Hyperbolic Interpolation and Iteration towards a Zero

Prof. W. Kahan, Math. Dept., and E.E. & Computer Sci. Dept., Univ. of Calif. @ Berkeley

Abstract:

Given a real function $f(x)$ about which we know how to compute its value, we seek one of its *Zeros* z, a root of the equation $f(z) = 0$, starting from some first guess(es). This z should be the limit of a sequence of presumably improving guesses

$$x_{n+1} := Hf(x_n, x_{n-1}, x_{n-2})$$

computed for $n = 0, 1, 2, 3, \ldots$ in turn by an *Hyperbolic Iterating Function* Hf to be defined. It will be compared with a few others, and its application to an eigenproblem will be analyzed in detail. The bigger questions are …

" When is a computed result at least about as accurate
as the data and the arithmetic's precision deserve ?
And how much is that much accuracy worth ? "

Prepared for U.C.B.'s Scientific and Engineering Numerical Computation Seminar, Wed. 9 Sept. 2009,
and posted at <www.cs.berkeley.edu/~wkahan/9Sep09.pdf> .
Mathematical details are posted at <…/Math128/Hyp.pdf> .

Motivation:

Large-scale Floating-Point computations employed in Science and Engineering

have become Undebuggable

when afflicted by occasional and often overlooked anomalies caused by roundoff.

"Overlooked" ? Then how does anyone know they're there ?

Evidence:

Some such bugs in heavily used software get fixed only after many years.

e.g. log, atanh, sinh, asinh, … in early versions of MATLAB

Some such bugs in heavily used software persist over decades.

e.g. in Jacobi-like programs for general symmetric eigenproblems used by some structural engineers since the 1960s.

e.g. Failure Modes in current MATLABs' lu, A\b, and c'/A ; see <www.cs.berkeley.edu/~wkahan/Math128/FailMode.pdf>

Thorough error-analysis of roundoff is too expensive, and consequently

"More honour'd in the breach than the observance". *Hamlet* 1.iv.14

Large-scale Floating-Point computations employed in Science and Engineering
have become Undebuggable
when afflicted by occasional and often overlooked anomalies caused by roundoff.

What could change "Undebuggable" to merely "difficult to debug" ?

- Appropriate tools built into hardware, compilers and debuggers;
  see §14 of <www.cs.berkeley.edu/~wkahan/Mindless.pdf> .

What could change "occasional and often overlooked anomalies" to "practically nonexistent anomalies" ?

- Floating-point arithmetic of extravagantly higher precision than widely deemed adequate, and fast enough to be used ***by default*** , at least initially.

  IEEE Standard 754-2004 includes 16-byte wide Quadruple-Precision, only as an option.
  IBM hardware and Sun, HP and GNU compilers include Quadruple, but too slow for default.

Precisions beyond the 8-byte Double deemed adequate in 1965 get practically no support from Java, MATLAB and other programming languages popular now.

How much extra precision is extravagant enough?
Actually, none!

D.M. Priest's theorems (1990 thesis) say that results from an algorithm using extra-precise arithmetic can always be obtained from an *unnatural* algorithm that uses no extra-precise arithmetic. Instead it may appear to be computing zero in numerous tricky ways.

The tricks may be estimating rounding errors in order to compensate for them,
thus, in effect, simulating extra-precise arithmetic in an ambiance that lacks it.

Priest's theorems do not say how much slower unnatural algorithms must run, nor how long a clever programmer must spend to find a tricky one of them not intolerably slow.

Tricky program(mer)s are the bane of the computing industry.

Let's consider only *natural* algorithms.

One can be derived by a competent analyst to solve a given problem,
and that algorithm would work were precision infinite. But it isn't.

How much more will that algorithm's defense against roundoff cost if …

- extra-precise arithmetic is available in hardware with language support ?
- extra-precise arithmetic is practically unavailable ? … Compare …

To compare costs, we have to know roughly …

How much extra precision is almost surely adequate?

IEEE 754 (2004)'s 16-byte-wide Quadruple Precision is almost surely adequate.
Why ?

Almost all large bodies of scientific and engineering data and intermediate results can be stored in arrays of 8-byte-wide Doubles if not 4-byte-wide Floats.

A rough old rule of thumb with some mathematical justification is that arithmetic precision somewhat exceeding twice both the precision of data and the accuracy sought in results suffices to practically preclude embarrassment by roundoff.

A few of the examples that support this rule of thumb:

- Nearly double roots
- Least squares fits to nearly linearly dependent models
- Trajectories determined mostly by least-effort principles
- Rank-1 updated symmetric eigenproblem

The last example is the topic for today.

Rank-1 updated symmetric eigenproblem

Given is a computed eigensystem of a real symmetric matrix $V = V'$, and a column $\mathbf{e}$ and scalar $\alpha \neq 0$. Desired is the eigensystem of $V + \alpha \cdot \mathbf{e} \cdot \mathbf{e}'$, and sooner than if computed from scratch.

