Tests for the Accuracy of Polynomial Zero-Finders

Prepared for Prof. Ming Gu's
Numerical Linear Algebra Seminar
380 Soda Hall, Noon - 1 pm., Wed. 28 Oct. 2015

by
W. Kahan, Professor Emeritus (*i.e.* retired)
Math. Dept. & Computer Sci. Dept.
Univ. of Calif. @ Berkeley

This is posted at `<www.eecs.berkeley.edu/~wkahan/28Oct15.pdf>`

Details are posted at
`<www.eecs.berkeley.edu/~wkahan/Math128/Fibs_2_6.pdf>`
`<www.eecs.berkeley.edu/~wkahan/Math128/PolyZbnd.pdf>`

Tests for the Accuracy of Polynomial Zero-Finders

Abstract

It is imprudent to trust a numerical zero-finder without first testing it. Some software used to compute zeros of polynomials has been found surprisingly inaccurate for polynomials even of low degree. Test data provided here takes the form of families of polynomials of degrees 2 to 6 with integer coefficients computable exactly in floating-point, and with zeros whose accuracies challenge numerical methods increasingly as a parameter n is increased. Provided too are formulas to compute the polynomials' zeros extra-accurately without extra-precise arithmetic, so they can be compared with a zero-finder's results to test their accuracy. If these polynomials differ too much from the ones you care about, how do you test your zero-finder on your data? Two easily computed error-bounds, one of them classical, are offered here and compared to help you decide which is better suited to your needs.

Details are posted at
<www.eecs.berkeley.edu/~wkahan/Math128/Fibs_2_6.pdf>
<www.eecs.berkeley.edu/~wkahan/Math128/PolyZbnd.pdf>

Ideally, Polynomials to Test Zero-Finders' Accuracies should have …

•> Coefficients computable *EXACTLY* in floating-point arithmetic
because rounding off coefficients alters zeros, sometimes a lot.

Our test-polynomials' coefficients come from Fibonacci numbers EXACTLY.

•> Zeros computable *EXTRA-ACCURATELY,* and without extra-precise arithmetic,
so that differences from a zero-finder's results are truly their errors, and
computable by the arithmetics of almost all programming languages.

Our test-polynomial's zeros require no more than IEEE-745 (1985 or 2008).

•> Ideally, a wide range of challenges for numerical zero-finders; but …

Our polynomials test only accuracies of clustered zeros, only degrees 2 - 6 .

(Other polynomials are needed to test high degrees, zeros of widely divagated magnitudes, resistance to premature over/underflow, ease of use, complex coefficients, speed, preservation of symmetries, ∞ and *NaN* inputs, … .)

Ideally, Polynomials to Test Zero-Finders' Accuracies should *Avoid* …

•> Small Integers, either as coefficients or as zeros.

WHY ?

Ideally, Polynomials to Test Zero-Finders' Accuracies should *Avoid* …

•> Small Integers, either as coefficients or as zeros.

Why ?

ROUNDOFF is ACCIDENTAL, RAGGED but NOT RANDOM.

That is why a realistic assessment of roundoff's impact upon a program can require that accuracy be sampled at vastly many numerical inputs to expose both typical and worst-case behavior.

Small integers are best avoided because they often incur **ATYPICAL ROUNDINGS** .

Floating-point Example: (Not for over-optimizing compilers)

q := k/d ; p := q·d ; L := (p = k) ; ... Is L *True* , or *False* ?

If these assignments are rounded according to IEEE 754 (1985) Binary, and if k and d are independent random floating-point numbers, L is *False* for about 11% of them.

But for all small integers $|k| \le 8000000$, say, and for every integer |d| drawn from { 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 16, 17, 18, 20, …} , L is always *True*. *cf.* the stopped clock

Ideally, Polynomials to Test Zero-Finders' Accuracies should *Avoid* …

•> Small Integers, either as coefficients or as zeros.

When *all* the coefficients a_j of a test-polynomial $\sum_j a_j \cdot x^{m-j}$ are small integers,
they should be *S T R E T C H E D* by a pseudo-random multiplication.

STRETCHING: For binary floating-point with p sig.bits,
Generate a pseudo-random integer R between 2^{p-1} and $2^p - 1$;
Obtain a multiplier M := floor($R/\max_j |a_j|$) ;
If $M > 0$, replace each a_j by $M \cdot a_j$; … NO ROUNDOFF OCCURS
Then run the test to compute zeros;
M may change their roundoff, but not the polynomial's true zeros.

(If $M = 0$, the coefficients a_j cannot *all* be small integers.)

All our test-polynomials' coefficients are derived from Fibonacci numbers F(n) .
They are small integers for small values of n , and should be stretched before use.

How well does one set of zeros approximate another ?

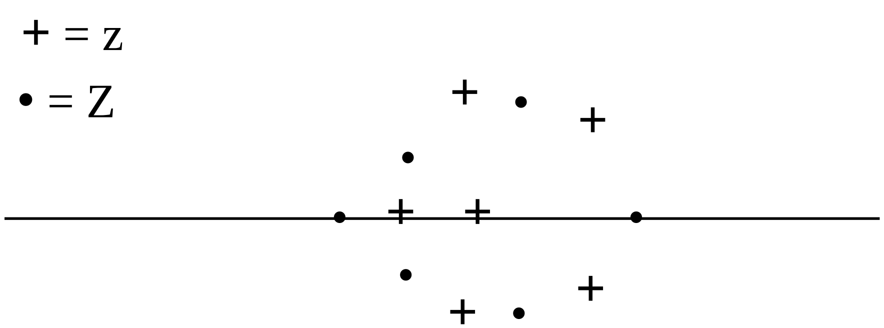
Suppose the True Zeros are $Z_1, Z_2, Z_3, \ldots, Z_J, \ldots, Z_m$, and
the Computed Zeros are $z_1, z_2, z_3, \ldots, z_j, \ldots, z_m$.

Which differences are the Errors $Z_J - z_j$?

