How (Not) to Solve a Real Quartic

W. Kahan, Prof. Emeritus
Math., and E.E.&C.S.
Univ. of Calif. @ Berkeley

For the Scientific & Engineering Computation Seminar at U.C. Berkeley.

Abstract:

A program that computes all four zeros of a real quartic polynomial, and does so well enough to be included in a Math. library, has to overcome failure modes that afflict all the obvious candidates, like MATLAB's roots(C) and text-book formulas centuries old. Most of those failure modes are due to roundoff. How can they be found and, preferably, overcome without carrying extravagantly high precision?

This document is posted at www.eecs.berkeley.edu/~wkahan/Math128/5Mar14.pdf .

What use are all the zeros of a *Monic Real Quartic*

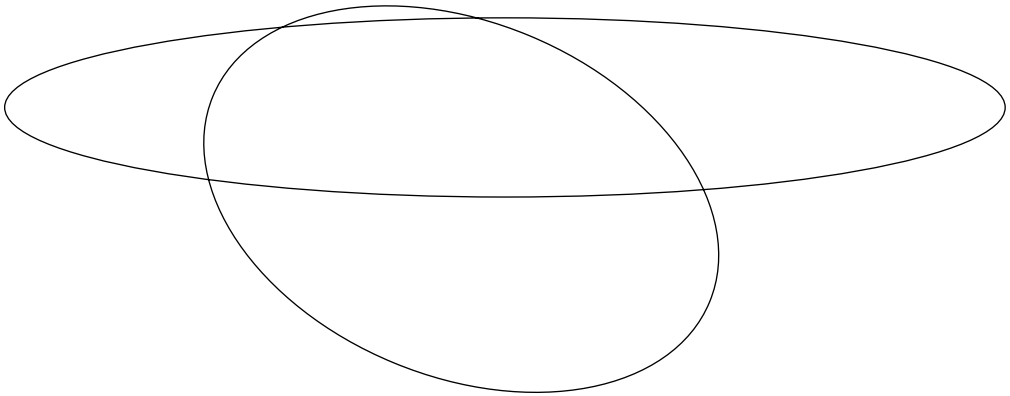
$$Q(x) := x^4 + a\cdot x^3 + b\cdot x^2 + c\cdot x + d$$

?

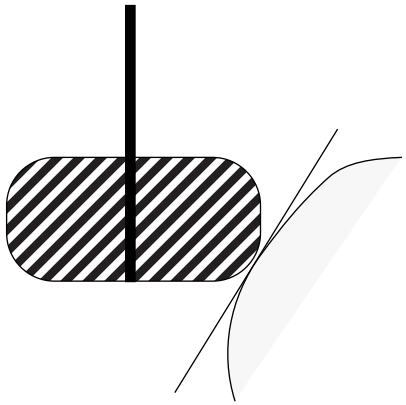
An Application:

They are needed to locate the intersections, if any, of uncentered conics like

Ellipses, Hyperbolas, Parabolas .



Another Application: to a Computer-Controlled Milling Machine:



One of the machine's cutting heads cuts a toroidal excavation. The machine must lower the head to barely touch a support-plane of the work to be produced from a metal blank. To position the head requires the solution of a quartic equation. According to p. 2 of

http://en.wikipedia.org/wiki/Quartic_function as of 12 June 2014, ...

"Over 10% of the computational time in a CAM system can be consumed simply calculating the solution to millions of quartic equations."

Other uses occur rarely so far as I know.

What must a floating-point program do to merit inclusion in a Math. library?

- Run at least about as fast as possible.
- Deliver at least about as much accuracy as the input data deserve, subject to the limitations of the arithmetic available on the computing platform used.

How much accuracy do the zeros of a quartic polynomial deserve?

It depends upon how given data determines the polynomial. If given are floating-point values of the coefficients a, b, c, d of $Q(x) := x^4 + a\cdot x^3 + b\cdot x^2 + c\cdot x + d$, …

- Simple well-isolated zeros should lose only a few sig. bits to roundoff.
- Nearly double zeros may lose roughly half of the arithmetic's sig. bits.
- Nearly triple zeros may lose roughly two thirds of the arithmetic's sig. bits.
- Nearly quadruple zeros may lose roughly three quarters of the arithmetic's sig. bits.

