Accuracy Tests for Polynomials' Zero-Finders

W. Kahan, Prof. Emeritus
Math. Dept., and E.E. & Computer Sci. Dept.
University of California
Berkeley CA 94720-1776

§0: Abstract

Test data is supplied to help assess the accuracies of zero-finders for real polynomials of degrees 2 to 6 . The data consist of polynomials, with integer coefficients constructed from Fibonacci numbers F(n) , and formulas for computing all their zeros fairly accurately. A computerized algebra system can verify all the many formulas. As n increases, the zeros cluster together more tightly, thus becoming harder for numerical zero-finders to compute accurately; and their computed zeros can be surprisingly inaccurate.

This document is posted on my web page at www.eecs.berkeley.edu/~wkahan/Math128/Fibs_2_6.pdf .

§1: Introduction

Assessments of a numerical program's accuracy can be time-consuming and tricky. Time consuming because accidents of roundoff can produce misleading assessments from too few tests, especially if they are randomly generated samples that miss difficult cases. Tricky because under-appreciated rounding errors in the tests' inputs can interfere with the accuracies of the expected outputs, thus undermining the assessments. The test data supplied hereunder consist of amply many polynomials, parameterized by a positive integer n , whose zeros get harder to compute accurately as n increases. The polynomials' coefficients, all floating-point integers constructed from Fibonacci numbers F(n) , are verifiably computed exactly until n gets too big, depending upon the arithmetic's precision. Formulas are supplied from which the polynomials' true zeros, real and complex, can be computed within a unit or two in the last sig. digit carried by the arithmetic.

These formulas' derivations are too tedious to reproduce here. Instead the formulas are devised to be confirmed by a computerized algebra system like *Maple*®, *Mathematica*® or *Derive*®. For instance, the Fibonacci numbers F(n) are best computed numerically from a *Recurrence*

$$F(n+1) := F(n) + F(n-1) \quad \text{starting from} \quad F(0) := 0 \text{ and } F(\pm 1) := \pm 1 ;$$

but a computerized algebra system might turn this definition into a *Recursion* from which F(n) cannot be represented economically for a symbolic rather than numerical value of n during the confirmations of the formulas for the zeros. Instead, the prior symbolic definitions

$$\tau := (1 + \sqrt{5})/2 ; \quad F(n) := (\tau^{2n} - (-1)^n)/(\sqrt{5}\cdot\tau^n) ;$$

let the algebra system confirm easily the recurrence from which will be computed an array of as many floating-point values F(n) as can be computed exactly without rounding errors. The array ends at the first n for which the expression $(F(n+1) - F(n)) - F(n-1)$ becomes nonzero. This happens at n = 36 carrying 24 sig.bits, n = 49 for 10 sig.dec., n = 73 for 15 sig.dec., n = 78 for 53 sig.bits, n = 93 for 64 sig.bits, and n = 165 for 113 sig.bits correctly rounded.

The test data supplied hereunder do not exhaust the tests of zero-finders; conscientious tests will include much more. Tests for accuracy will include a larger range of problems some of them chosen with wide-ranging coefficients to kindle premature over/underflows. For instance, the zeros x of polynomial $P(n, x/\lambda)$ and those of $P(n, x)$ multiplied by λ should match closely, exactly if $\lambda = \pm 2^k$ in binary arithmetic unless integer $|k|$ is too big. Speed and ease of use need tests too, including the handling of special cases of coefficients that are zero or infinite or *NaN* (*N*ot *A N*umber). Some applications demand greater fidelity from a zero-finder; it may, for instance, be expected not to miscompute neighboring real zeros as slightly complex nor *vice versa*. A miscomputation like that would be exposed by our formulas. These do expose larger errors than might reasonably be expected in §11 below; but they cannot expose inaccuracies some zero-finders suffer when a polynomial's zeros divagate very widely in magnitude.

§2: Stretching Small Integers

Roundoff is accidental, ragged but not random. This is why a realistic assessment of roundoff's impact upon a program can require that its accuracy be sampled at numerous sets of input data. Among these, small integers are best avoided because they often incur atypical roundings.

For instance, if the two assignments

$$y := w/x ; \quad z := x \cdot y ;$$

are rounded in accordance with IEEE Standard 754 (1985) for Binary Floating-Point Arithmetic, we would expect the relation "$z = w$" to be violated at least occasionally by their two rounding errors. In fact, when w and x are independent random floating-point numbers then $z \neq w$ about 11% of the time. But for *all* small integers $|w| \leq 8000000$, say, and for *every* integer x drawn from {1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 16, 17, 18, 20, … } we *always* get $z = w$. This cancellation of rounding error is explained in "A Computation with Almost No Significance" posted on my web page at www.eecs.berkeley.edu/~wkahan/CS279/Div754.pdf .

A zero-finder may subject a polynomial $\sum_j a_j x^{n-j}$ with small integer coefficients a_j to exact *Preconditioning* transformations that enhance the accuracy of computed zeros in tight clusters. An example is in §7 of "To Solve a Real Cubic Equation" posted at .../Math128/Cubic.pdf. To test such a zero-finder's accuracy without preconditioning, those small integer coefficients a_j must first be *Stretched* by multiplications by an integer M that are performed exactly to turn each a_j into a sufficiently bigger integer $M \cdot a_j$ without changing the polynomial's zeros. The scheme on p. 13 of "On the Cost of Floating-Point Computation Without Extra-Precise Arithmetic" posted at .../Qdrtcs.pdf gets a suitable multiplier $M := \text{floor}(R/\max_j |a_j|)$ wherein R is a random integer between 2^{p-1} and $2^p - 1$ for binary floating-point with p sig.bits. If $M = 0$ the coefficients a_j cannot all be deemed "small integers".

The coefficients constructed from Fibonacci numbers $F(n)$ hereunder will, for small integers n, be integers small enough that they should be stretched to generate (in)accuracies more nearly typical of any zero-finder under test.

§3: Avoiding Roundoff

Some polynomial zero-finders, deemed *Backward Stable*, can compute a given polynomial's zeros badly and yet almost as accurately as if each were a zero exactly of a polynomial whose coefficients differ by at most one rounding error from each given coefficient. This is why a fair assessment of a zero-finder's accuracy begins with coefficients stored unblemished by roundoff.

The coefficients constructed hereunder from Fibonacci numbers
F(n) will be unblemished by roundoff unless integer n is too big.
How big is "too big" ?

The simplest way to decide whether any chosen n is too big uses the INEXACT flag mandated by IEEE Standard 754 and provided by conforming hardware. This flag is raised just when a rounding error or over/underflow produces an arithmetic result different from what would have been produced if the arithmetic's precision and range were unlimited. To use this flag, …

Choose n ;
Put the INEXACT flag down ;
Compute all the desired polynomial's coefficients ;
Test the INEXACT flag ;

If the flag is still down, all the computed coefficients are exact — unblemished by roundoff. If the flag has been raised, rounding errors have contaminated some coefficient(s) unless they all have cancelled like the errors described earlier in §2. However, for none of the four binary arithmetics mentioned in §1 will rounding errors cancel if any occur during the calculations of our coefficients, in which case the raised flag will indicate correctly that n is too big,

Alas, most programming languages deny programmers access to the hardware's flags. In these benighted languages a programmer must resort to tricks to discover whether a rounding error has occurred. For correctly rounded arithmetic conforming to IEEE Standard 754 (1985) for *Binary* Floating-Point, all tricks that will be necessary are listed below. In each trick w is the result of operating upon floating-point numbers f and g, and $z \neq 0$ just when w is blemished.