If every zero Z_J and z_j is Real, sort them separately first to minimize $\max_J |Z_J - z_J|$.

But if some zeros are Complex (not real), though they come in complex conjugate pairs, the ordering that minimizes $\max_J |Z_J - z_J|$ can be unobvious.

- -



Which + goes with which • ?

How well does one set of zeros approximate another ?

Suppose the True Zeros are $Z_1, Z_2, Z_3, \ldots, Z_J, \ldots, Z_m$, and
the Computed Zeros are $z_1, z_2, z_3, \ldots, z_j, \ldots, z_m$.

Which differences are the Errors $Z_J - z_j$?

If every zero Z_J and z_j is Real, sort them separately first to minimize $\max_J |Z_J - z_J|$.

But if some zeros are Complex (not real), though they come in complex conjugate pairs, the ordering that minimizes $\max_J |Z_J - z_J|$ can be unobvious.

- -

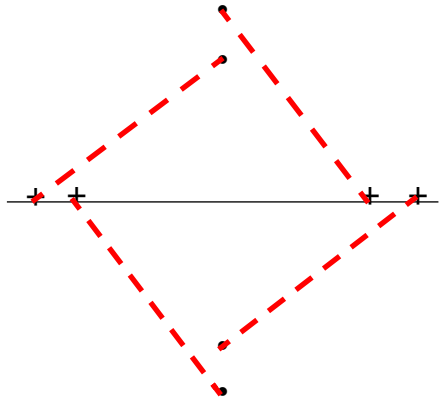
Let π range over Permutations of $\{ 1, 2, 3, \ldots, m \}$. The gauge of difference we seek is

$$\text{MinMaxError} := \min_{\pi} \max_J |Z_J - z_{\pi(J)}| ,$$

The minimizing permutation π need not be unique.
Must all m**!** of them be examined?
Our $m! \le 6! = 720$; it's tolerable.
I wish I had a better idea.

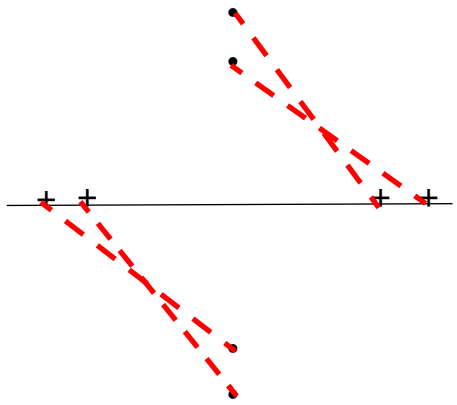
Two ambiguous examples, one with m = 4, one with m = 5 :

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



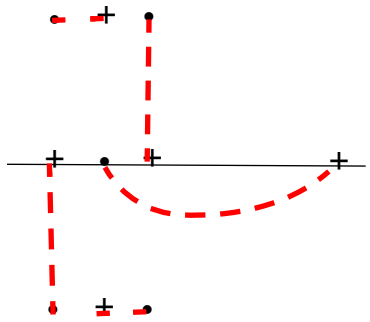
+ $\{z_{\pi(j)}\} := \{16, 24, 23, 17\}$

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



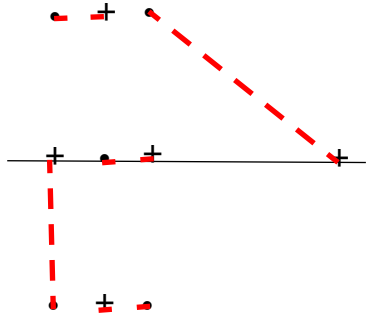
+ $\{z_{\pi(j)}\} := \{24, 16, 23, 17\}$

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



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

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



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

Extra-Accurate presentations of true zeros Z

Why?

True zero Z ; computed zero z ; error $z - Z$.

Rounding Z before the subtraction $z - Z$ may obscure how accurately z has been computed when it has been computed very accurately.

Instead of rounding off the true zero Z , our formulas present it in three pieces:

E.g., " $Z = 13/8 + x + \xi$ " with $13/8 = 1.625 >> |x| >> |\xi|$ (when n is big).

Then the error $z - Z$ is actually computed as

$$\text{Error} := ((z - 13/8) - x) - \xi .$$

Unless z is terribly inaccurate, the subtraction $z - 13/8$ cancels exactly (no rounding) and subsequent subtractions produce $z - Z$ as if computed with one extra sig.dec.

Three Flavors of Equations

Equations serve diverse purposes. Some serve in a program's assignments. *E.g.*:

$$F(0) := 0 ; \quad F(\pm 1) := \pm 1 ; \quad F(n+1) := F(n) + F(n-1) ;$$
$$F2(n) := F(n-1) + F(n+1) ;$$

These include a ***Recurrence*** to compute an array of Fibonacci numbers $F(n)$, and then another assignment to compute an array of related numbers $F2(n)$.

Some equations should be checked for correctness by a computerized algebra system like MATHEMATICA®, MAPLE®, DERIVE®, … .

E.g.: $F(2n) = F2(n)\cdot F(n) ; \quad F(2n \pm 1) = F2(n \pm 1)\cdot F(n) + (-1)^n .$

But the algebra systems cannot confirm these equations for *all* (*i.e.*, *symbolic*) n because they turn the ***Recurrence*** for $F(n)$ into ***Recursion***. A third flavor of equation is needed:

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

If these definitions precede the previous equations they all become easy to confirm, as are

$$(1 - \sqrt{5})/2 = -1/\tau ; \qquad ((1 \pm \sqrt{5})/2)^n = (F2(n) \pm F(n)\cdot\sqrt{5})/2 ;$$
$$F(3n) = (5F(n)^2 + 3\cdot(-1)^n)\cdot F(n) ; \quad F(3n \pm 1) = \pm(\pm F(3n \pm 3) - F(3n)) ;$$

regardless of whether τ is ever computed in floating-point.