" Nearly multiple" means that a tiny perturbation to the coefficients of Q suffices to increase some zeros' multiplicities. These zeros of polynomials of higher degree need not look very close together; for a striking example easy to analyze see my web page's www.eecs.berkeley.edu/~wkahan/Math128/Poly.pdf , pp. 6-7.

The hypersensitivity of a polynomial's zeros to perturbations is a consequence of its representation as a weighted sum of monomials:

$$Q(x) := x^4 + a\cdot x^3 + b\cdot x^2 + c\cdot x + d$$

The zeros of a polynomial generated differently may suffer far less from perturbations.

Example: $Q(x) := \mathrm{Det}(x\cdot\mathbf{I} - \mathbf{A})$ for a real matrix $\mathbf{A}$ may lose only a few sig. bits to perturbations of the elements of $\mathbf{A}$, especially if $\mathbf{A}$ remains symmetric. See my …/MathH110/NormOvrv.pdf, Part V, about eigenvalues of $\mathbf{A}$.

Example: The zeros of Orthogonal Polynomials generated by a three-term recurrence are well determined by the coefficients of the recurrence, not so well by the monomials' coefficients. See pp. 15-16 of my …/MathH110/HilbMats.pdf. This phenomenon plays a crucial rôle in the computation of nodes τ_j and weights w_j for Gaussian Quadrature formulas $\sum_j w_j\cdot f(\tau_j) \approx \int_\alpha^\beta w(\tau)\cdot f(\tau)\cdot d\tau$.

What follows concerns the accuracies of zeros when given the monomials' coefficients in Q(x) above. We wish not to lose more sig. bits than are warranted by the zeros' near multiplicities. Losses will be negligible in arithmetic carrying *extravagant precision*, say four to five times as many sig. bits as are trusted in the data and desired in results.

A Less Stringent Requirement for Accuracy:
"Backward" Error-Analysis

Each computed zero Z of $Q(x) := x^4 + a\cdot x^3 + b\cdot x^2 + c\cdot x + d$ should, ideally, satisfy a

Nearby Equation $Z^4 + A\cdot Z^3 + B\cdot Z^2 + C\cdot Z + D = 0$ ***exactly***, wherein

hypothetical (not computed) array $[A, B, C, D]$ is close enough to $[a, b, c, d]$ that

$\| [(A-a)\cdot Z^3,\ (B-b)\cdot Z^2,\ (C-c)\cdot Z,\ D-d] \|$ doesn't much exceed roundoff in a computable

$\| [a\cdot Z^3,\ b\cdot Z^2,\ c\cdot Z,\ d] \|$. (Any familiar vector norm $\|\ldots\|$ will do.)

Questions for Research:

- Which numerical root-finders satisfy this less stringent requirement? (Not many)
- Can the same array $[A, B, C, D]$ serve for every computed zero? (Probably)
- Can that array be constructed from all four computed zeros of Q ? For example, MATLAB's poly([Z1, Z2, Z3, Z4]) offers a candidate $[1, A, B, C, D]$. But, is poly([…]) accurate enough for all polynomials of large degree?

This less stringent requirement can be violated by MATLAB's roots([1, a, b, c, d]) …
… despite published error-analyses.

MATLAB's roots([1, a, b, c, d]) computes all zeros of $Q(x) := x^4 + a\cdot x^3 + b\cdot x^2 + c\cdot x + d$.

Try roots([1, -1, $-S^2$, S^2, -1]) for any big $S \geq 10^9$;

get alleged "roots" $\approx$ [-S, **0**, 1, S]' instead of $\approx$ [-S, $1/S^2$, 1, S]' , as if the input "d := -1" had been degraded to "d := 0" .

These input coefficients determine all 4 zeros within relative errors roughly like $1/S^2$ despite the zeros' widely disparate magnitudes. MATLAB has crushed the tiniest zero.

Why?

The computed zeros have been obtained from the eigenvalues of a ***Companion*** Matrix:

$$\mathbf{C} := \begin{bmatrix} -a & -b & -c & -d \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ; \qquad \mathrm{Det}(x\cdot\mathbf{I} - \mathbf{C}) = Q(x) .$$

But MATALB's eig(**C**) crushed its tiniest eigenvalue because a stopping criterion used by eig's QR-iteration deemed that tiny eigenvalue negligible compared with $\|\mathbf{C}\|$.