When $|f| \geq |g|$, $w := f + g$; $z := (f - w) + g$; (DON'T IGNORE PARENTHESES!)
$w := 3.0 \cdot f$; $z := (w - 2.0 \cdot f) - f$;
$w := 6.0 \cdot f$; $z := (w - 4.0 \cdot f) - 2.0 \cdot f$;
$w := 15.0 \cdot f$; $z := (w - 16.0 \cdot f) + f$;

If $z \neq 0$ the chosen n is too big; a computed coefficient is unavoidably blemished by roundoff.

§4: Attenuating Roundoff

When n is big, but not too big, every constructed polynomial's zeros $Z_J(n)$ will cluster tightly enough that most numerical zero-finders will compute them too inaccurately to determine their differences $Z_J(n) - Z_K(n)$ reliably. An adequate assessment of the inaccuracies of the zero-finder's computed zeros $z_j(n)$ requires errors $Z_J(n) - z_j(n)$ to be computed as accurately as differences $Z_J(n) - Z_K(n)$. This task entails two sub-tasks:

•1: Which true zero $Z_J(n)$ is rightly approximated by a zero-finder's computed zero $z_j(n)$?

•2: How can rounding errors in $Z_J(n) - z_j(n)$ be kept tinier than differences $Z_J(n) - Z_K(n)$?

Task •1 amounts to finding a permutation π of $\{1, 2, \ldots, m\}$ to match pairs drawn from

$$\{ z_{\pi(1)}, z_{\pi(2)}, z_{\pi(3)}, \ldots z_{\pi(m)} \} \quad \text{and} \quad \{ Z_1, Z_2, Z_3, \ldots, Z_m \}$$

in a way that minimizes $\max_J |Z_J - z_{\pi(J)}|$. Here m is the polynomial's degree. The number of complex (non real) zeros in $\{Z_J\}$ may differ from that number in $\{z_j\}$, but they should all come in complex conjugate pairs. More than one permutation π may minimize the max, no π a mere complex conjugation of another. **Figures** on the next page exhibit three examples. Each Z_J appears as a "•" , each z_j as a "+" ; thick dashed lines connect pairs in a minimized max.

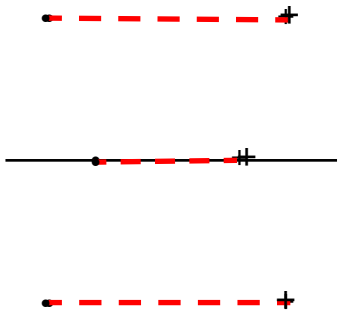
($\imath := \sqrt{-1}$)

Consequently no algorithm so simple as sorting, which minimizes the max when all zeros are real, can always succeed when complex zeros are present. Perhaps the simplest algorithm, and easiest to prove correct, examines all m**!** permutations; our $m! \le 6! = 720$. A function perms in MATLAB® generates all permutations of a complex row. However, sorting on the real parts and then ordering the imaginary parts has minimized the max for all our smaller values of n .

Task •2 is aided by the way our formulas for the true zeros present them. Let's drop subscripts for the moment and let Z be a true zero and z the tested zero-finder's computed zero. Each true zero is presented as $Z = 13/8 + x + y$ in which x and y are small addenda computable accurately enough from their formulas. As n increases, x and y get smaller. Then difference $Z - z = (13/8 - z) + x + y$ is computed more accurately from this equation's right-hand side than from its left because $(13/8 - z)$ shrinks with no rounding error unless z is very inaccurate. To attenuate the rounding error in Z it is never computed explicitly. (13/8 = 1**.**625)

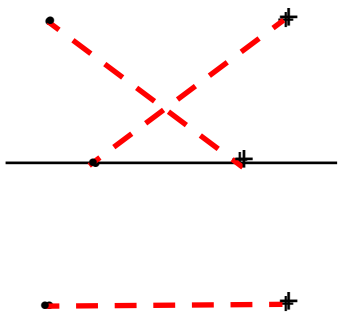
Figures: Two Permutations that Minimize the Max

• $\{Z_j\} := \{20+3\mathbf{i}, 20-3\mathbf{i}, 21\}$



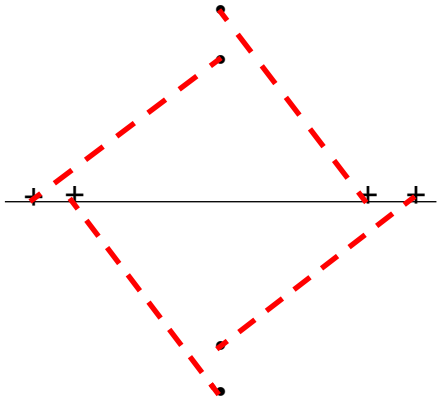
+ $\{z_{\pi(j)}\} := \{25+3\mathbf{i}, 25-3\mathbf{i}, 24\}$

• $\{Z_j\} := \{20+3\mathbf{i}, 20-3\mathbf{i}, 21\}$



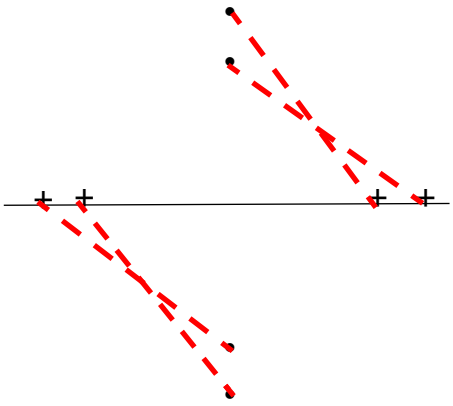
+ $\{z_{\pi(j)}\} := \{24, 25-3\mathbf{i}, 25+3\mathbf{i}\}$

• $\{Z_j\} := \{20+3\mathbf{i}, 20-3\mathbf{i}, 20+4\mathbf{i}, 20-4\mathbf{i}\}$



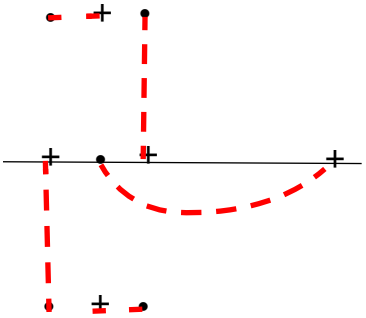
+ $\{z_{\pi(j)}\} := \{16, 24, 23, 17\}$

• $\{Z_j\} := \{20+3\mathbf{i}, 20-3\mathbf{i}, 20+4\mathbf{i}, 20-4\mathbf{i}\}$