Three Flavors of Equations

•> "A = B" ;

Expressions A and B take the same values, provided this equation is confirmed. It may help to shorten the computation or confirmation of other equations.

•> "A := C" ;

Expression C is a good way to compute variable A , provided the equation is confirmed.

•> "A ≡ E" ;

Symbolic expression E defines A for later use by an automated algebra system, and may be used to compute A in the absence of "A ≡ E := C" .

More Examples of Equations:

$X(n) \equiv F(n+1)/F(n) := 13/8 - F(n-6)/(8F(n))$; (recall the presentation of zeros Z)

$b \equiv \sqrt[3]{\tau}$, the positive cube root ; $p \equiv 1/b$; (will be needed for cubics' zeros)

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

Examples of Polynomials and their Zeros

Cubic: $C(n, x) \equiv F(n)\cdot x^3 - 3F(n+1)\cdot x^2 + 3F(n+2)\cdot x - F(n+3)$

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

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

Sixth degree: $V6(n, x) \equiv F(6n)\cdot x^6 - 6F(6n+1)\cdot x^5 + 15F(6n+2)\cdot x^4 - 20F(6n+3)\cdot x^3 +$
$+ 15F(6n+4)\cdot x^2 - 6F(6n+5)\cdot x + F(6n+6)$

has two real zeros, namely $\{ X(n) ,\quad X2(n) \}$,
and four complex, namely $\{ X2(2n) + (\pm(1 - 2X2(2n)) + \mathbf{i}\cdot\sqrt{15})/(2F2(2n) \pm 2) \}$
and their complex conjugates,
wherein $X2(n) \equiv F2(n+1)/F2(n) := 13/8 - F2(n-6)/(8F2(n))$.

As n increases, zeros cluster closer, thus becoming more vulnerable to roundoff.

Like these above, formulas for other polynomials' zeros, some far more complicated, have all been confirmed (laboriously) by an automated algebra system (DERIVE).

Are Polynomials' Floating-Point Coefficients Exactly Right ?

Ideally, corruption by roundoff should be revealed by IEEE 754's INEXACT Flag thus:

Lower INEXACT Flag ;
Compute all of a polynomial's coefficients in floating-point ;
If INEXACT Flag is now Raised, coefficients are corrupted by roundoff.

But most programming languages, like MATLAB, deny access to the INEXACT Flag
though it exists in all hardware conforming fully to IEEE 754.

Without that flag, programmers must turn to …

Tedious Tricks:

E.g., $F(n+1) := F(n) + F(n-1)$ in floating-point ;
If $(F(n+1) - F(n)) - F(n-1) \neq 0$ then $F(n+1)$ has been corrupted;
Else $n <$ { 36 for 24 sig.bits, 73 for 15 sig.dec., 78 for 53 sig.bits, …}.

E.g., $c3 := 15 \cdot f$ in *Binary* floating-point ;
If $(c3 - 16 \cdot f) + f \neq 0$ then c3 has been corrupted.

In Decimal floating-point, most tricks are too much trickier to exhibit here.

Samples of Numerical Results

… from tests of MATLAB's polynomial zero-finder `roots(Coeffs)`.

Its zeros are computed as the *Eigenvalues* of the polynomial's *Companion Matrix*.

The first column shows True zeros; the second shows Errors in `roots`.
Red Italicized Digits are the digits obscured by computational errors.

Example: Results from a test of `roots` on a 6th degree polynomial V(8, x).

MinMaxErr ≈ 0.00593756

```
TrueZ__rootsErr =
 1.61904761904762                                  -0.0059
 1.61854034451496 + 0.00087782940757i              -0.0030 - 0.0051i
 1.61854034451496 - 0.00087782940757i              -0.0030 + 0.0051i
 1.61752717391304 + 0.00087703427224i               0.0030 - 0.0051i
 1.61752717391304 - 0.00087703427224i               0.0030 + 0.0051i
 1.61702127659574                                   0.0059
```

Over 12 of roughly 15 sig.dec. carried by MATLAB's arithmetic have been lost, thus obscuring differences in the third dec., as an experienced Error-Analyst would expect.

Would you have expected that loss? Would you have noticed it?

Example: Results from a test of `roots` on a cubic polynomial $(x + 1)\cdot Q(68, x)$.

MinMaxErr ≈ 0.000000027

```
TrueZ__rootsErr =
  1.6180339887499l                -0.000000027
  1.6180339887498                  0.000000027
 -1.0000000000000                  0
```

A very nearly double zero can be expected to lose about half of the 15 sig.dec. carried by MATLAB's arithmetic, as happens here. My own real cubic solver does better:

Results from a test of my cubic solver on a cubic polynomial $(x + 1)\cdot Q(68, x)$.

MinMaxErr ≈ 0.00000000034

```
TrueZ___qbc3_Err =
  1.61803398874991                -0.00000000034
  1.61803398874988                 0.00000000034
 -1.0000000000000                  0
```

My real cubic solver qbc3.exe runs under DOS 6.22 or pre-2012 Windows on 32-bit Intel x86 architectures and many AMD clones. It retains about 9 sig.dec. instead of about 7 at near-double zeros.

Example: Results from a test of `roots` on a cubic polynomial C(42, x) .

MinMaxErr ≈ 0.00001

```
TrueZ__rootsErr =
  1.6180 3713527852                          -0.00001
  1.61803 241548559 + 0.00000 272496605i      0.0000052 - 0.000009i
  1.61803 241548559 - 0.00000 272496605i      0.0000052 + 0.000009i
```

A very nearly triple zero can be expected to lose about 10 of the 15 sig.dec. carried by MATLAB's arithmetic, as happens here. My own real cubic solver does much better:

Results from a test of my cubic solver on a cubic polynomial C(42, x) .

MinMaxErr ≈ 0.0000000000000003

```
TrueZ___qbc3_Err =
  1.61803713527852                           0.0000…
  1.61803241548559 + 0.00000272496605i       0.0000… + 0.0000…i
  1.61803241548559 - 0.00000272496605i       0.0000… - 0.0000…i
```

My real cubic solver qbc3.exe always finds nearly triple zeros fairly accurately because it evaluates cubics using a recurrence different from Horner's and no slower. A story for another day.