This task is soon reduced to computing the eigensystem of $\Lambda + \alpha \cdot \mathbf{c} \cdot \mathbf{c}'$ wherein $\Lambda = \mathrm{Diag}(\lambda_1, \lambda_2, \ldots, \lambda_K)$ has distinct sorted eigenvalues $\lambda_1 < \lambda_2 < \ldots < \lambda_K$ of V, and row $\mathbf{c}' = [c_1, c_2, \ldots, c_K]$ obtained from $\mathbf{e}'$ has every $c_j \neq 0$. Desired eigenvalues are the K roots ξ_j of the *Spectral* (or *Secular*) *Equation* …

$$f(\xi) := \sum_k c_k^2/(\xi - \lambda_k) - 1/\alpha = 0 .$$

After an eigenvalue ξ has been computed, its eigenvector has direction

$$\mathbf{x} := (\xi \cdot I - \Lambda)^{-1} \cdot \mathbf{c} .$$

At a conference in 1990 I showed why this simple formula for $\mathbf{x}$ would give wretched results occasionally unless ξ were computed in arithmetic somewhat more than twice as precise as the data. My intent was to stimulate demand for extra-precise arithmetic in hardware and supported by programming languages.

Spectral Equation : $f(\xi) := \sum_k c_k^2/(\xi - \lambda_k) - 1/\alpha = 0$, to be solved for ξ .

Eigenvector: $\mathbf{x} := (\xi\cdot I - \Lambda)^{-1}\cdot\mathbf{c}$. Can be very unsatisfactory for innocuous data unless either ξ is computed extra-precisely or …

In 1994, Ming Gu and his thesis advisor Stan Eisenstat published a tricky† algorithm that supplanted that simple formula for $\mathbf{x}$ by another, not much more costly, yet seemingly accurate enough without needing extra-precise arithmetic.

"*Seemingly* accurate enough" ?

Gu and Eisenstat found *Normwise Backward Error-Bounds* for their computed eigensystem, showing that the computed eigenvectors were all orthonormal as nearly as could reasonably be desired, and each eigenvector/value pair were no worse than if Λ and $\mathbf{c}$ had been perturbed by amounts below $K\cdot\varepsilon\cdot\|\Lambda + \alpha\cdot\mathbf{c}\cdot\mathbf{c}'\|$ and $K\cdot\varepsilon\cdot\|\mathbf{c}\|$ respectively, where roundoff threshold $\varepsilon := (1.000\ldots001 - 1)/2$.

These error-bounds were deemed exemplary at the time they were found.

---

† Tricky: Some computers commercially significant then but now long gone required the application of *unoptimized* assignments like $\lambda := (\lambda + \lambda) - \lambda$ to lop off the last bit of every λ_j and ξ_j before use.

Spectral Equation : $f(\xi) := \sum_k c_k^2/(\xi - \lambda_k) - 1/\alpha = 0$, to be solved for ξ .

Eigenvector: $\mathbf{x} := (\xi\cdot I - \Lambda)^{-1}\cdot\mathbf{c}$, obtained from a different tricky formula.

Errors: No worse than if Λ and $\mathbf{c}$ had been perturbed in norm by less than $K\cdot\varepsilon\cdot\|\Lambda + \alpha\cdot\mathbf{c}\cdot\mathbf{c}'\|$ and $K\cdot\varepsilon\cdot\|\mathbf{c}\|$ respectively. These bounds seem tiny at first.

What if the numerical data λ_k, α and c_k , and therefore the eigenvalues ξ_j , have wildly diverse magnitudes? Perturbations allowed by those error-bounds are big enough to overwhelm tiny data-elements and obliterate tiny eigenvalues.

Do tiny eigenvalues and their eigenvectors deserve to be messed up that way?

How sharply do diverse data slightly uncertain (perhaps only in end-figures) determine the desired results? Hard to say. None the less …

We are greedy; we want it all, on the cheap, and soon.

We desire results at least about as accurate as the data and the arithmetic's precision deserve, and about as soon as possible.

We desire results at least about as accurate as the data and the arithmetic's precision deserve, and about as soon as possible.

What does this mean? Is the desire reasonable ?

Eigensystems are most unlikely to be computed for their own sakes. They are intermediate results between partially processed initial data and final results that may well bear directly upon human affairs. Ideally the final results and their variations should depend solely upon the initial data and their variations, not upon by-products of the arithmetic used for the computation.

Roundoff's effects are tolerable only if negligible in the results or at worst small compared with the results' uncertainties inherited from uncertainties in the data. Often the most economical way to inhibit intolerable effects is to carry precision so extravagant that we need never find out whether the results are as accurate as the data deserve. What if that precision is practically unavailable or too slow?