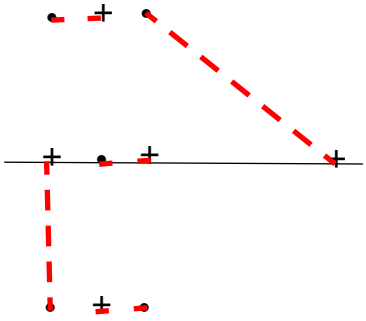
+ $\{z_{\pi(j)}\} := \{24, 16, 23, 17\}$

• $\{Z_j\} := \{19+3\imath, 19-3\imath, 20, 21+3\imath, 21-3\imath\}$



+ $\{z_{\pi(j)}\} := \{20+3\imath, 19, 25, 21, 20-3\imath\}$

• $\{Z_j\} := \{19+3\imath, 19-3\imath, 20, 21+3\imath, 21-3\imath\}$



+ $\{z_{\pi(j)}\} := \{20+3\imath, 19, 21, 25, 20-3\imath\}$

§5: Formulas Involving Fibonacci Numbers F(n)

These come in three flavors. "A = B" says A and B are the same values or functions, which may be useful later to compute A or help in other equations. "A := C" says expression C is a good way to evaluate A in floating-point. "A ≡ E" is the symbolic definition of A in terms of previously defined entities appearing in symbolic expression E; this formula may also serve to evaluate A in floating-point if no formula like "A ≡ E := C" is available.

Symbolic variable n is a positive integer; k is an arbitrary integer.

$\tau \equiv (1 + \sqrt{5})/2 = 1 + 1/\tau$; $(1 - \sqrt{5})/2 = -1/\tau$. All subsequent functions F… below are integers.

$F(k) \equiv (\tau^{2k} - (-1)^k)/(\tau^k \cdot \sqrt{5}) := F(k-1) + F(k-2) = -F(-k)\cdot(-1)^k$; $F(0) = 0$; $F(\pm 1) = 1$.

$F(k) = 3F(k-2) - F(k-4) = 4F(k-3) + F(k-6) = 7F(k-4) - F(k-8) = 11F(k-5) + F(k-10)$.

$F2(k) \equiv F(k-1) + F(k+1) = F2(k-1) + F2(k-2) = F2(-k)\cdot(-1)^k$; $F2(0) = 2$; $F2(\pm 1) = \pm 1$.

$F2(k) = 3F2(k-2) - F2(k-4) = \tau^k + (-1/\tau)^k$; $((1 \pm \sqrt{5})/2)^k = (F2(k) \pm F(k)\cdot\sqrt{5})/2$.

$F(2k) = F2(k)\cdot F(k)$; $F(2k \pm 1) = F(k)\cdot F2(k \pm 1) + (-1)^k$; $F2(2k) = 5F(k)^2 + 2\cdot(-1)^k$.

$F3(k) \equiv 5F(k)^2 + 3\cdot(-1)^k$; $F(3k) = F3(k)\cdot F(k)$; $F(3k \pm 1) = \pm(\pm F(3k \pm 3) - F(3k))/2$.

$F4(k) \equiv (5F(k)^2 + 2\cdot(-1)^k)\cdot F2(k) = F2(2k)\cdot F2(k)$; $F(4k) = F4(k)\cdot F(k)$.

$F5(k) \equiv 5\cdot(F(k)^2 + (-1)^k)\cdot F(k)^2 + 1$; $F(5k) = 5\cdot F5(k)\cdot F(k)$.

$X(n) \equiv F(n+1)/F(n) := 13/8 - F(n-6)/(8\cdot F(n))$.

$X2(k) \equiv F2(k+1)/F2(k) := 13/8 - F2(k-6)/(8\cdot F2(k))$.

After all definitions "A ≡ E" have been fed to a computerized algebra system, it can confirm quickly though laboriously all the other equations "A := C" and "A = B" as well as all the formulas below for the polynomial's zeros. The user of these formulas should confirm them if only to guard against transcription errors.

§6: Quadratic Polynomials' Zeros

$Q(n, x) \equiv F(n)\cdot x^2 - 2F(n+1)\cdot x + F(n+2)$ has two zeros $\{ X(n) \pm \mathbf{i}^n/F(n) \}$.

$Q2(k, x) \equiv F2(k)\cdot x^2 - 2F2(k+1)\cdot x + F2(k+2)$ has two zeros $\{ X2(k) \pm \mathbf{i}^{k+1}\cdot\sqrt{5}/F2(k) \}$.

$Xe(n) \equiv F(n+2)/F(n) := 21/8 - F(n-6)/(8F(n))$.

$X2e(k) \equiv F2(k+2)/F2(k) := 21/8 - F2(k-6)/(8F2(k))$

$Qe(n, x) \equiv F(n)\cdot x^2 - 2F(n+2)\cdot x + F(n+4)$ has two zeros $\{ Xe(n) \pm \mathbf{i}^n/F(n) \}$.

$Q2e(k, x) \equiv F2(k)\cdot x^2 - 2F2(k+2)\cdot x + F2(k+4)$ has two zeros $\{ X2e(k) \pm \mathbf{i}^{k+1}\cdot\sqrt{5}/F2(k) \}$.

§7: Cubic Polynomials' Zeros

The first six cubics descend from the foregoing four quadratics:

$Ç(n, x) \equiv (x + 1)\cdot Q(n, x) := F(n)\cdot x^3 - F2(n)\cdot x^2 - F(n-1)\cdot x + F(n+2)$ has three zeros

$$\{ -1,\quad X(n) \pm \mathbf{i}^n/F(n) \} .$$

$Ç2(k, x) \equiv (x + 1)\cdot Q2(k, x) := F2(k)\cdot x^3 - 5F(k)\cdot x^2 - F2(k-1)\cdot x + F2(k+2)$ has three zeros

$$\{-1,\quad X2(k) \pm \mathbf{i}^{k+1}\cdot\sqrt{5}/F2(k) \} .$$

$Çe(n, x) \equiv (x + 1)\cdot Qe(n, x) := F(n)\cdot x^3 - F(n+3)\cdot x^2 + F(n+1)\cdot x + F(n+4)$ has three zeros

$$\{ -1,\quad Xe(n) \pm \mathbf{i}^n/F(n) \} .$$

$Ç2e(k, x) \equiv (x + 1)\cdot Q2e(k, x) := F2(k)\cdot x^3 - F2(k+3)\cdot x^2 + F2(k+1)\cdot x + F2(k+4)$ has three zeros

$$\{ -1,\quad X2e(k) \pm \mathbf{i}^{k+1}\cdot\sqrt{5}/F2(k) \} .$$

$\overline{Ç}(n, x) \equiv (x - 1)\cdot Q(n, x) := F(n)\cdot x^3 - F(n+3)\cdot x^2 + F2(n+2)\cdot x - F(n+2)$ has three zeros

$$\{ +1,\quad X(n) \pm \mathbf{i}^n/F(n) \} .$$

$\overline{Ç2}(k, x) \equiv (x - 1)\cdot Q2(k, x) := F2(k)\cdot x^3 - F2(k+3)\cdot x^2 + 5F(k+2)\cdot x - F2(k+2)$ has three zeros

$$\{+1,\quad X2(k) \pm \mathbf{i}^{k+1}\cdot\sqrt{5}/F2(k) \} .$$

$C(n, x) \equiv F(n)\cdot x^3 - 3F(n+1)\cdot x^2 + 3F(n+2)\cdot x - F(n+3)$ has a special case worth handling first:
$C(3n, x)$ has integer factors and consequently has three comparatively uncomplicated zeros:

$$\{ X(n),\quad X(n) - 3F2(n)\cdot(-1)^n/(2F(3n)) \pm \mathbf{i}\cdot\sqrt{15}/(2F3(n)) \} .$$

To cope with $C(n, x)$ more generally, a cube root must be computed:

$b \equiv \sqrt[3]{\tau}$, the positive cube root; $p \equiv 1/b$.