When Do Zeros' Accuracies Matter? A Didactic Example

Given a positive value of $\theta < 1$, but very near 1,
we seek a value of β that Minimizes the Maximum of the magnitudes
of the zeros Z of the quartic polynomial in x, …

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

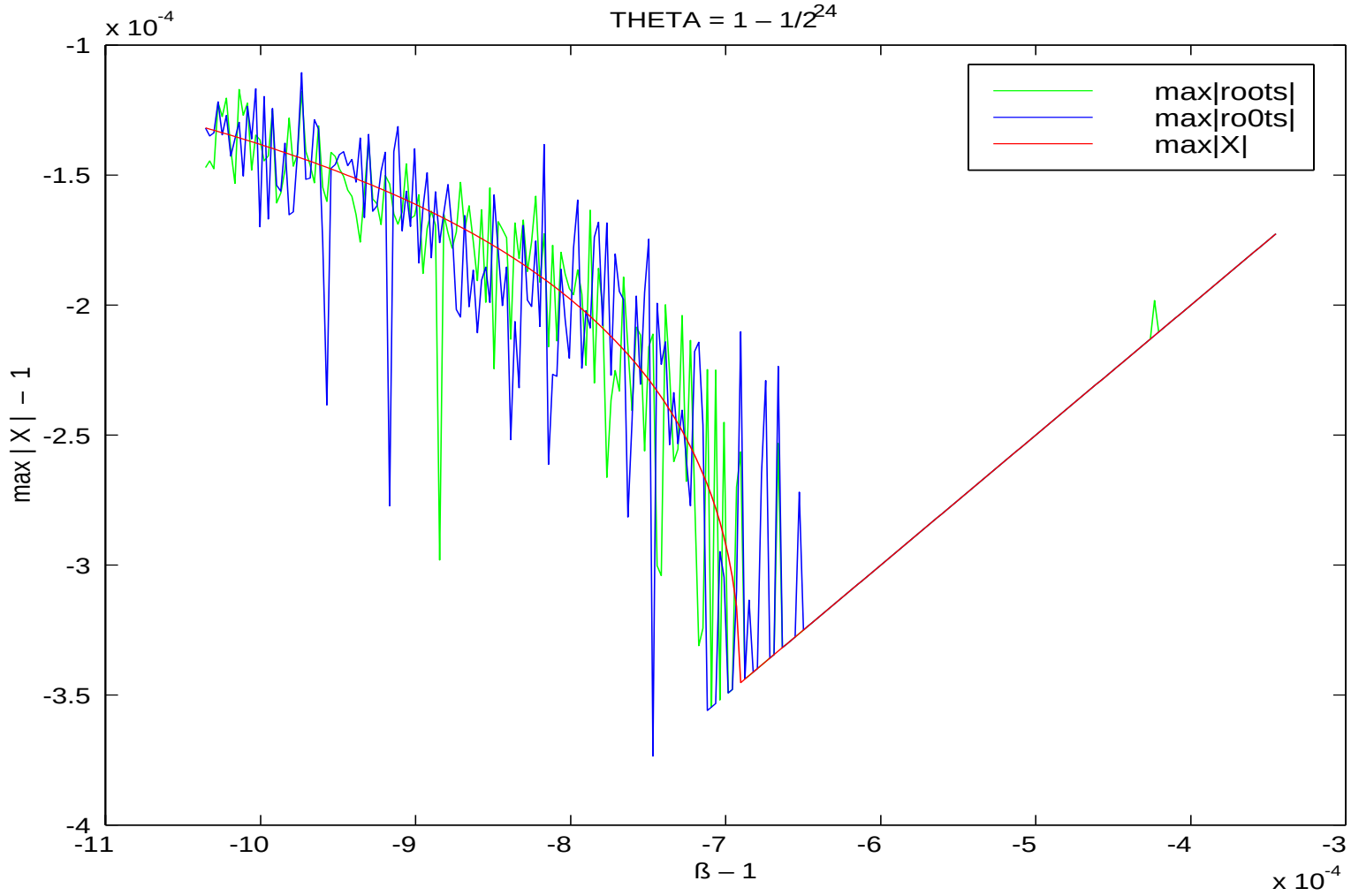
A MATLAB program computed the coefficients of H and fed them to `roots`.
The largest |X| of the computed zeros X was plotted against β
to find β where that Max|X| is minimized.

Also plotted are Max|X| computed by a modified version `ro0ts` of `roots`,
and Max|Z| for the true zeros Z.