There is an alternative to **C** , a different kind of Companion Matrix …

See Miroslav Fiedler (2003) "A note on companion matrices" pp. 325-331 of *Lin.Algebra & its Applications* **372**.

In 2002 Miroslav Fiedler invented a different kind of *Companion* Matrix:

$$\mathbf{F} := \begin{bmatrix} -a & -b & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -c & 0 & -d \\ 0 & 1 & 0 & 0 \end{bmatrix} ; \qquad \mathrm{DET}(x\cdot\mathbf{I} - \mathbf{F}) = Q(x) := x^4 + a\cdot x^3 + b\cdot x^2 + c\cdot x + d\ .$$

In MATLAB's roots.m replace **C** by **F** when its dimension is even (by **F.'** when its dimension is odd) to get a revised program we shall call "ro0ts.m". This is about as accurate as roots.m, and more accurate when **F** has one extremely tiny eigenvalue:

$$\mathrm{ro0ts}([1,\ -1,\ -S^2,\ S^2,\ -1]) \approx [-S,\ 1/S^2,\ 1,\ S]'\ .$$

Alas, neither roots.m nor ro0ts.m copes well with ***two*** extremely tiny zeros:

$$\mathrm{roots}([1,\ 0,\ -S^2,\ 0,\ 1]) \approx \mathrm{ro0ts}([1,\ 0,\ -S^2,\ 0,\ 1]) \approx [-S,\ \mathbf{0},\ \mathbf{0},\ S]' \quad \text{if } S \geq 10^{12}$$

instead of the correct $\approx [-S,\ -1/S,\ 1/S,\ S]'$.

Research Project: When eig(**F**) is invoked in roots or ro0ts, can its stopping criteria be altered to achieve the desired accuracy for tiny zeros without much degrading speed for the other zeros?

The **Classical Algebraic Method** finds **Quadratic Factors**, each of the form

$$F(x,\ \alpha,\ \beta) := x^2 + \alpha\cdot x + \beta \quad \text{(for different coefficients } \alpha \text{ and } \beta\text{), of …}$$

$$Q(x) := x^4 + a\cdot x^3 + b\cdot x^2 + c\cdot x + d \equiv F(x,\ a/2+\Psi,\ \beta)\cdot F(x,\ a/2-\Psi,\ \tau)$$

in which β and τ are the larger and lesser roots z respectively of $F(z,\ \Phi,\ d) = 0$, so the two unknowns Ψ and Φ must satisfy $\Psi := \sqrt{(\Phi - b + a^2/4)}\cdot\mathrm{sign}(a\cdot\Phi - 2c)$ and

$$P(\Phi) := \Phi^3 - b\cdot\Phi^2 + (a\cdot c - 4d)\cdot\Phi + d\cdot(4b - a^2) - c^2 = 0\ .$$

This $P(x)$ is called a "Resolvent Cubic", and its largest (rightmost) real zero Φ is always at least as big as $b - a^2/4$ (and $2\sqrt{d}$ too if $d \ge 0$), whence we can deduce that Ψ and β and τ are real. In short, we can always find two ***real*** factors F of Q.

What can go wrong?

There are formulas for the zeros of a cubic, but if all are real they require complex cube roots involving trigonometric functions. See my web page's …/Math128/Cubic.pdf , which offers an *iterative* method that loses less to roundoff than do those formulas.

If Q has a multiple zero, so has P, which can worsen roundoff's injury to accuracy; then the factors' product can differ from Q by more than a few end-bits of coefficients.

Digression: Little-known relations between the Quartic

$$Q(z) := z^4 + a\cdot z^3 + b\cdot z^2 + c\cdot z + d$$

and its Resolvent Cubic

$$P(\zeta) := \zeta^3 - b\cdot\zeta^2 + (a\cdot c - 4d)\cdot\zeta + d\cdot(4b - a^2) - c^2 \ .$$

- If the zeros of Q are $\{ v, w, x, y \}$, then
  the zeros of P are $\{ v\cdot w+x\cdot y,\ v\cdot x+w\cdot y,\ v\cdot y+w\cdot x \}$.

Consequently Q and P have the same ***Discriminant***

$$\left((v-w)\cdot(w-x)\cdot(x-y)\cdot(y-v)\cdot(v-x)\cdot(y-w) \right)^2 .$$

This means that …

- Q has multiple zeros $\leftrightarrow$ P has multiple zeros.
- Q has 4 distinct zeros, all real or all complex $\leftrightarrow$ P has 3 distinct real zeros.
- Q has 4 distinct zeros, 2 real, 2 complex $\leftrightarrow$ P has 3 distinct zeros, 1 real.
- Q has 2 zeros far tinier than the others $\leftrightarrow$ P has 2 zeros far tinier than the other.