$\Phi(n) \equiv p^n + (-b)^n$; $f(n) \equiv b^n - (-p)^n$.

Then $C(n, x)$ has three zeros

$$\{ X(n) + \Phi(n)/F(n),\quad X(n) - (\Phi(n) \pm \mathbf{i}\cdot f(n)\cdot\sqrt{3})/(2F(n)) \} .$$

§8: Quartic Polynomials' Zeros

$H(n, x) \equiv F(n)\cdot x^4 - 4F(n+1)\cdot x^3 + 6F(n+2)\cdot x^2 - 4F(n+3)\cdot x + F(n+4)$ comes in two flavors:
$H(2n, x) = Q(n, x)\cdot Q2(n, x)$ has four comparatively uncomplicated zeros, two of them real:

$$\{ X(n) \pm \mathbf{i}^n/F(n),\quad X2(n) \pm \mathbf{i}^{n+1}\cdot\sqrt{5}/F2(n) \} .$$

Because they entail complex square roots, the four zeros of $H(2n-1, x)$ are complicated:

$\{ X(2n-1) - (\mathbf{i} \pm \sqrt{-2 - \mathbf{i}\cdot F2(2n-1)})/F(2n-1) \}$ and their complex conjugates.

§9: Quintic Polynomials' Zeros

$K5(n, x) \equiv F5(n)\cdot F(n)\cdot x^5 - F(5n+1)\cdot x^4 + 2F(5n+2)\cdot x^3 - 2F(5n+3)\cdot x^2 + F(5n+4)\cdot x - F5(n+1)\cdot F(n+1)$

$:= F(5n)/5\cdot x^5 - F(5n+1)\cdot x^4 + 2F(5n+2)\cdot x^3 - 2F(5n+3)\cdot x^2 + F(5n+4)\cdot x - F(5n+5)/5$.

The integer factors of $K5(n, x)$ account for the simplicity of its one real zero $X(n)$. But four complex zeros of $K5(n, x)$ are complicated expressions, starting with the auxiliary function(s)

$\Delta5(n, \pm\sqrt{5}) \equiv \pm\sqrt{5}\cdot 2\cdot(F5(n)\cdot(-1)^n + F3(n)) + 10F5(n) + 2F3(n)\cdot(-1)^n + 4$.

Then, besides its real zero $X(n)$, the four complex zeros of $K5(n, x)$ are

$\{ X(n) + (\pm\sqrt{5}\cdot F(2n) - (F2(3n)\cdot(-1)^n + F2(n))/F(n) + \mathbf{i}\cdot\sqrt{\Delta5(n, \pm\sqrt{5})})/(4F5(n)) \}$

and their complex conjugates.

§10: Sixth Degree Polynomials' Zeros

These come in two flavors, V6 and W6 . The simpler one is

$V6(n, x) \equiv \ldots$

$F(6n)\cdot x^6 - 6F(6n+1)\cdot x^5 + 15F(6n+2)\cdot x^4 - 20F(6n+3)\cdot x^3 + 15F(6n+4)\cdot x^2 - 6F(6n+5)\cdot x + F(6n+6)$

because it has two real zeros, namely { X(n), X2(n) }, and four complex, namely

$\{ X2(2n) + (\pm(1 - 2X2(2n)) + \mathbf{i}\cdot\sqrt{15})/(2F2(2n) \pm 2) \}$ and their complex conjugates.

$W6(n, x) \equiv V6(n+\frac{1}{2}, x) := \ldots$

$F(6n+3)\cdot x^6 - 6F(6n+4)\cdot x^5 + 15F(6n+5)\cdot x^4 - 20F(6n+6)\cdot x^3 + 15F(6n+7)\cdot x^2 - 6F(6n+8)\cdot x + F(6n+9)$

has six complex zeros. Two of them are $\{ X(2n+1) \pm \mathbf{i}/F(2n+1) \}$. The other four are

$\{ X2(2n+1) + (\pm\sqrt{15} + \mathbf{i} - 2\mathbf{i}\cdot X2(2n+1))/(2F2(2n+1) + 2\mathbf{i}) \}$ and their complex conjugates.

§11: Numerical Examples

After the foregoing formulas were confirmed by *Derive*® they were translated into programs for MATLAB, whose function roots(C) treats the row C as the coefficients of a polynomial whose zeros are computed as the eigenvalues of the polynomial's *Companion* matrix. The first thing roots(C) does is divide C by the polynomial's leading coefficient, thereby incurring rounding errors that often do almost as much damage to the zeros as do the rest of roots' rounding errors. (Perhaps less damage might be done to zeros by dividing eigenvalues instead of C by that leading coefficient.) Consequently roots should be expected to lose all but a fraction 1/m of the 53 sig.bits (like 15 - 16 sig.dec.) carried by MATLAB's arithmetic when computing m zeros clustered closely, as our formulas reveal at larger values of n . They do not reveal how inaccurately roots finds smaller zeros when they are too much smaller than the others, but that is a story for another day.

Displayed below are the results from tests of roots(…). They show the polynomial's name, the value of parameter n , and the MinMax|Error| , followed by two columns, first the true zeros Z_j computed from the foregoing formulas, and then the corresponding errors $Z_J - z_{\pi(J)}$. Some of these are surprisingly big even for moderate values of n ; digits of Z_j jeopardized by these errors are *italicized*. Then the displayed results are assessed in §12.