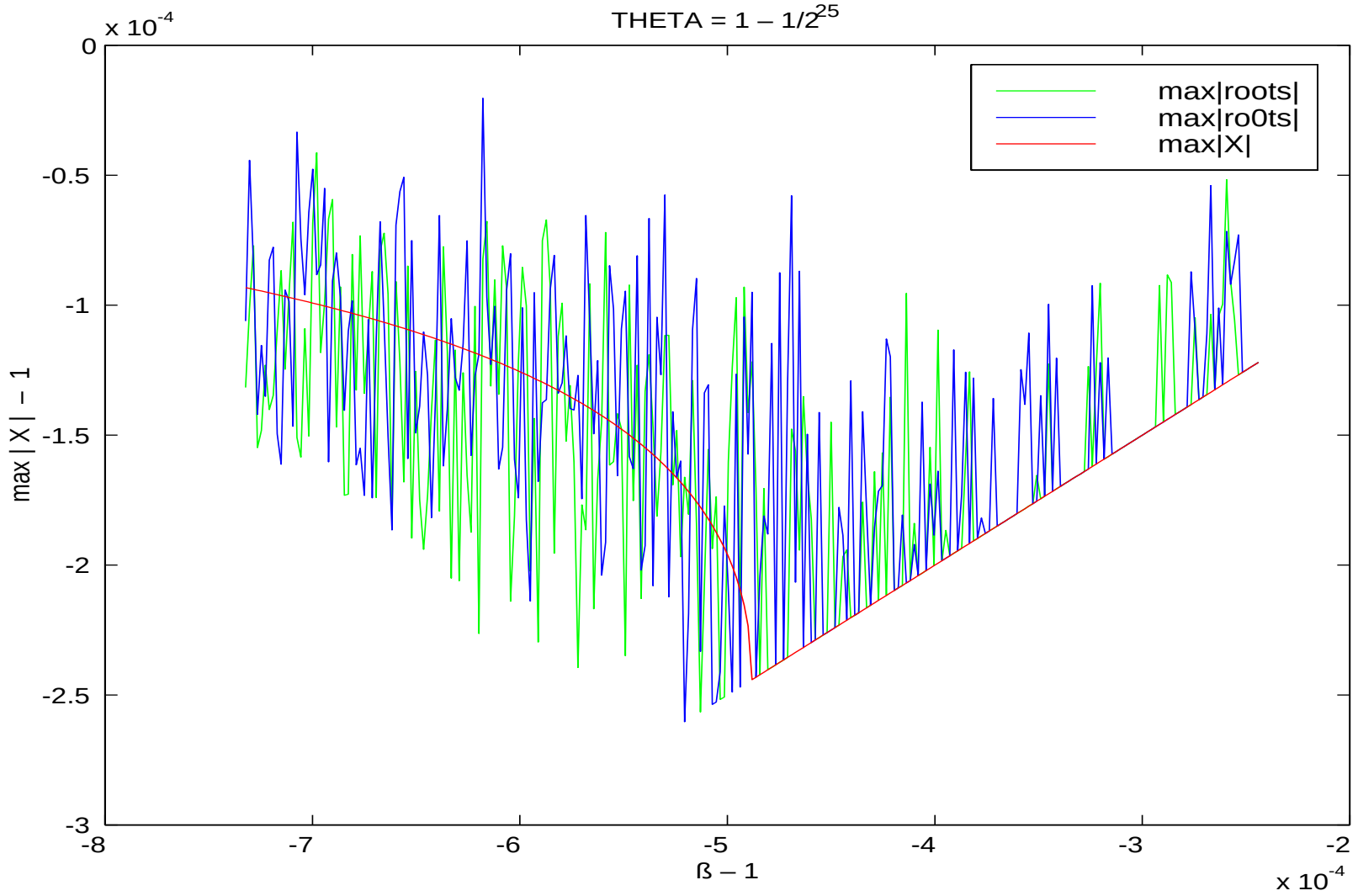
Green `roots'` Max|X| Blue `ro0ts'` Max|X| Red True Max|Z|

To obtain readable numbers at tick-marks on the axes,
we actually plot max|X| – 1 against $\beta - 1$.

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



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



Computed zeros for $\theta = 1 - 1/2^{25} = 0.99999997019768$ and
MiniMaxing $\beta = 0.99951183793383$

True zeros Z(…)	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

MATLAB's arithmetic carries 53 sig.bits (over 15 sig.dec.),
but `roots` and `ro0ts` lose all but the leading three
(though the computed zeros differ in their fourth digits)
thus determining β to at most three sig.dec.

But β is determined sharply by θ :

the minimizing $\beta = \theta^2/(1 + \sqrt{((1-\theta)(1+\theta))})^2$,

Computed values of closely clustered zeros often appear less closely clustered,

Now that we see the numerical results, we see a need for a change of variables:

Replace variables x by $y := 1-x$, θ by $t := 1-\theta$ and β by $b := 1-\beta$ to get

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

Solve "H = 0" for relatively unclustered zeros Y to get accurate zeros $X := 1 - Y$.

Note that the change of variables has been carried out
EXTRA-PRECISELY
(actually infinitely precisely)
using an automated algebra system.

Conclusions Inferred from Test Results

Inaccuracies exposed by tests 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 ?

How shall we discover whether extra-precise arithmetic is necessary to cope with polynomials that differ greatly from the ones tested ?

Compute error-bounds for computed zeros.

Error-Bounds for an Approximate Zero

Given are the n+1 numerical coefficients of a polynomial P of degree n .

We wish to compute one of the zeros z of P . It may be complex.

We have computed an approximation x to z , and

values of P(x) and its derivatives P'(x) and P"(x) are available.

How close is x to z ? Which zero z ?

Error-Bounds for an Approximate Zero

Given are the n+1 numerical coefficients of a polynomial P of degree n .

We wish to compute one of the zeros z of P . It may be complex.

We have computed an approximation x to z , and

values of P(x) and its derivatives P'(x) and P"(x) are available.

How close is x to z ? Which zero z ?

A zero z of P nearest x satisfies these inequalities:

•> Laguerre's: $|x - z| \le n \cdot |P(x)/P'(x)|$ [£]

•> Kahan's: $|x - z| \le n \cdot |P(x)| / \sqrt{\left(|P'(x)|^2 + |(n-1)\cdot P'(x)^2 - n\cdot P(x)\cdot P''(x)| \right)}$ [$\mathcal{K}$]

[$\mathcal{K}$] ≤ [£] always, usually not by much, sometimes by a lot.

But [$\mathcal{K}$] costs more than [£] . Which should be used when?

Error-Bounds for an Approximate Zero

Proved in <www.eecs.berkeley.edu/~wkahan/Math128/PolyZbnd.pdf>

A zero z of P nearest x satisfies these inequalities:

•> Laguerre's: $|x - z| \le n \cdot |P(x)/P'(x)|$ [£]

•> Kahan's: $|x - z| \le n \cdot |P(x)| / \sqrt{\left(|P'(x)|^2 + |(n-1) \cdot P'(x)^2 - n \cdot P(x) \cdot P''(x)| \right)}$ [$\mathcal{K}$]

[$\mathcal{K}$] ≤ [£] always, usually not by much, sometimes by a lot.