So, exclusively real arithmetic, without trig. functions, suffices to compute the one needed zero of P and all four zeros of Q, real or complex. However, coincidences between multiplicities can lose accuracy excessively among computed zeros of Q.

Iterative Refinement of Real Quadratic Factors of a Quartic:

Suppose two Quadratics' coefficients $\alpha, \beta, \sigma, \tau$ come close to satisfying

$$F(x, \alpha, \beta)\cdot F(x, \sigma, \tau) \equiv Q(x) .$$

By matching coefficients of powers of x we obtain four slightly nonlinear equations in the four unknowns $\alpha, \beta, \sigma, \tau$. The 4-by-4 Jacobian matrix of first partial derivatives has a simple form making Newton's iteration an attractive way to solve the equations.

The determinant of that Jacobian matrix vanishes just when $F(x, \alpha, \beta)$ and $F(x, \sigma, \tau)$ have a zero in common. This coincidence can be averted by making the zero in common instead a double zero of only one of the factors, unless Q has worse than a double zero.

Research Project: What to do, when a too-near-vanishing Jacobian determinant cannot be averted, has not been figured out yet.

Iterative refinement of zeros, either singly (Newton's or Laguerre's iteration) or all together (Weierstrass' iteration), is not recommended unless a way is found to prevent some of the approximate zeros in a cluster from becoming refined onto the same true zero in a way that loses track of a different true zero. And it needs complex arithmetic.

Iterative Methods that pick off the zeros one at a time work best:

Better than Newton's or Halley's iterations is ***Laguerre**'s iteration*, which replaces x by the nearest $x + \Delta x$ satisfying

$$\big((3-m)\cdot Q'(x)^2 - (4-m)\cdot Q(x)\cdot Q''(x)\big)\cdot\Delta x^2 \;-\; 2m\cdot Q(x)\cdot Q'(x)\cdot\Delta x \;-\; 4m\cdot Q(x)^2 = 0$$

to head towards a zero of estimated multiplicity m. ($m := 1$ may go slow.) To assure convergence, the formula for Δx must be inhibited by ***D'Alembert**'s principle*:

A minimum of the magnitude of the quartic $Q(x)$
(or of any other analytic function of a complex variable)
can occur only at a zero.

Thus, Δx should be ***inhibited*** (shortened) by the restriction $|Q(x + \Delta x)| < |Q(x)|$.

When an approximate zero Z is found, it must be removed from Q by ***Deflation***:
Replace $Q(x)$ by a polynomial $(Q(x) - Q(Z)\cdot(x/Z)^k)/(x - Z)$ in x with k chosen to Maximize $|(\text{Coeff. of } x^k \text{ in } Q)\cdot Z^k|$ to Minimize the relative perturbation of Q. This amounts to running Horner's Recurrence from both ends to meet at the k^{th} coeff.

See Brian T. Smith's ZERPOL on the IBM 7094, IMSL's version, HP-71B version.
Research Project: Software-engineer a better version not hampered by copyrights.

Tests

Ideally, tests should be unnecessary for a program that has been proved mathematically to deliver satisfactory results for all admissible input data. However …

- Many a valuable numerical program works better than has been (or can be?) proved.
- A "proof" can be wrong. Incorrect proofs have been published. Because the proof's length tends to far exceed the proved program's length, the proof is more vulnerable to error, especially in definitions of "satisfactory" and "admissible". See p. 5 .

Test data generated randomly is necessary but generally incapable of exposing ***all*** of a program's failure modes. For example, the Intel Pentium's Divide Bug of 1994 was not exposed in advance by any of several billion pairs of test operands. See too my web page's …/FailMode.pdf. Designing efficient test data faces difficult challenges; good tests explore ***all*** significant possibilities but not too redundantly lest tests run too long.

Efficient tests cluster about ***all*** of a program's ***Singularities***. These can be hard to find. Not all are identifiable with mathematical pathologies of the given problem, here zeros of Q of high multiplicity. For instance, p. 8's largest real zero Φ of P can jump discontinuously at certain admissible data; how soon can you locate such data?