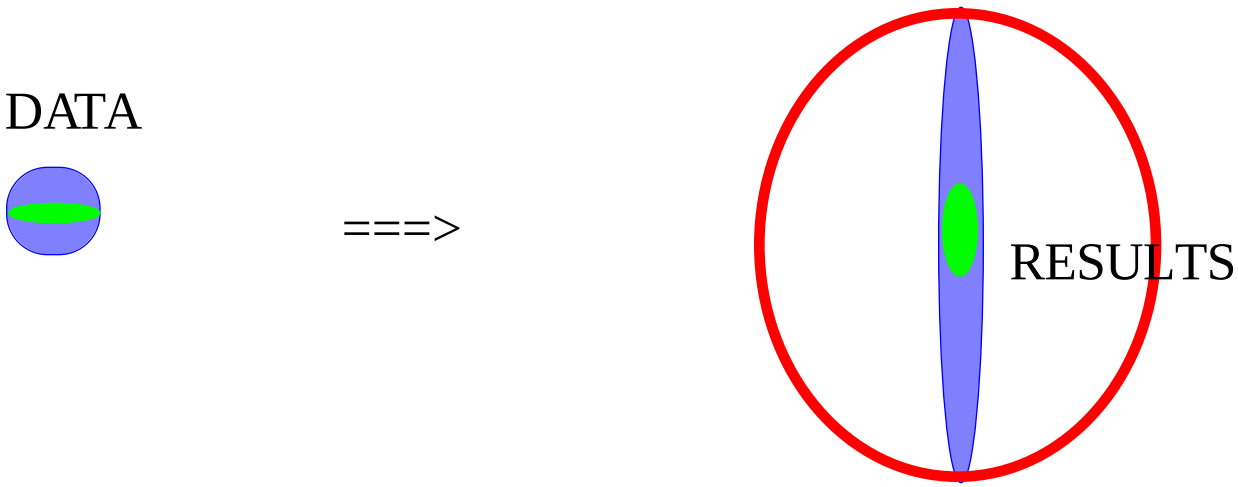
Suppose the arithmetic's precision matches the stored data's.

To estimate the cost of managing, without extra-precise arithmetic, to compute results at least about as accurate as deserved, shouldn't we first gauge roughly

how much accuracy is deserved?

This question turns out to be too difficult to answer in general.

The various norms numerical analysts invoke in Backward Error-Analyses, to explain the accuracies their programs achieve, can be inappropriate to gauge the correlated uncertainties results inherit from uncertainties in their data. The next picture will help to explain why.



Often relatively few parameters of data determine vastly many more elements of a matrix. Uncertainties in the parameters are inherited as correlated uncertainties in the elements and inherited again as correlated uncertainties in the eigensystem of the matrix, and so on.

A numerical analyst's norm used to gauge the data's uncertainty may exaggerate it in some directions by orders of magnitude even if it is as tight as possible. The exaggerated uncertainty is then propagated into the numerical analyst's gauge of the result's uncertainty, which his chosen norm may well exaggerate again.

This is why the accuracy a result "deserves" can be so difficult to assess.

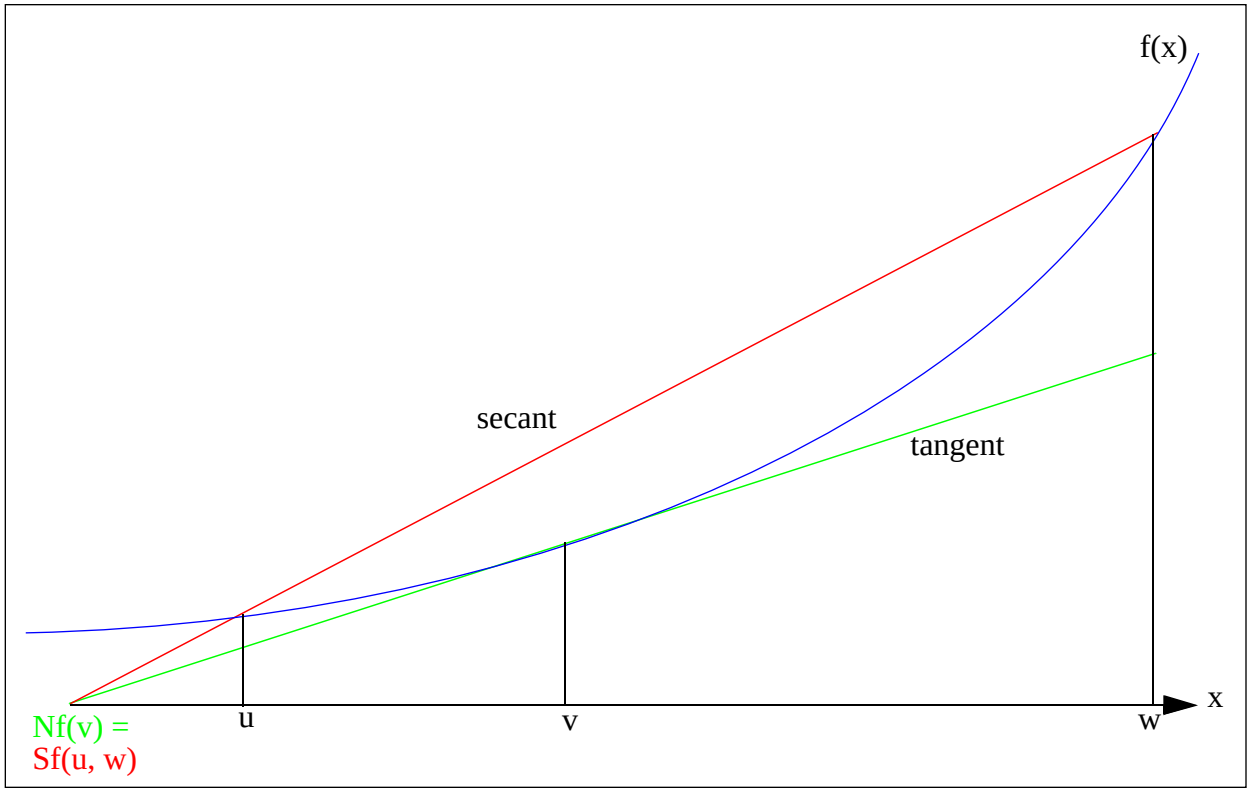
The Spectral Equation: $f(\xi) := \sum_k c_k^2/(\xi - \lambda_k) - 1/\alpha = 0$.