But [$\mathcal{K}$] costs more than [£] . Which should be used when?

[£] = ∞ @ zeros of P' not of P ; usually there are n−1 of them.
[$\mathcal{K}$] = ∞ @ Double zeros of P' ; usually there are none of them, at most (n−1)/2 .

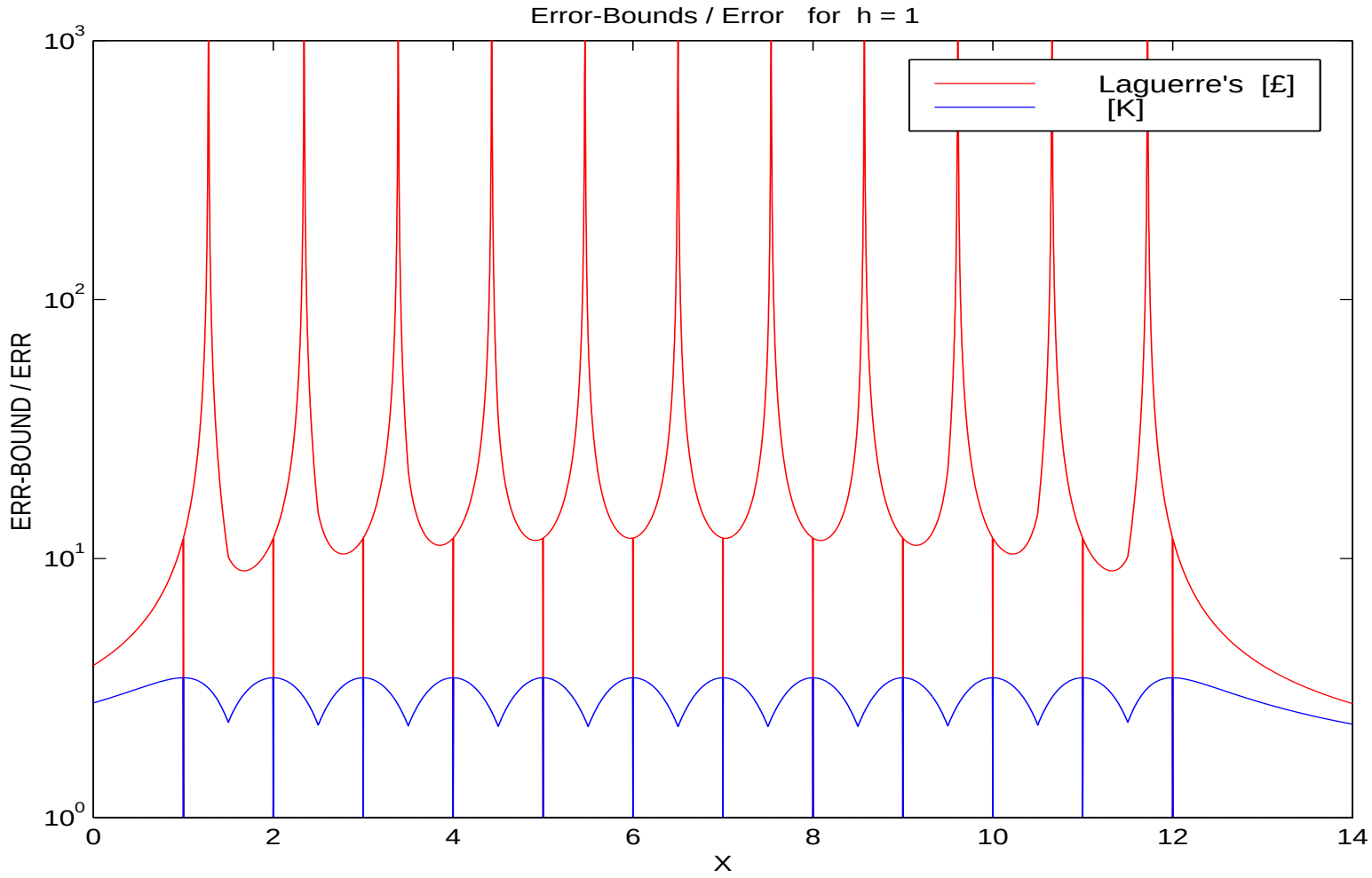
When is [$\mathcal{K}$]/|x−z| << [£]/|x−z| for x near a zero of P ?

It happens when x has fallen amidst a *cluster* of zeros of P .

Alas, a cluster of zeros can be as unobvious as an unmarked minefield.

Example 1: $e(x) := x^{12} - 78x^{11} + 2717x^{10} - \ldots - 1486442880x + 479001600$

Error-Bounds/|Error| for Polynomial $e(x) := \prod_{1 \le k \le 12} (x - k)$



$$e(x) := x^{12} - 78\cdot x^{11} + 2717\cdot x^{10} - 55770\cdot x^9 + 749463\cdot x^8 - 6926634\cdot x^7 + 44990231\cdot x^6 -$$
$$- 206070150\cdot x^5 + 657206836\cdot x^4 - 1414014888\cdot x^3 + 1931559552x\cdot^2 -$$
$$- 1486442880\cdot x + 479001600$$

has zeros $z = 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11$ and 12 .