Results computed for tests of roots(…) by MATLAB 5.2 on an ancient μ68040-based Apple Quadra 950

Quadratic polynomials

Q(n,x): n = 3 MinMaxErr = 0
TrueZ__rootsErr =
1.50000000000000 - 0.50000000000000**i** 0
1.50000000000000 + 0.50000000000000**i** 0

Q2(n,x): n = 3 MinMaxErr = 0
TrueZ__rootsErr =
2.30901699437495 0
1.19098300562505 0

Q(n,x): n = 36 MinMaxErr = 2.39181e-09
TrueZ__rootsErr =
1.6180340557*2755* -0.00000000239181
1.6180339217*7224* 0.00000000239181

Q2(n,x): n = 36 MinMaxErr = 2.75184e-09
TrueZ__rootsErr =
1.61803398874989 + 0.0000000*6697766***i** -0.00000000000000 + 0.00000000275184**i**
1.61803398874989 - 0.0000000*6697766***i** -0.00000000000000 - 0.00000000275184**i**

Q(n,x): n = 75 MinMaxErr = 1.10196e-08
TrueZ__rootsErr =
1.6180339*8874989* -0.00000001101956
1.6180339*8874989* 0.00000001101956

Q2(n,x): n = 75 MinMaxErr = 1.10196e-08
TrueZ__rootsErr =
1.6180339*8874990* -0.00000001101956
1.6180339*8874989* 0.00000001101956

Cubic polynomials

C(n,x): n = 3 MinMaxErr = 1.25607e-15
TrueZ__rootsErr =
1.75000000000000 + 0.96824583655185**i** 0.00000000000000 - 0.00000000000000**i**
1.75000000000000 - 0.96824583655185**i** 0.00000000000000 + 0.00000000000000**i**
1.00000000000000 0

C(3*n,x): n = 3 MinMaxErr = 3.01981e-14
TrueZ__rootsErr =
1.67647058823529 + 0.11391127488845**i** 0.00000000000001 + 0.00000000000003**i**
1.67647058823529 - 0.11391127488845**i** 0.00000000000001 - 0.00000000000003**i**
1.50000000000000 -0.00000000000003

C(n,x): n = 36 MinMaxErr = 1.40799e-06
TrueZ__rootsErr =
1.61805555555556 0.00000140796202
1.61802320534707 + 0.00001867704130**i** -0.00000070398101 + 0.00000121935939**i**
1.61802320534707 - 0.00001867704130**i** -0.00000070398101 - 0.00000121935939**i**

C(3*n,x): n = 12 MinMaxErr = 1.40799e-06
TrueZ__rootsErr =
1.61805555555556 0.00000140796202
1.61802320534707 + 0.00001867704130**i** -0.00000070398101 + 0.00000121935939**i**
1.61802320534707 - 0.00001867704130**i** -0.00000070398101 - 0.00000121935939**i**

C(n,x): n = 71 MinMaxErr = 1.4186e-05
TrueZ__rootsErr =
1.618033398889324 + 0.0000000002*4828***i*** -0.00001418596060 + 0.00000000024828**i**
1.61803398889324 - 0.00000000024828**i** 0.00000709319531 + 0.00001228515113**i**
1.61803398846321 0.00000709276528 - 0.00001228539941**i**

C(3*n,x): n = 24 MinMaxErr = 1.41859e-05
TrueZ__rootsErr =
1.618033398895790 -0.00001418589593
1.61803398864589 + 0.00000000018014**i** 0.00000709294796 - 0.00001228521927**i**
1.61803398864589 - 0.00000000018014**i** 0.00000709294796 + 0.00001228521927**i**

Quartic polynomials
~~~~~~~~~~~~~~~~~~~
H(n,x): n = 3 MinMaxErr = 3.12642e-15
TrueZ__rootsErr =
2.05589297025142 - 1.39945371997393**i** -0.00000000000000 - 0.00000000000000**i**
2.05589297025142 + 1.39945371997393**i** -0.00000000000000 + 0.00000000000000**i**
0.94410702974858 + 0.39945371997393**i** 0.00000000000000 + 0.00000000000000**i**
0.94410702974858 - 0.39945371997393**i** 0.00000000000000 - 0.00000000000000**i**

H(n,x): n = 4 MinMaxErr = 4.88498e-15
TrueZ__rootsErr =
3.00000000000000 0.00000000000000
1.33333333333333 + 0.74535599249993**i** 0.00000000000000 + 0.00000000000000**i**
1.33333333333333 - 0.74535599249993**i** 0.00000000000000 - 0.00000000000000**i**
1.00000000000000 -0.00000000000000

H(n,x): n = 21 MinMaxErr = 1.16025e-09
TrueZ__rootsErr =
1.628140048*84777* - 0.010198247*23144***i*** -0.00000000081705 + 0.00000000082377**i**
1.628140048*84777* + 0.010198247*23144***i*** -0.00000000081705 - 0.00000000082377**i**
1.607927921*18695* + 0.010015532*08435***i*** 0.00000000081705 - 0.00000000081044**i**
1.607927921*18695* - 0.010015532*08435***i*** 0.00000000081705 + 0.00000000081044**i**

H(n,x): n = 22 MinMaxErr = 1.54012e-11
TrueZ__rootsErr =
1.62932697476131 0.00000000001540
1.617977528089*89* + 0.011235955056*18***i*** -0.00000000000173 + 0.00000000001382**i**
1.61797752808*989* - 0.01123595505*618***i*** -0.00000000000173 - 0.00000000001382**i**
1.60685392976131 -0.00000000001195
~~~~~~~~~~~~~~~~~~~

H(n,x): n = 69 MinMaxErr = 0.00017734
TrueZ__rootsErr =

1.618*03408622561* - 0.000*00009747572***i***	-0.00012538326364 + 0.00012535252315***i***
1.618*03408622561* + 0.000*00009747572***i***	-0.00012538326364 - 0.00012535252315***i***
1.618*03389127418* + 0.000*00009747572***i***	0.00012538326364 - 0.00012541397166***i***
1.618*03389127418* - 0.000*00009747572***i***	0.00012538326364 + 0.00012541397166***i***

H(n,x): n = 70 MinMaxErr = 0.000177402
TrueZ__rootsErr =

1.618*03409712203*	-0.00012537236722 - 0.00012544999887***i***
1.618*03398874989* + 0.000*00010837213***i***	0.00012548073935 - 0.00012540307524***i***
1.618*03398874989* - 0.000*00010837213***i***	-0.00012548073936 + 0.00012534162674***i***
1.618*03388037777*	0.00012537236723 + 0.00012551144737***i***