We wish to solve it for K roots ξ_j each at least about as accurately as the data deserve, but without ever knowing how much accuracy is deserved.

We must choose …
- An iterative method: Secant, Newton's, Muller's, Halley's, or …
- Initial guess(es) to start the iteration
- A criterion which stops the iteration when the deserved accuracy is reached

Typically, an iterative method is derived from a *Model*, a family of functions each of which has a zero relatively easy to compute, and from which family a function can be chosen to approximate f closely near (a) recent iterate(s).

For instance, the Model for Secant and Newton's iterations is the family of Linear functions …



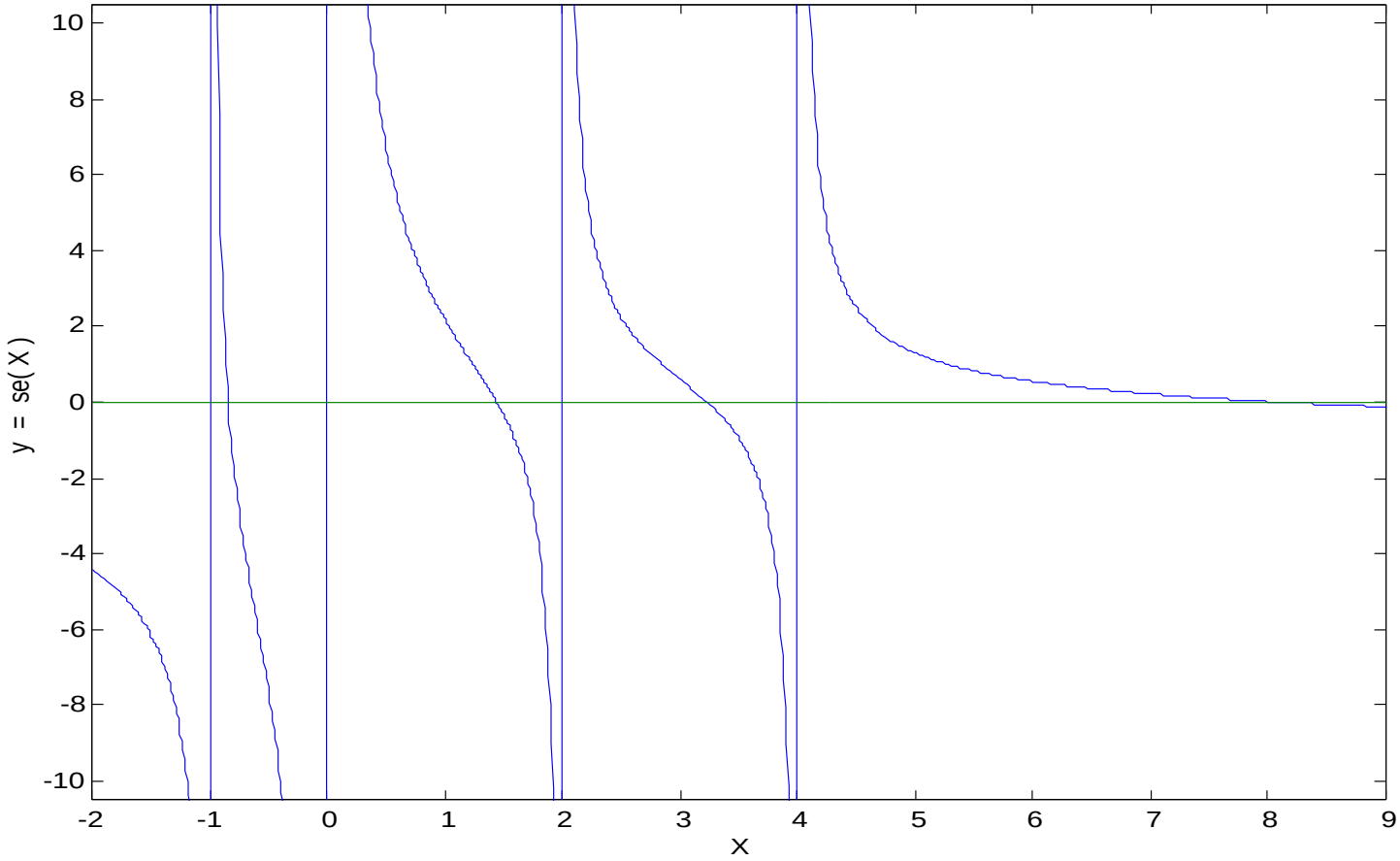
Newton's Iterating Function: $Nf(v) := v - f(v)/f'(v) = Sf(v, v)$.