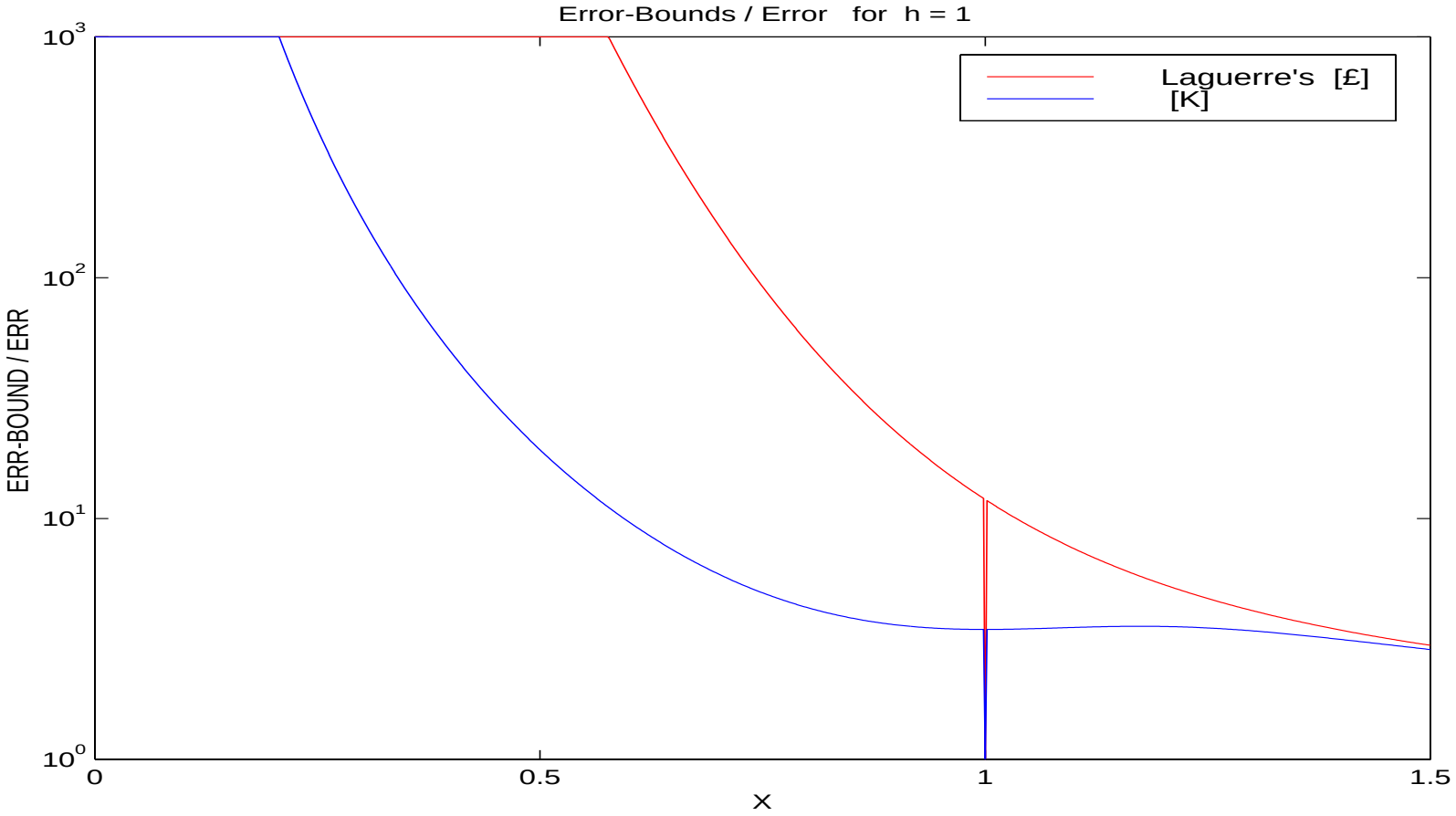
They don't look clustered, but some actually are clustered by the following criterion:

A polynomial has a cluster of zeros if some can be made to coalesce
by a relatively small perturbation of the polynomial's coefficients.

A nearby polynomial $\hat{e}(x) := e(x) - e(-x)\cdot 5.600278/10^{10}$ has coefficients differing from those of $e(x)$ in the tenth sig.dec., and yet this relatively tiny perturbation moves zeros 8 and 9 of e to a double zero of $\hat{e}$ near 8.4835138 . Thus, though adjacent zeros are separated by gaps of 1 , some must be considered *clustered* (or *ill-conditioned*).

On the other hand, adjacent zeros of $b(x) := x^{12} - 1$ are separated by only 0.517638 and yet none are clustered. And this shows up in the behaviors of [£] and [$\mathcal{K}$] . ...

