we already analyzed. The double real root r_0 must be a solution to $c(r_0) = 0$ and $c'(r_0) = 0$ but $c''(r_0) \neq 0$. The zero root sum condition for the depressed polynomial is $2r_0 + r_1 + r_2 = 0$. We also know that $c_1 \neq 0$; otherwise, $c(x) = x^4 + c_2x^2 + c_0$, which is the case of multiplicity vector $(2, 2, 0, 0)$ that was analyzed previously.

The common real root r_0 for $c(x)$ and $c'(x)$ must also be a root for the functions

$$\begin{aligned} f(x) &= 4c(x) - xc'(x) = 2c_2x^2 + 3c_1x + 4c_0 \\ g(x) &= c_2c'(x) - 2xf(x) = -6c_1x^2 + (2c_2^2 - 8c_0)x + c_1c_2 \\ h(x) &= 3c_1f(x) + c_2g(x) = (9c_1^2 - 8c_0c_2 + 2c_2^3)x + c_1(12c_0 + c_2^2) \end{aligned} \tag{28}$$

The double real root is therefore $r_0 = -c_1(12c_0 + c_2^2)/(9c_1^2 - 8c_0c_2 + 2c_2^3)$. To construct r_1 and r_2, factor out $(x - r_0)^2$ from $c(x)$ and solve the resulting quadratic equation,

$$c(x) = (x - r_0)^2(x^2 + \alpha x + \beta) = x^4 + (\alpha - 2r_0)x^3 + (\beta - 2r_0\alpha + r_0^2)x + (r_0^2\alpha - 2r_0\beta)x + (r_0^2\beta)$$

Choose $\alpha = 2r_0$ and $\beta = c_2 + 2r_0\alpha - r_0^2 = c_2 + 3r_0^2$. When the discriminant $\alpha^2 - 4\beta$ is positive, the roots to the quadratic are the real roots r_1 and r_2. When the discriminant is negative and the roots are nonreal z_0 and $\bar{z}_0$. The zero root sum condition for the depressed polynomial is $2r_0 + z_0 + \bar{z}_0 = 0$, which implies the non-real roots are of the form $z_0 = -r_0 + \imath v$, and $\bar{z}_0 = -r_0 - \imath v$. Solving the quadratic equation shows $v = \sqrt{c2 + r_0^2}$.

6.6 Multiplicity Vector $(1, 1, 1, 1)$

We can implement a quartic root finder so that the case $c_0 = 0$ is processed first, because a real root is zero and the reduced polynomial is cubic. The algorithm for cubic root finding may then be applied. We can also handle the case $c_0 \neq 0$ and $c_1 = 0$, because the quartic is then biquadratic, $x^4 + c_2x^2 + c_0$, and the quadratic root finding may be applies. In this section we therefore assume that $c_0 \neq 0$ and $c_1 \neq 0$.

Introduce a parameter t that will be determined later. Consider the expression

$$(x^2 + t)^2 = x^4 + 2tx + t^2 = -c_2x^2 - c_1x - c_0 + 2tx^2 + t^2 = (2t - c_2)x^2 - c_1x + (t^2 - c_0) \tag{29}$$

The right-hand side is a quadratic polynomial whose discriminant $\delta(t) = c_1^2 - 4(2t - c_2)(t^2 - c_0)$ is a cubic polynomial of t that has at least one real-valued root. Choose $\hat{t}$ to be the largest root; then the right-hand side of Equation (29) is the square of a quadratic polynomial,

$$(x^2 + \hat{t}\,)^2 = (\alpha x - \beta)^2$$

where $\alpha = \sqrt{2\hat{t} - c_2}$ and $\beta = c_1/(2\alpha) = \text{Sign}(c_1)\sqrt{\hat{t}^2 - c_0}$. The number α must be a positive real because $2\hat{t} - c_2 > 0$. To see this, observe that $\delta(c_2/2) = c_1^2 > 0$ and $\delta(t) \leq 0$ for $t \geq \hat{t}$, which means $c_2/2 < \hat{t}$. Although the equation for β is theoretically correct, numerical problems might occur when α is small, so use th other equation involving $\sqrt{\hat{t}^2 - c_0}$; this formulation suggests that if α is nearly zero, then so is c_1. We now have two quadratic equations

$$x^2 - \alpha x + (\hat{t} + \beta) = 0, \;\; x^2 + \alpha x + (\hat{t} - \beta) = 0$$

The discriminant of the first quadratic is $D_0 = \alpha^2 - 4(\hat{t} + \beta)$ and the discriminant of the second quadratic is $D_1 = \alpha^2 - 4(\hat{t} - \beta)$. Floating-point rounding errors when computing the discriminants that are theoretically nearly zero can cause misclassifications of the root domains and multiplicities. However, we can use the discriminant of the quartic and the auxiliary quantities to deduce the theoretically correct signs of D_0 and D_1 and handle the numerical computations accordingly.