Secant Iterating Function: $Sf(u, w) := u - f(u)/f^{\dagger}(u, w)$ where

Divided Difference $f^{\dagger}(u, w) := (f(u) - f(w))/(u - w)$; $f^{\dagger}(v, v) := f'(v)$.

These iterating functions are analyzed exhaustively in <www.cs.berkeley.edu/~wkahan/Math128/RealRoots.pdf> .

But the Spectral Equation's f looks very different from the f plotted above.



This graph of a Spectral Equation's $f(x)$ is for data $\Lambda := \mathrm{Diag}([-1, 0, 2, 4])$, $\alpha := 1$ and $\mathbf{c}' := [1, 2, 1, 1]$. An appropriate Model is a family of functions of which each has at least one (finite) pole. The simplest such Model's functions each has one pole and one zero; its graph is an Hyperbola (or straight line).

Hyperbolic Iterating Function $\mathrm{H}f(u, v, w)$:

It is the zero of a Bilinear Rational function $(a\cdot x - b)/(c\cdot x + d)$ whose graph, an hyperbola, cuts the graph of $f(x)$ thrice, namely at $x = u$, $x = v$ and $x = w$.

$$\mathrm{H}f(u, v, w) := u - f(u)/\left(f^{\dagger}(u, v) - f(v)\cdot f^{\dagger\dagger}(u, v, w)/f^{\dagger}(v, w) \right) .$$

The convergence behavior of this iterating function can be inferred from

$$\frac{\mathrm{H}f(w, x, y) - \xi}{(w-\xi)\cdot(x-\xi)\cdot(y-\xi)} = \mathrm{R}f(w, x, y) := \frac{f^{\dagger\dagger}(w, x, y)\cdot f^{\dagger\dagger}(x, y, z) - f^{\dagger}(x, y)\cdot f^{\dagger\dagger\dagger}(w, x, y, \xi)}{f^{\dagger}(w, x)\cdot f^{\dagger}(x, y) - f(x)\cdot f^{\dagger\dagger}(w, x, y)} .$$

This identity is important: ξ is a simple zero of f because $f(\xi) = 0 > f'(\xi)$, so if $w \to \xi$, $x \to \xi$ and $y \to \xi$, then

$$\left(\mathrm{H}f(w,x,y) - \xi \right)/((w-\xi)\cdot(x-\xi)\cdot(y-\xi)) \to \mathrm{R}f(\xi, \xi, \xi) = \left(f''(\xi)^2/4 - f'(\xi)\cdot f'''(\xi)/6 \right)/f'(\xi)^2 ,$$

Consequently, when the Hyperbolic iteration $x_{n+1} := \mathrm{H}f(x_n, x_{n-1}, x_{n-2}) \to \xi$, it converges with *Order* $Ø \approx 1.839$ or faster, which means that, for all n big enough, x_n and ξ agree in over $ß_1 + ß_2\cdot Ø^n$ decimal places for some constants $ß_1$ and $ß_2 > 0$. This iteration's convergence is fast, faster than Newton's **!**

But this Hyperbolic iteration is too vulnerable to roundoff.

$$Hf(u, v, w) := u - f(u)/\big(f^{\dagger}(u, v) - f(v)\cdot f^{\dagger\dagger}(u, v, w)/f^{\dagger}(v, w) \big) .$$

Hyperbolic iteration $x_{n+1} := Hf(x_n, x_{n-1}, x_{n-2})$ is too vulnerable to roundoff because, as iterates converge, computed values of f dwindle. Then roundoff looms relatively larger in these values and gets amplified in divided differences with small divisors, causing subsequent iterates to dither unpredictably.