Quintic polynomials
~~~~~~~~~~~~~~~~~~~~

K5(n,x): n = 3 MinMaxErr = 2.93274e-11
TrueZ__rootsErr =

1.720854687*78688* + 0.080230161*59326***i***	0.00000000002181 + 0.00000000001448***i***
1.720854687*78688* - 0.080230161*59326***i***	0.00000000002181 - 0.00000000001448***i***
1.57422727942*624* + 0.11422403794*972***i***	-0.00000000000714 + 0.00000000002733***i***
1.57422727942*624* - 0.11422403794*972***i***	-0.00000000000714 - 0.00000000002733***i***
1.500000000*0000*	-0.00000000002933

K5(n,x): n = 4 MinMaxErr = 6.19257e-10
TrueZ__rootsErr =

1.666666666*66667*	0.00000000058950
1.631905447*63165* + 0.045850299*23838***i***	0.00000000020018 + 0.00000000056588***i***
1.631905447*63165* - 0.045850299*23838***i***	0.00000000020018 - 0.00000000056588***i***
1.57984621*33953* + 0.02703375*08134***i***	-0.00000000049492 + 0.00000000037220***i***
1.57984621*33953* - 0.02703375*08134***i***	-0.00000000049492 - 0.00000000037220***i***

K5(n,x): n = 7 MinMaxErr = 0.000135072
TrueZ__rootsErr =

1.620*18089057473* + 0.0015*6210595591***i***	-0.00010924050028 - 0.00007944088592***i***
1.620*18089057473* - 0.0015*6210595591***i***	-0.00010924050028 + 0.00007944088592***i***
1.6172*1177360769* + 0.002*52084110960***i***	0.00004179526439 - 0.00012832457469***i***
1.6172*1177360769* - 0.002*52084110960***i***	0.00004179526439 + 0.00012832457469***i***
1.615*38461538462*	0.00013489047179

K5(n,x): n = 8 MinMaxErr = 0.000641489
TrueZ__rootsErr =

1.619*04761904762*	-0.00064148897910
1.618*34670424759* + 0.000*96385261194***i***	-0.00019796615204 - 0.00061000765474***i***
1.618*34670424759* - 0.000*96385261194***i***	-0.00019796615204 + 0.00061000765474***i***
1.617*21445810334* + 0.000*59509057704***i***	0.00051871064160 - 0.00037669091927***i***
1.617*21445810334* - 0.000*59509057704***i***	0.00051871064160 + 0.00037669091927***i***
~~~~~~~~~~~~~~~~~~~~

K5(n,x): n = 11 MinMaxErr = 0.00160978
TrueZ__rootsErr =

1.61807966797730 + 0.00003318893723**i**	-0.00160944004942 + 0.00003318893723**i**
1.61807966797730 - 0.00003318893723**i**	-0.00046500242233 + 0.00154067132945**i**
1.61801653985250 + 0.00005369779648**i**	-0.00052813054714 - 0.00152016247020**i**
1.61801653985250 - 0.00005369779648**i**	0.00132079239075 + 0.00091808369983**i**
1.61797752808989	0.00128178062815 - 0.00097178149630**i**

K5(n,x): n = 12 MinMaxErr = 0.00163355
TrueZ__rootsErr =

1.61805555555556	-0.00163355247117
1.61804065302680 + 0.00002051117549**i**	-0.00050401737284 - 0.00155334909119**i**
1.61804065302680 - 0.00002051117549**i**	-0.00050401737284 + 0.00155334909119**i**
1.61801654107016 + 0.00001267633022**i**	0.00132079360842 - 0.00095910516609**i**
1.61801654107016 - 0.00001267633022**i**	0.00132079360842 + 0.00095910516609**i**

6th degree polynomials
~~~~~~~~~~~~~~~~~~~~~~~~

V6(n,x): n = 2 MinMaxErr = 3.52379e-12
TrueZ__rootsErr =

2.00000000000000	0.00000000000248
1.75000000000000 + 0.32274861218395**i**	0.00000000000193 + 0.00000000000230**i**
1.75000000000000 - 0.32274861218395**i**	0.00000000000193 - 0.00000000000230**i**
1.43750000000000 + 0.24206145913796**i**	-0.00000000000141 + 0.00000000000311**i**
1.43750000000000 - 0.24206145913796**i**	-0.00000000000141 - 0.00000000000311**i**
1.33333333333333	-0.00000000000352

W6(n,x): n = 2 MinMaxErr = 2.77226e-11
TrueZ__rootsErr =

1.80165088855853 - 0.11833189895987**i**	-0.00000000001385 + 0.00000000000343**i**
1.80165088855853 + 0.11833189895987**i**	-0.00000000001385 - 0.00000000000343**i**
1.60000000000000 - 0.20000000000000**i**	-0.00000000000792 + 0.00000000002056**i**
1.60000000000000 + 0.20000000000000**i**	-0.00000000000792 - 0.00000000002056**i**
1.45244747209721 - 0.08658613382702**i**	0.00000000002177 + 0.00000000001716**i**
1.45244747209721 + 0.08658613382702**i**	0.00000000002177 - 0.00000000001716**i**

V6(n,x): n = 3 MinMaxErr = 4.99161e-11
TrueZ__rootsErr =

1.75000000000000	-0.00000000004806
1.67647058823529 + 0.11391127488845**i**	-0.00000000002427 - 0.00000000004362**i**
1.67647058823529 - 0.11391127488845**i**	-0.00000000002427 + 0.00000000004362**i**
1.55263157894737 + 0.10192061437388**i**	0.00000000002697 - 0.00000000003840**i**
1.55263157894737 - 0.10192061437388**i**	0.00000000002697 + 0.00000000003840**i**
1.50000000000000	0.00000000004268

W6(n,x): n = 3 MinMaxErr = 3.32769e-09
TrueZ__rootsErr =

1.68605493885987 - 0.04089844616758**i**	0.00000000254323 - 0.00000000131123**i**
1.68605493885987 + 0.04089844616758**i**	0.00000000254323 + 0.00000000131123**i**
1.61538461538462 - 0.07692307692308**i**	0.00000000026943 - 0.00000000309353**i**
1.61538461538462 + 0.07692307692308**i**	0.00000000026943 + 0.00000000309353**i**
1.55266240080759 - 0.03629870347612**i**	-0.00000000281266 - 0.00000000177834**i**
1.55266240080759 + 0.03629870347612**i**	-0.00000000281266 + 0.00000000177834**i**
~~~~~~~~~~~~~~~~~~~~~~~~

V6(n,x): n = 6 MinMaxErr = 0.00372244
TrueZ__rootsErr =

1.62500000000000	0.00347658051384
1.62149532710280 + 0.00603268433989**i**	0.00184877715014 + 0.00300463362453**i**
1.62149532710280 - 0.00603268433989**i**	0.00184877715014 - 0.00300463362453**i**
1.61455108359133 + 0.00599533025729**i**	-0.00172584597386 + 0.00321540684212**i**
1.61455108359133 - 0.00599533025729**i**	-0.00172584597386 - 0.00321540684212**i**
1.61111111111111	-0.00372244286640

W6(n,x): n = 6 MinMaxErr = 0.00100037
TrueZ__rootsErr =

1.62175496850777 - 0.00215308055376**i**	0.00083769049481 - 0.00054682821446**i**
1.62175496850777 + 0.00215308055376**i**	0.00083769049481 + 0.00054682821446**i**
1.61802575107296 - 0.00429184549356**i**	-0.00010145449730 - 0.00090448297626**i**
1.61802575107296 + 0.00429184549356**i**	-0.00010145449730 + 0.00090448297626**i**
1.61432124666895 - 0.00213881237364**i**	-0.00073623599751 - 0.00037037707007**i**
1.61432124666895 + 0.00213881237364**i**	-0.00073623599751 + 0.00037037707007**i**

V6(n,x): n = 7 MinMaxErr = 0.00429552
TrueZ__rootsErr =

1.62068965517241	-0.00429552281864
1.61935866983373 + 0.00229987134573**i**	-0.00214011690911 - 0.00372002341476**i**
1.61935866983373 - 0.00229987134573**i**	-0.00214011690911 + 0.00372002341476**i**
1.61670616113744 + 0.00229442141363**i**	0.00214774684872 - 0.00370680809504**i**
1.61670616113744 - 0.00229442141363**i**	0.00214774684872 + 0.00370680809504**i**
1.61538461538462	0.00428026293943