When $\Delta > 0$, $c_2 < 0$, and $a_1 = 4c_0 - c_2^2 < 0$, we know that there are four simple real roots. In this case we know that $D_0 > 0$ and $D_1 > 0$. If rounding errors cause either quantity to be nonpositive, we can clamp them to a small positive number (or to zero to avoid having the programmer specify the small positive number). The roots are $(\alpha \pm \sqrt{D_0})/2$ and $(-\alpha \pm \sqrt{D_1})/2$.

When $\Delta > 0$, $c_2 > 0$, and $a_1 > 0$, we know that there are two complex conjugate pairs. In this case we know that $D_0 < 0$ and $D_1 < 0$. If an implementation wants only real roots, we simply do not report any, even when rounding errors cause either quantity to be nonnegative. However, if an implementation wants non-real roots to be returned, we can clamp D_0 and D_1 to a small negative number (or to zero to avoid having the programmer specify the small negative number). The roots are $(\alpha \pm \imath\sqrt{-D_0})/2$ and $(-\alpha \pm \imath\sqrt{-D_1})/2$.

When $\Delta < 0$, we know that there are two simple real roots and one complex conjugate pair. One of D_0 and D_1 is positive, the other negative. If both are theoretically nearly zero, floating-point rounding errors can prevent us from identifying the proper signs (so that we can clamp the results). However, we can infer the ordering of D_0 and D_1 from β, specifically from the sign of c_1. Observe that $D_1 - D_0 = 8\beta = 8c_1/\sqrt{2\hat{t} - c_2}$. If $c_1 > 0$, then $D_1 > D_0$, so theoretically $D_1 > 0$ and $D_0 < 0$. The roots are $(\alpha \pm \imath\sqrt{-D_0})/2$ and $(-\alpha \pm \sqrt{D_1})/2$. If $c_1 < 0$, then $D_1 < D_0$, so theoretically $D_1 < 0$ and $D_0 > 0$. The roots are $(\alpha \pm \sqrt{D_0})/2$ and $(-\alpha \pm \imath\sqrt{-D_1})/2$. We can clamp the numerical computations accordingly.