Confluent Hyperbolic Iteration is much less degraded by roundoff.

The most thoroughly Confluent Hyperbolic Iteration is Halley's Iteration

$$x_{n+1} := Hf(x_n, x_n, x_n) = x_n - f(x_n)/\big(f'(x_n) - \tfrac{1}{2} f(x_n)\cdot f''(x_n)/f'(x_n) \big) .$$

If it converges to a simple zero it does so at least cubically (Order $Ø = 3$) at the cost of computing two derivatives along with f per iteration, and needs only a few correct sig. digits of f' to dither far less widely than Hyperbolic Iteration.

BUT, unless started close enough to a sought zero, Halley's iteration can easily jump past a nearby pole and go elsewhere unless retracted. Every retracted iteration is a wasted iteration and, like ants at a picnic, they come in convoys.

The Confluent Hyperbolic Iterating Function

$$Hf(y, x,x) = Hf(x,x, y) := x - f(x)/\big(f'(x) - f(x)\cdot f^{\dagger\dagger}(x,x,y)/f^{\dagger}(x,y) \big)$$

maintains a *Straddle*.

A Straddle is a pair of arguments u and v between which lies one zero ξ but no pole of the Spectral Equation's f. Because all three of $f'(u)$, $f^{\dagger}(u, v)$ and $f'(v)$ have the same negative sign, both of $Hf(u,u, v)$ and $Hf(u, v,v)$ also lie between u and v. Moreover, because $Rf(u,u, v) < 0$ and $Rf(u, v,v) < 0$ (it's not obvious), $Hf(u,u, v)$ lies between u and ξ, and $Hf(u, v,v)$ between v and ξ. Thus each of these iterating functions maintains and shrinks a Straddle.

Why is a Straddle worth maintaining?

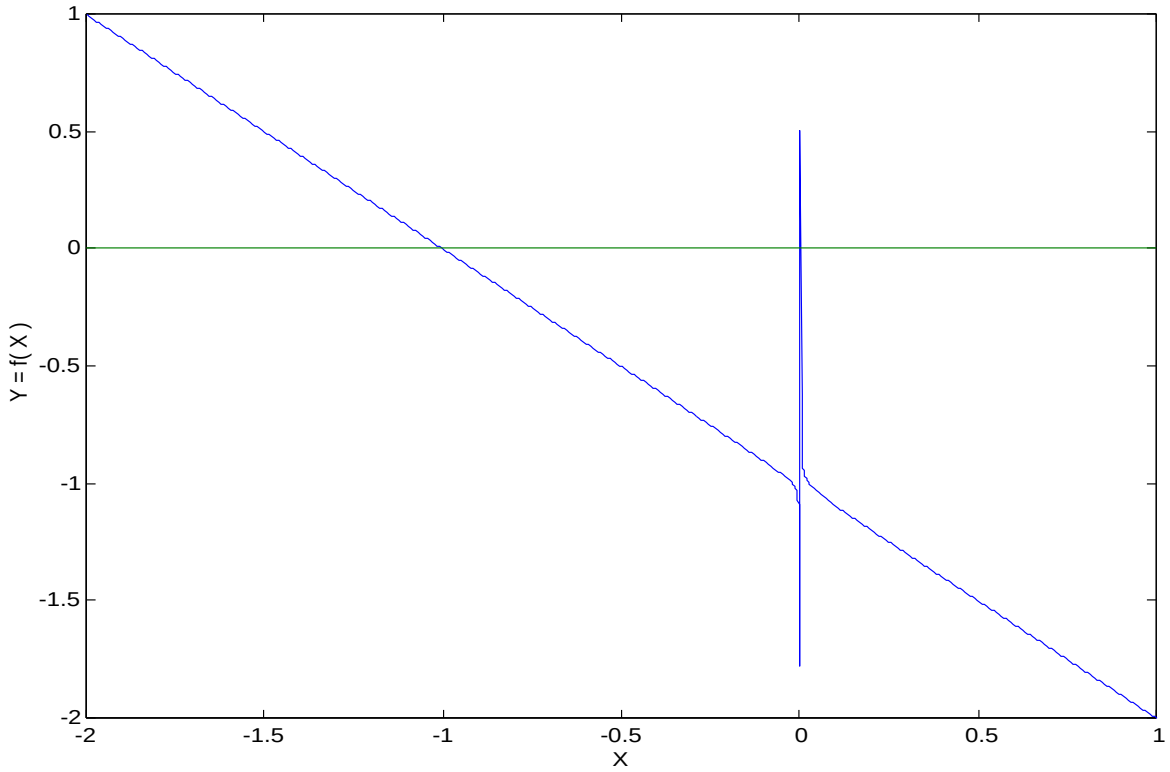
Without a Straddle, convergence from only one side can enter a regime of tiny steps difficult to diagnose: Are the steps tiny because iterates have come about as close to the sought zero as they ever will? Or are the steps tiny because iterates have entered a regime of very slow convergence to a sought zero not yet close?