W6(n,x): n = 7 MinMaxErr = 0.00559143
TrueZ__rootsErr =

1.61945430422200 - 0.00082071429928**i**	-0.00553087376906 - 0.00082071429928**i**
1.61945430422200 + 0.00082071429928**i**	-0.00204448252085 - 0.00519918046120**i**
1.61803278688525 - 0.00163934426230**i**	-0.00346599985760 + 0.00438055049819**i**
1.61803278688525 + 0.00163934426230**i**	0.00347437259652 - 0.00436188524637**i**
1.61661487514244 - 0.00081863260641**i**	0.00205646085372 + 0.00518259690226**i**
1.61661487514244 + 0.00081863260641**i**	0.00551052269726 + 0.00081863260641**i**

V6(n,x): n = 10 MinMaxErr = 0.00680336
TrueZ__rootsErr =

1.61818181818182	-0.00680335980924
1.61810789369298 + 0.00012802404291**i**	-0.00339089304986 - 0.00589187071757**i**
1.61810789369298 - 0.00012802404291**i**	-0.00339089304986 + 0.00589187071757**i**
1.61796007403490 + 0.00012800711747**i**	0.00340165974618 - 0.00587322239119**i**
1.61796007403490 - 0.00012800711747**i**	0.00340165974618 + 0.00587322239119**i**
1.61788617886179	0.00678182641660

W6(n,x): n = 10 MinMaxErr = 0.00687222
TrueZ__rootsErr =

1.61811310859581 - 0.00004568201947**i**	-0.00687206939524 - 0.00004568201947**i**
1.61811310859581 + 0.00004568201947**i**	-0.00338567814703 - 0.00597421274101**i**
1.61803398501736 - 0.00009135757354**i**	-0.00346480172549 + 0.00592853718694**i**
1.61803398501736 + 0.00009135757354**i**	0.00347557072864 - 0.00590987193512**i**
1.61795487263651 - 0.00004567555453**i**	0.00339645834779 + 0.00595555395414**i**
1.61795487263651 + 0.00004567555453**i**	0.00685052019133 + 0.00004567555453**i**

```
 V6(n,x):  n = 11     MinMaxErr = 0.00689473
TrueZ__rootsErr =
  1.61809045226131                          -0.00689472572975
  1.61806221907984 + 0.00004889883524i      -0.00343656766300 - 0.00597099592525i
  1.61806221907984 - 0.00004889883524i      -0.00343656766300 + 0.00597099592525i
  1.61800575699424 + 0.00004889636585i       0.00344734270552 - 0.00595233314282i
  1.61800575699424 - 0.00004889636585i       0.00344734270552 + 0.00595233314282i
  1.61797752808989                          0.00687317564470
```

§12: Assessment

The errors in zeros computed by roots(…) can be surprisingly big. Different generations of MATLAB run on diverse computer arithmetics, including Intel's in Windows® machines and old PowerPC®-based Macintoshes®, produced results different than displayed above, but the MinMax|Errors| almost never differed by factors bigger than 2 . Similar results were obtained when the *Companion* matrix in roots(…) was replaced by M. Fiedler's Companion matrix as described on pp. 7 - 8 of my posting …/5Mar14.pdf . Even if differences between computed zeros lose relatively few of their leading digits to cancellation, the remaining digits can be quite wrong. Why are they so wrong?

The variations in computed zeros cannot be blamed entirely upon initial roundings of quotients when coefficients were divided by the leading coefficient. Neither should variations be blamed entirely upon roundoff inside program roots, which rarely does more damage than perturbations in the coefficients' last few bits. Zeros of some polynomials expressed as sums of monomials are hypersensitive to small perturbations of coefficients. Our surprise comes from seeing that hypersensitivity in polynomials of such low degrees.

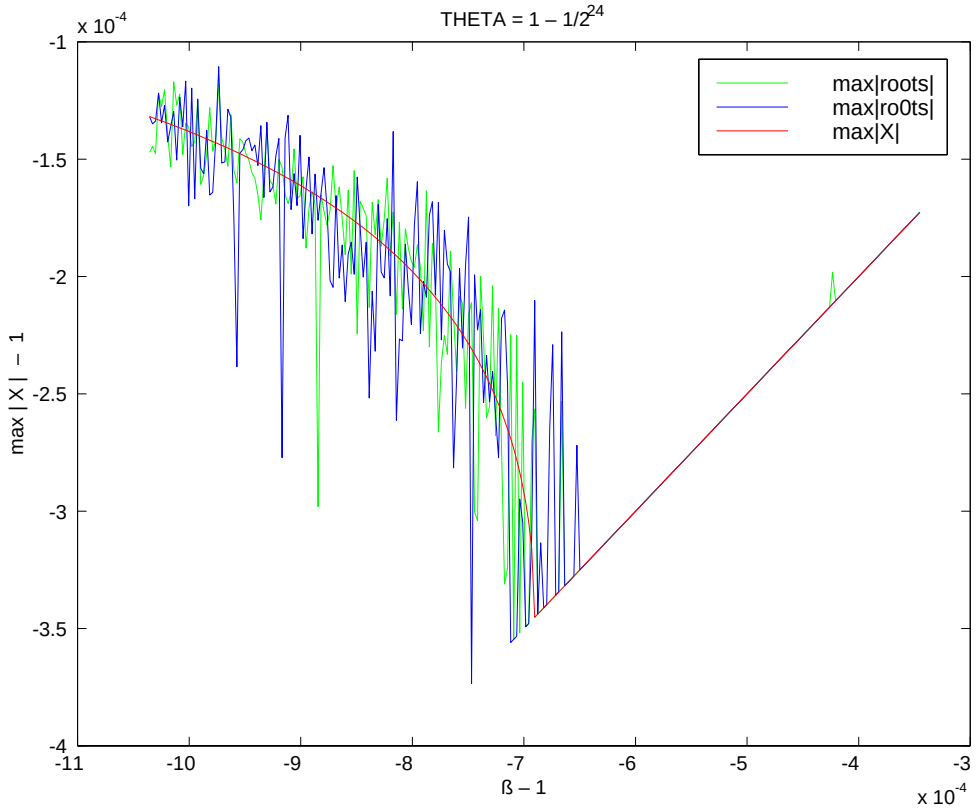
When might errors like the worst ones seen above be troublesome? That can happen during the numerical optimization of coefficients' parameters according to criteria imposed upon the zeros of a polynomial. Often they are repeated or clustered closely when the optimum is achieved, but then numerical errors may obscure it. Here is an example contrived for didactic effect:

Given a positive value $\theta < 1$ but near 1 , we seek the best value of parameter β to minimize the maximum of the magnitudes of the zeros x of the quartic polynomial

$$H(x, \theta, \beta) := x^4 - (\beta+1)(\theta^2+\theta)\cdot x^3 + (\theta^3\cdot(\beta+1)^2+2\beta)\cdot x^2 - (\theta^2+\theta)(\beta^2+\beta)\cdot x + \beta^2 .$$