Error-Bounds/|Error| for Polynomial $b(x) := x^{12} - 1$

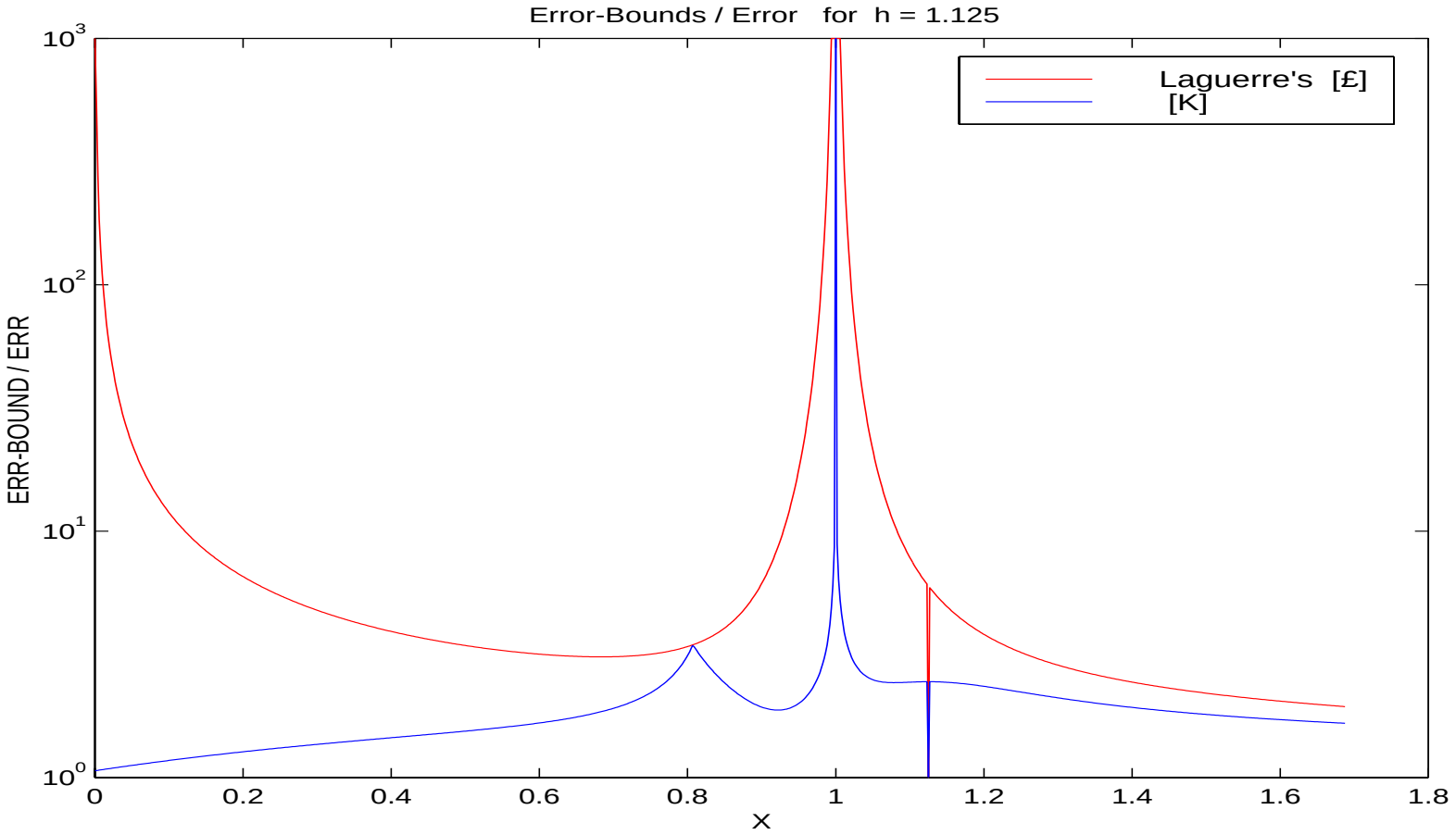


$[£]/|x-z| \approx 12 > [\mathcal{K}]/|x-z| \approx \sqrt{12} \approx 3.4641$ when x is near a zero z of b.

Both $[£]/|x-z|$ and $[\mathcal{K}]/|x-z|$ spike to ∞ as x approaches a multiple zero of b'.
When both spike up to ∞, [£]'s spike is generally much wider than $[\mathcal{K}]$'s.

Example 3:

Error-Bounds/|Error| for $g(x) := (x^2 - h^2)(x^4 + (h^2 - 3)(x^2 + h^2) + 3$ @ $h = 9/8$



Zeros of g : $z = \pm 1.125$ and $z \approx \pm(0.939246 + 0.122459\cdot\mathbf{i}$ and its complex conjugate) .

Derivative g' has a simple zero $\zeta = 0$, and two double zeros $\zeta = \pm 1$ where both [£] and [$\mathcal{K}$] spike up to ∞ . Again, [£]'s spike is far wider than [$\mathcal{K}$]'s .

Summary of Asymptotic Behaviors for Polynomials P of Degree n :

First let z be an n-tuple zero of P ; then $[£] = [\mathcal{K}] = |x-z|$. (Unlikely.)

Next let z be a zero of P of positive multiplicity $m < n$. As $x \to z$,

$$[£]/|x-z| \to n/m \quad > \quad [\mathcal{K}]/|x-z| \to \sqrt{n/m} \ .$$

Next let ζ be a simple zero of P' , but not a zero of P . As $x \to \zeta$,

$$[£] \approx n\cdot|P(\zeta)/P''(\zeta)|/|x-\zeta| \to \infty \quad >> \quad [\mathcal{K}] \to \sqrt{n\cdot|P(\zeta)/P''(\zeta)|} \ .$$

Finally let ζ be a zero of P' of multiplicity $m \geq 2$, but not a zero of P . As $x \to \zeta$,

$$[£] \approx n\cdot m!\cdot|P(\zeta)/P^{[m+1]}(\zeta)|/|x-\zeta|^m \quad >> \quad [\mathcal{K}] \approx \sqrt{\big(\, n\cdot(m-1)!\cdot|P(\zeta)/P^{[m+1]}(\zeta)|\,/|x-\zeta|^{m-1}\,\big)} \ .$$

As both [£] and $[\mathcal{K}]$ spike up to ∞ , $|x-\zeta|\cdot[£]/[\mathcal{K}]^2 \to m$ in the spike.

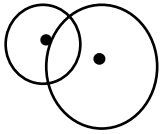
This is why $[\mathcal{K}]$'s spike is so much narrower than [£]'s ,
if $[\mathcal{K}]$ has a spike; and then x is unlikely to fall onto it.

Resolution of Ambiguity

Given two estimates x_1 and x_2 of zeros of P,

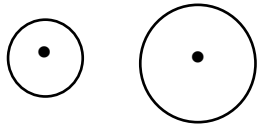
do x_1 and x_2 approximate different zeros, or the same one ?

This question has to be answered by error-bounds:



Ambiguous:
overlapping
error-bounds

Unambiguous:
disjoint
error-bounds



Compute error-bounds $[£]_1$ for x_1 and $[£]_2$ for x_2 .

If $[£]_1 + [£]_2 < |x_1 - x_2|$ then x_1 and x_2 approximate different zeros of P .

Otherwise compute smaller error-bounds $[\mathcal{K}]_1$ and $[\mathcal{K}]_2$.

If $[\mathcal{K}]_1 + [\mathcal{K}]_2 < |x_1 - x_2|$ then x_1 and x_2 are unambiguous.

Otherwise the two approximations are still ambiguous.

If the computed values of P'', P' and especially P are accurate enough, they can be used to improve the estimates x_1 and x_2 using, say, Laguerre's iteration formula.
Occasionally, however, attempts to improve clustered estimates actually worsen them.

Conclusion:

Compute [£] first.

Then, if [£] seems too big,

compute [$\mathcal{K}$] .