What causes very slow convergence? There is more than one cause. One is …

Hidden Zeros

A zero is Hidden by a pole when the two are so close that they nearly cancel. Here is an example:

$$f(x) = (\xi - x)\cdot(1 + 1/x)\,, \quad \xi = 0.0005$$



Confluent Hyperbolic Iteration converges to an isolated Hidden zero slowly at first from one side, quickly from the other, provide the zero has been straddled. Convergence to a zero Hidden in a cluster can start slowly from both sides.

How is the First Straddle found ?

The Spectral Equation

$$f(\xi) := \sum_k c_k^2/(\xi - \lambda_k) - 1/\alpha = 0$$

with K terms in the sum is approximated by a simplified Spectral Equation with only two terms in the sum obtained by moving distant poles to or away from the pole(s) immediately next to the sought zero ξ. Poles beyond the neighbor on one side of ξ are moved onto this neighbor; poles beyond the neighbor on the other side of ξ are moved away to ∞. The two two-term equations' roots can be proved to straddle ξ. And they cost little to compute.

For the mathematical details see <…/Hyp.pdf>.

Bi-Confluent Hyperbolic Iteration:

Given that u and v straddle a zero but no pole of f, one iteration-step replaces u by H(u,u, v) and v by H(u, v,v). Convergence is ultimately cubic through a nested sequence of Straddles that shrink onto the zero ξ. Convergence to an *isolated* Hidden zero appears to be initially quadratic, which is fast enough.

When should Bi-Confluent Hyperbolic Iteration be Stopped?

I don't know yet. The choice of a stopping criterion would be far easier if extra-precise arithmetic, albeit slow, were available for use in the last few iterations.

Stopping when roundoff first causes the iterates' nesting to fail may be stopping too soon; I have not yet become satisfied that this simple criterion produces results at least about as accurate as the data deserve. How should it be tested?

If a stopping criterion stops iterating several iterations beyond the first failure of nesting, how many iterations will be wasted?

Other stopping criteria are under consideration …

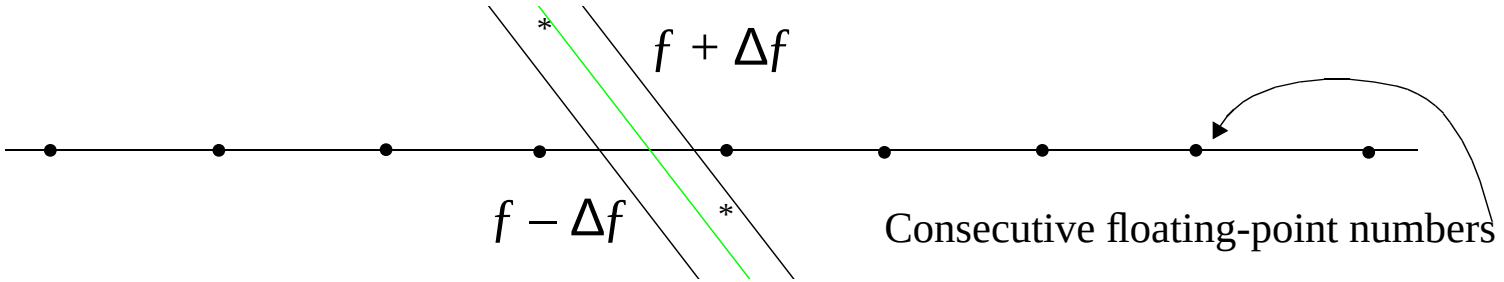
Stopping when the computed f is probably largely junk

The computed value of $f(\xi) := \sum_k c_k^2/(\xi - \lambda_k) - 1/\alpha$ accumulates roundoff not exceeding $\Delta f(\xi) := æ\cdot\sum_k c_k^2/|\xi - \lambda_k|$ in which $æ$ is some modest multiple of ε, the roundoff threshold, that depends upon dimension K and programming details. $æ := 2K\cdot\varepsilon$ usually produces a valid upper bound Δf for roundoff; and putting $æ := 2\sqrt{K}\cdot\varepsilon$ produces an estimate Δf almost never exceeded. Δf can be computed quickly from quantities already computed during the iterations.

It seems reasonable to stop iterating as soon as an iterate ξ satisfies …

$$|\text{computed } f(\xi)| \le \Delta f(\xi) + \varepsilon\cdot|\xi|\cdot|f'(\xi)| \,.$$

The second term on the right ensures that the inequality can be satisfied by at least one floating-point number ξ. Otherwise, without that term, the estimate Δf may be so tight that no floating-point number ξ satisfies the inequality …



I would prefer to find an efficient stopping criterion that never computes Δf .