We shall pretend not to know a formula $X(\theta, \beta, \pm) := \frac{1}{2}\left((\beta+1)\theta \pm \sqrt{((\beta+1)^2\cdot\theta^2 - 4\beta)} \right)$ for the four zeros of H , namely $\{X(\theta, \beta, \pm),\ X(\theta^2, \beta, \pm)\}$, and the minimum Ξ of their maximum magnitude, namely $\Xi(\theta) := \theta/(1 + \sqrt{((1-\theta)(1+\theta))})$, achieved at the optimal value of $\beta = \Xi^2$. Instead we shall plot graphs, as functions of β , of the maximum magnitude of the zeros of H computed three ways: in green from roots, in blue from ro0ts, and in red from formula X . Here ro0ts is my program adapted from MATLAB's roots by replacing its Companion matrix by Fiedler's as described in …/5Mar14.pdf . To obtain readable numbers at tick-marks on the axes, we actually plot $\max|X| - 1$ against $\beta - 1$.

$$\theta = 1 - 1/2^{24} = 0.99999994039536$$

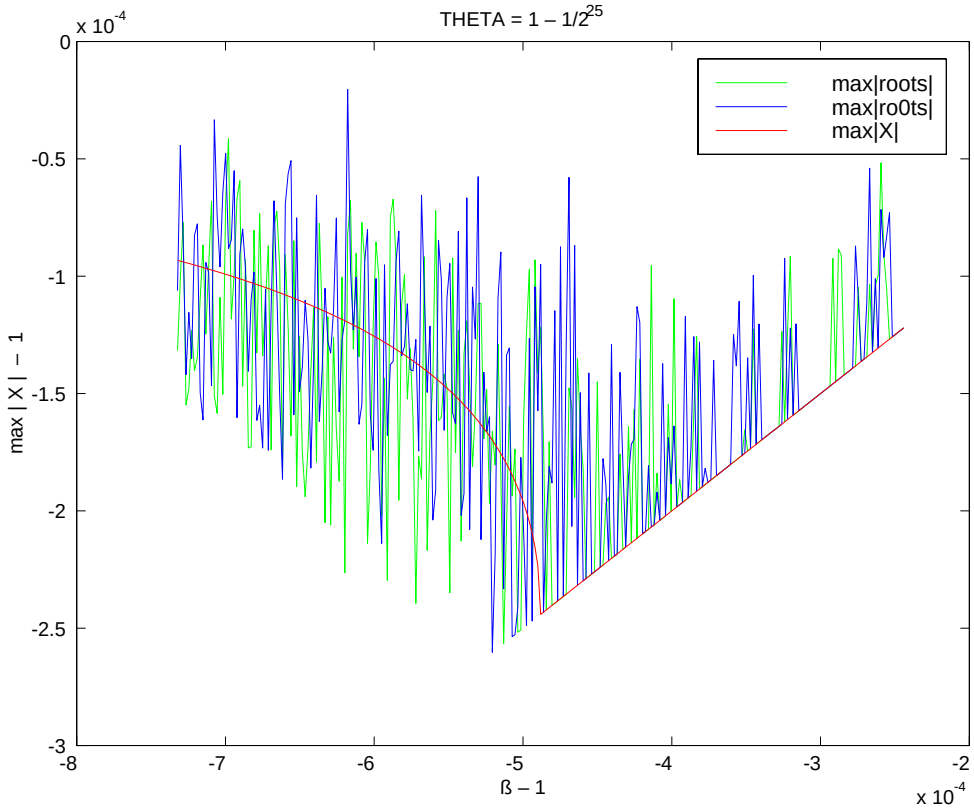


When β takes its sharply determined optimum value $\beta = \Xi^2 = 0.9993097043806$, all four true zeros X(…) have the same minimized magnitude Ξ and two of them coincide at Ξ :

True zeros X(…)	Errors in roots(…)	Errors in ro0ts(…)
0.9996*5479260621*	0.00008875	0.00013494
0.9996*5479260621*	-0.00008875	-0.00013496
0.99965473302214 + 0.0003*4514778902***i***	-0.00000000 - 0.00001123***i***	0.00000001 - 0.00002544***i***
0.99965473302214 - 0.0003*4514778902***i***	-0.00000000 + 0.00001123***i***	0.00000001 + 0.00002544***i***

Though MATLAB's arithmetic carries over 15 sig.dec., at most the leading four decimal digits of the computed roots(…) and ro0ts(…) can be trusted, and these determine at most the three leading decimal digits of β .

$$\theta = 1 - 1/2^{25} = 0.99999997019768$$



When β takes its sharply determined optimum value $\beta = \Xi^2 = 0.99951183793383$, all four true zeros X(…) have the same minimized magnitude Ξ and two of them coincide at Ξ :

True zeros X(…)	Errors in roots(…)	Errors in ro0ts(…)
0.999*75588917187*	0.00012230	0.00014906
0.999*75588917187*	-0.00012232	-0.00014909
0.9997558*5937682* + 0.0002*4408102581*i	0.00000001 - 0.00002893i	0.00000002 - 0.00004192i
0.9997558*5937682* - 0.0002*4408102581*i	0.00000001 + 0.00002893i	0.00000002 + 0.00004192i

Though MATLAB's arithmetic carries over 15 sig.dec., at most the leading three decimal digits of the computed roots(…) and ro0ts(…) can be trusted, and these determine at most the three leading decimal digits of β. However, that may be more than can reasonably be expected from a program that searches numerically for an optimum after rounding off the coefficients.

Now that the foregoing numerical results are visible, they imply that better numerical results can be obtained by replacing variables x by y := 1–x , θ by t := 1–θ and β by b := 1–β to get

$$H(1-y, 1-t, 1-b) = y^4 + \big(2(t-3)t - (t-1)(t-2)b\big)\cdot y^3 -$$
$$- \big(b^2\cdot(t-1)^3 - (4t^2 - 9t + 3)bt + (4t^2 - 6t - 6)t\big)\cdot y^2 + \big((3b + (8-5b)t + 2(b-2)t^2)(b-2)t\big)\cdot y +$$
$$+ (2-t)(b-2)^2\cdot t^2 .$$

Then solve H = 0 numerically for roots Y to get X := 1–Y much more accurately than before. Note that the change of variables was carried out exactly, *i.e.* with infinite precision.

§13: Conclusion

Inaccuracies exposed by the tests above reinforce the demands of *Prudence*: that our numerical root-finder's accuracy be tested before we rely upon it. If then we desire numerically computed zeros more nearly faithful than roots(…) to a polynomial's coefficients stored in a computer's memory, we shall have to employ extra-precise arithmetic. The preparation of coefficients may entail extra-precise arithmetic too. How shall we discover whether extra-precise arithmetic is necessary to cope with polynomials that differ greatly from the ones tested here? This question may be answered afterwards by easily computed error-bounds, for which see

`www.eecs.berkeley.edu/~wkahan/Math128/PolyZbnd.pdf`