6.7 A Mixed-Type Implementation

The straightforward floating-point implementation suffers the same problem as for the quadratic and cubic polynomials. Floating-point rounding errors can cause a misclassification of the root domain or multiplicity. The implementation listed next uses exact rational arithmetic to provide a correct classification. The precondition is that $p_4 \neq 0$.

```
void SolveQuartic(Real fp0, Real fp1, Real fp2, Real fp3, Real fp4, map<Real, int>& rmMap)
{
    Rational p0 = fp0, p1 = fp1, p2 = fp2, p3 = fp3, p4 = fp4;
    Rational q0 = p0 / p4;
    Rational q1 = p1 / p4;
    Rational q2 = p2 / p4;
    Rational q3 = p3 / p4;
    Rational q3fourth = q3 / 4;
    Rational q3fourthSqr = q3fourth *  q3fourth;
    Rational c0 = q0 - q3fourth * (q1 - q3fourth * (q2 - q3fourthSqr * 3));
    Rational c1 = q1 - 2 * q3fourth * (q2 - 4 * q3fourthSqr);
    Rational c2 = q2 - 6 * q3fourthSqr;

    map<Rational, int> rmLocalMap;
    SolveDepressedQuartic(c0, c1, c2, rmLocalMap);

    for_each (rm in rmLocalMap)
    {
        Rational root = rm.first - q3fourth;
        rmMap.insert((Real)root, rm.second);
    }
}

void SolveDepressedQuartic(Rational c0, Rational c1, Rational c2, map<pair<Real,int>>& rmMap)
{
    // Handle the special case of c0 = 0, in which case the polynomial
    // reduces to a depressed cubic.
    if (c0 == 0)
    {
        SolveDepressedCubic(c1, c2, rmMap);
        map_iterator iter = rmMap.find(0);
        if (iter != rmMap.end())
        {
```

```
        // The cubic has a root of zero, so increase its multiplicity.
        ++iter->second;
    }
    else
    {
        // The cubic does not have a root of zero.  Insert one for the quartic.
        rmMap.insert(std::make_pair(mZero, 1));
    }
    return;
}

// Handle the special case of c1 = 0, in which case the quartic is a
// biquadratic x^4 + c1*x^2 + c0 = (x^2 + c2/2)^2 + (c0 - c2^2/4).
if (c1 == 0)
{
    SolveBiquadratic(c0, c2, rmMap);
    return;
}

// At this time, c0 != 0 and c1 != 0, which is a requirement for the
// general solver that must use a root of a special cubic polynomial.
Real c0sqr = c0 * c0, c1sqr = c1 * c1, c2sqr = c2 * c2;
Rational delta = c1sqr * (-27 * c1sqr + 4 * c2 * (36 * c0 - c2sqr))
    + 16 * c0 * (c2sqr * (c2sqr - 8 * c0) + 16 * c0sqr);
Rational a0 = 12 * c0 + c2sqr;
Rational a1 = 4 * c0 - c2sqr;

// Correct classification of the sign of delta.
if (delta > 0)
{
    if (c2 < 0 && a1 < 0)
    {
        // Four simple real roots.
        map<Real,int> rmCubicMap;
        SolveCubic(c1sqr - 4 * c0 * c2, 8 * c0, 4 * c2, -8, rmCubicMap);
        Rational t = (Rational)GetMaxRoot(rmCubicMap);
        Rational alphaSqr = 2 * t - c2;
        Rational alpha = (Rational)sqrt((double)alphaSqr);
        double sgnC1 = (c1 > 0 ? 1.0 : -1.0);
        Rational arg = t * t - c0;
        Rational beta = (Rational)(sgnC1 * sqrt(max((double)arg, 0.0)));
        Rational D0 = alphaSqr - 4 * (t + beta);
        Rational sqrtD0 = (Rational)sqrt(max((double)D0, 0.0));
        Rational D1 = alphaSqr - 4 * (t - beta);
        Rational sqrtD1 = (Rational)sqrt(max((double)D1, 0.0));
        Rational r0 = (+alpha - sqrtD0) / 2;
        Rational r1 = (+alpha + sqrtD0) / 2;
        Rational r2 = (-alpha - sqrtD1) / 2;
        Rational r3 = (-alpha + sqrtD1) / 2;
        rmMap.insert(r0, 1);
        rmMap.insert(r1, 1);
        rmMap.insert(r2, 1);
        rmMap.insert(r3, 1);
    }
    else // c2 >= 0 or a1 >= 0
    {
        // Two complex-conjugate pairs.  The values alpha, D0, and D1 are
        // those of the if-block.
        // Complex z0 = (alpha - i*sqrt(-D0))/2;
        // Complex z0conj = (alpha + i*sqrt(-D0))/2;
        // Complex z1 = (-alpha - i*sqrt(-D1))/2;
        // Complex z1conj = (-alpha + i*sqrt(-D1))/2;
    }
}
else if (delta < 0)
{
    // Two simple real roots, one complex-conjugate pair.
    map<Real,int> rmCubicMap;
    SolveCubic(c1sqr - 4 * c0 * c2, 8 * c0, 4 * c2, -8, rmCubicMap);
    Rational t = (Rational)GetMaxRoot(rmCubicMap);
    Rational alphaSqr = 2 * t - c2;
    Rational alpha = (Rational)sqrt((double)alphaSqr);
```

```
        double sgnC1 = (c1 > 0 ? 1.0 : -1.0);
        Rational arg = t * t - c0;
        Rational beta = (Rational)(sgnC1 * sqrt(max((double)arg, 0.0)));
        Rational r0, r1;
        if (sgnC1 > 0.0)
        {
            Rational D1 = alphaSqr - 4 * (t - beta);
            Rational sqrtD1 = (Rational)sqrt(max((double)D1, 0.0));
            r0 = (-alpha - sqrtD1) / 2;
            r1 = (-alpha + sqrtD1) / 2;