How can the chosen criterion's effectiveness be tested other than by comparing the final $|\text{computed } f(\xi)|$ with $\Delta f(\xi)$? Is there a better way? I hope so.

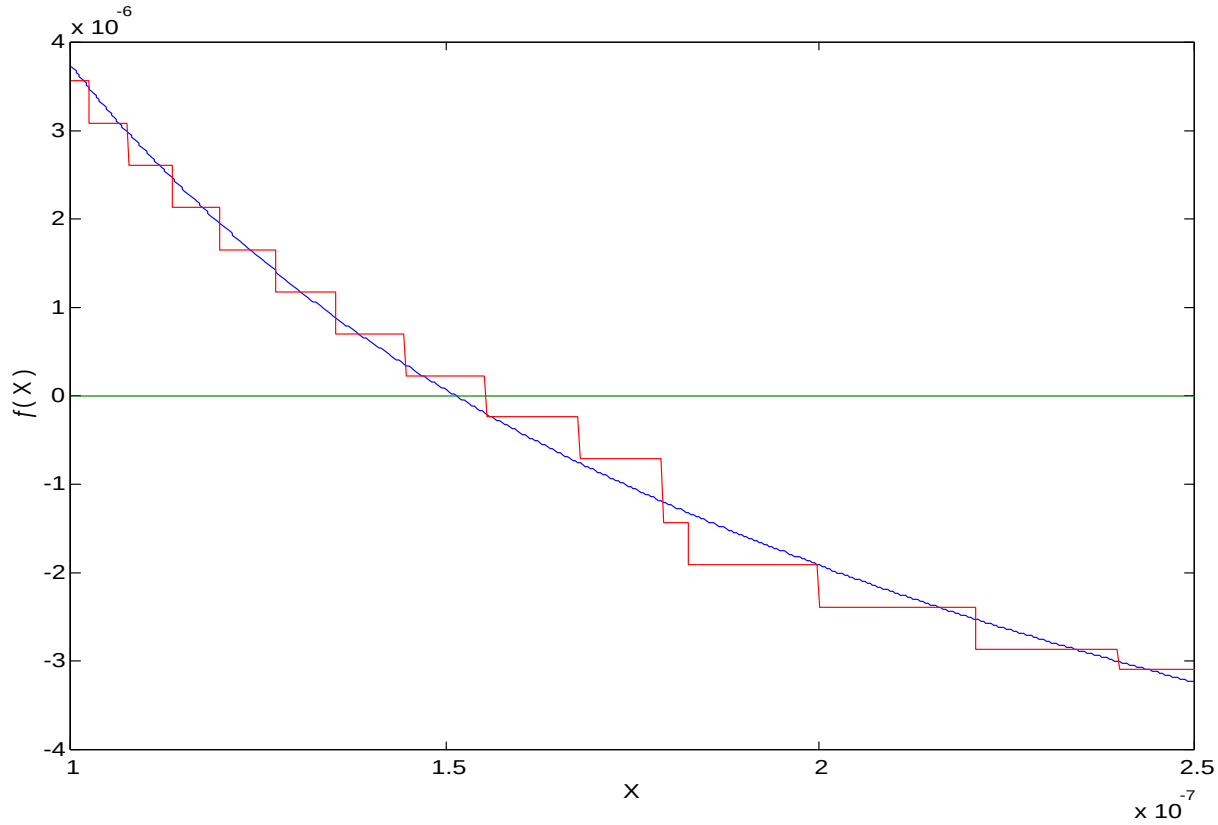
The trouble with the stopping criterion

$$|\text{computed } f(\xi)| \le \Delta f(\xi) + \varepsilon\cdot|\xi|\cdot|f'(\xi)|$$

is that it tends to gross pessimism when Δf is an always-valid upper bound for the accumulation of roundoff. Pessimism is due to rounding errors' resemblance, on computers that conform to IEEE Standard 754 for floating-point arithmetic, to independent random variables with mean zero in so far as they tend to cancel partially as they accumulate, the more so as dimension K increases.

But roundoff is not random on those computers.

In fact computed values of the Spectral Equation's $f(x)$ turn out to be monotone non-increasing between poles. Below is a snapshot comparing values of a typical Spectral Equation's $f(x)$ computed with 4-byte floating-point versus 8-byte:



This staircase graph is typical, though the sizes of horizontal steps and vertical risers are often far more diverse. Were roundoff random or biased, the graph of f would appear ragged or shifted, and its zeros ξ_k would need a trickier program and noticeably longer to be computed "at least about as accurately as the data and the arithmetic's precision deserve". Still, this much accuracy may be achievable only at the cost of several concluding iterations of *Binary Chop*, — too slow.