            // One complex conjugate pair.
            // Complex z0 = (alpha - i*sqrt(-D0))/2;
            // Complex z0conj = (alpha + i*sqrt(-D0))/2;
        }
        else
        {
            Rational D0 = alphaSqr - 4 * (t + beta);
            Rational sqrtD0 = (Rational)sqrt(max((double)D0, 0.0));
            r0 = (+alpha - sqrtD0) / 2;
            r1 = (+alpha + sqrtD0) / 2;

            // One complex conjugate pair.
            // Complex z0 = (-alpha - i*sqrt(-D1))/2;
            // Complex z0conj = (-alpha + i*sqrt(-D1))/2;
        }
        rmMap.insert(r0, 1);
        rmMap.insert(r1, 1);
    }
    else  // delta = 0
    {
        if (a1 > 0 || (c2 > 0 && (a1 != 0 || c1 != 0)))
        {
            // One double real root, one complex-conjugate pair.
            Rational r0 = -c1 * a0 / (9 * c1sqr - 2 * c2 * a1);
            rmMap.insert(r0, 2);

            // One complex conjugate pair.
            // Complex z0 = -root0 - i*sqrt(c2 + root0^2);
            // Complex z0conj = -root0 + i*sqrt(c2 + root0^2);
        }
    else
    {
        if (a0 != 0)
        {
            // One double real root, two simple real roots.
            Rational r0 = -c1 * a0 / (9 * c1sqr - 2 * c2 * a1);
            Rational alpha = 2 * r0;
            Rational beta = c2 + 3 * r0 * r0;
            Rational discr = alpha * alpha - 4 * beta;
            Rational temp1 = (Rational)sqrt((double)discr);
            Rational r1 = (-alpha - temp1) / 2;
            Rational r2 = (-alpha + temp1) / 2;
            rmMap.insert(r0, 2);
            rmMap.insert(r1, 1);
            rmMap.insert(r2, 1);
        }
        else
        {
            // One triple real root, one simple real root.
            Rational r0 = -3 * c1 / (4 * c2);
            Rational r1 = -3 * r0;
            rmMap.insert(r0, 3);
            rmMap.insert(r1, 1);
        }
    }
}

void SolveBiquadratic(Rational const& c0, Rational const& c2, map<Rational, int>& rmMap)
{
    // Solve 0 = x^4 + c2*x^2 + c0 = (x^2 + c2/2)^2 + a1, where
    // a1 = c0 - c2^2/2.  We know that c0 != 0 at the time of the function
```

```
// call, so x = 0 is not a root.  The condition c1 = 0 implies the quartic
// Delta = 256*c0*a1^2.

Rational c2Half = c2 / 2;
Rational a1 = c0 - c2Half * c2Half;
Rational delta = 256 * c0 * a1 * a1;
if (delta > 0)
{
    if (c2 < 0)
    {
        if (a1 < 0)
        {
            // Four simple roots.
            Rational temp0 = (Rational)sqrt(-(double)a1);
            Rational temp1 = -c2Half - temp0;
            Rational temp2 = -c2Half + temp0;
            Rational r1 = (Rational)sqrt((double)temp1);
            Rational r0 = -r1;
            Rational r2 = (Rational)sqrt((double)temp2);
            Rational r3 = -r2;
            rmMap.insert(r0, 1);
            rmMap.insert(r1, 1);
            rmMap.insert(r2, 1);
            rmMap.insert(r3, 1);
        }
        else  // a1 > 0
        {
            // Two simple complex conjugate pairs.
            // double thetaDiv2 = atan2(sqrt(a1), -c2/2) / 2.0;
            // double cs = cos(thetaDiv2), sn = sin(thetaDiv2);
            // double length = pow(c0, 0.25);
            // Complex z0 = length*(cs + i*sn);
            // Complex z0conj = length*(cs - i*sn);
            // Complex z1 = length*(-cs + i*sn);
            // Complex z1conj = length*(-cs - i*sn);
        }
    }
    else  // c2 >= 0
    {
        // Two simple complex conjugate pairs.
        // Complex z0 = -i*sqrt(c2/2 - sqrt(-a1));
        // Complex z0conj = +i*sqrt(c2/2 - sqrt(-a1));
        // Complex z1 = -i*sqrt(c2/2 + sqrt(-a1));
        // Complex z1conj = +i*sqrt(c2/2 + sqrt(-a1));
    }
}
else if (delta < 0)
{
    // Two simple real roots.
    Rational temp0 = (Rational)sqrt(-(double)a1);
    Rational temp1 = -c2Half + temp0;
    Rational r1 = (Rational)sqrt((double)temp1);
    Rational r0 = -r1;
    rmMap.insert(r0, 1);
    rmMap.insert(r1, 1);

    // One complex conjugate pair.
    // Complex z0 = -i*sqrt(c2/2 + sqrt(-a1));
    // Complex z0conj = +i*sqrt(c2/2 + sqrt(-a1));
}
else  // delta = 0
{
    if (c2 < 0)
    {
        // Two double real roots.
        Rational r1 = (Rational)sqrt(-(double)c2Half);
        Rational r0 = -r1;
        rmMap.insert(r0, 2);
        rmMap.insert(r1, 2);
    }
    else  // c2 > 0
    {
```

```
            // Two double complex conjugate pairs.
            // Complex z0 = -i*sqrt(c2/2);  // multiplicity 2
            // Complex z0conj = +i*sqrt(c2/2);  // multiplicity 2
        }
    }
}
```