Finding Every Root of a Broad Class of Real Continuous Functions in a Given Interval

Robert C. Tausworthe*

This article presents a mathematical root finder capable of finding all roots of an arbitrary continuous function within a given interval subject to very lenient parameterized assumptions, which are (1) that adjacent roots are separated at least by a given amount, `xGuard`; (2) any point whose function value is less than εf in magnitude is considered to be a root; (3) function values at distances `xGuard` away from a root are larger than εf, unless there is another root located in this vicinity; (4) a root is considered found if, during iteration, two root candidates differ by less than a prespecified εx; and (5) that the optimum cubic polynomial matching the function at the end and two internal points, and that is within a relative error fraction εL at its midpoint, is reliable in indicating whether the function has extrema within the interval. The robustness of this method depends solely on choosing these four parameters that control the search. The roots of discontinuous functions were also found, but at degraded performance.

I. Introduction

One of the most pervasive needs within Deep Space Network (DSN) Metric Prediction Generator (MPG) view period event generation is that of finding solutions to given occurrence conditions. While the general form of an equation expresses equivalence between its left-hand and right-hand expressions, the traditional treatment of the subject subtracts the two sides, leaving an expression of the form $f(x) = 0$. Values of the independent variable x satisfying this condition are *roots*, or *solutions*. Generally speaking, there may be no solutions, a unique solution, multiple solutions, or a continuum of solutions to any given equation.

In particular, all view period events are modeled as zero crossings of various metrics. For example, the time at which the elevation of a spacecraft reaches its maximum value, as viewed from a Deep Space Station (DSS), is found by locating that point at which the derivative of the elevation function becomes zero. Moreover, each event type may have several occurrences within a given time interval of interest. For example, a spacecraft in a low Moon orbit will experience several possible occultations per day, each of which must be located in time.

* SGT, Inc.; consultant, DSN Planning and Execution Software Systems Section, under a contract to Raytheon, Inc.

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Government sponsorship acknowledged.

The MPG is charged with finding *all* specified event occurrences that take place within a given time interval (e.g., a "pass"), without any special clues from operators as to when they may occur, for the entire spectrum of missions undertaken by the DSN. For each event type, the event metric function is a known form that can be computed for any instant within the interval.

The MPG is required to operate in unattended, noninteractive mode for each of its many applications; it is required to work for all targets submitted to it; it is required to produce all predictions requested of it; it is required to predict an event when there will be one, and not to report one when none will occur; and it is required to produce predictions within specified accuracy.

Computational resource requirements in these applications are usually proportional to the number of times a given function is evaluated, so the number of samples required to predict an event is an important design consideration. Efficiency, together with robustness and accuracy requirements that are placed on the MPG, make stringent demands on the methods that may be used. However, as competing design considerations, robustness is considered more important than an algorithm's complexity and time consumption. Consequently, brute force methods are acceptable when more sophisticated methods are absent and when their use is necessary to meet schedules, provided there is enough computational power available to meet efficiency requirements.

The MPG has developed sophisticated and robust polynomial curve-fitting algorithms to extract and reproduce the dynamic characteristics of many prediction data types, such as uplink and downlink frequencies and antenna pointing angles. Because this capability was available, and because implementation resources were scarce, the curve-fitting object was adapted, when needed, to serve view period event detection as well. This approach provided the needed robustness, but resulted in somewhat unnatural and awkward event-detection designs that have since proved noticeably time-consuming in DSN operations.

In order to determine the feasibility of developing a more natural method of event detection and to quantify its robustness and efficiency, the author subsequently has investigated methods that apply single-root-finder techniques to the search for multiple roots over the time intervals of interest. As many textbooks in numerical analysis warn the reader, as experience bears out, and as literature searches verify, currently (1) there are no foolproof methods for finding single roots of any multiplicity; and (2) there are no rigorous methods to conduct multiple[1] root searches.

The usual advice in textbooks that is given to one seeking to solve an equation is to analyze the object function carefully to gain insight, and then to use this insight in choosing the right methods to apply. Unfortunately, this advice is not practical in MPG applications; but it can be useful, to a limited extent, in algorithm design. The general character of a metric function in an application may be known, but the particulars may vary widely over a considerable dynamic range.

---

[1] Multiple roots in the MPG context means the occurrence of events at different instants in time. The multiplicity of roots is not deemed significant.

The goal of the study was thus to develop an algorithm that, while not being completely foolproof, is nevertheless efficient and robust over a class of functions wide enough to encompass those found in view period event detection, that adapts its operation to the dynamical character of each function subjected to it, and that is still more efficient than attained using the MPG curve-fitting object.

The methods (there are two) developed in the study (one of which is reported here) seem to provide efficient and robust operations, as judged by simulated MPG applications. By design, they work best on functions that are continuous. They both appear to perform reliably, but at somewhat degraded efficiency, when applied to discontinuous functions.

In keeping with the goal of the study, the algorithm that is reported on in this article is named[2] `EveryRoot`. This method iteratively samples the given object function at various locations across the interval. The other method developed, bearing the name `EveryDvRoot`, operates in a similar fashion on samples of the function and its first derivative. Although it is not further described here, its implementation in *Mathematica* form may be found among the MPG *Mathematica* Archives, available via the Service Preparation Subsystem (SPS) portal.[3]

II. Accuracy and Object Function Criteria

Because the MPG is required to find roots of so many differently varying functions, it appears that not any of the now-known root finding methods can be trusted to work unattended for all applications. Even for one given event type, such as finding the time of maximum elevation as described above, the widely varying characteristics of targets over the entire set of DSN missions make the robustness of using any one method problematic. In order to make root finding a more robust, reliable, and accurate process, it is necessary to set forth practical parameterized criteria for the kinds of functions that are found in MPG event metrics and for processing techniques that can yield required event detection accuracies. The following hypotheses were thus set forth.

(1) Functions of interest are presumed to be real-valued and continuous over the given search interval. If an actual object function is piecewise continuous, then the root finder will presumably be applied separately to each continuous segment. Points at which discontinuities occur are presumed known.[4]

(2) These functions may not have distinct roots spaced more closely than some specified tolerance. In particular, if a root is found to exist at $x = r$, then no other distinct root is presumed to lie within the interval $[r - \texttt{xGuard}, r + \texttt{xGuard}]$ for a prespecified `xGuard` value. This guard is meant to ensure (see criterion 4, below) that a just-found root can be isolated so as not to be found again in later actions.

(3) A root is deemed to exist at a value $x = r$ if the function magnitude at this point is less than some small number, i.e., $|f(r)| \leq \varepsilon f$ for prespecified εf.

---

[2] A similar function named `AllRoots` is described in the literature, but it finds all roots of a polynomial.

[3] The portal URL is https://spsweb.fltops.jpl.nasa.gov/portalappsops/Main.do?ft=spa. This is an internal JPL website and a user name and password are required for access.

[4] If points of discontinuity are not known *a priori*, then performance may not be as reliable or efficient in finding roots in the vicinity of these points.

(4) It is presumed that the value of `xGuard` above can be chosen to be large enough that the function values at root-guarded distances are not mistaken for zeros (i.e., that $|f(r \pm \texttt{xGuard}| > \varepsilon f$ unless there actually is a root here), and yet small enough that no root is likely to exist within the guarded interval.

(5) It is presumed that if successive root estimates x_i and x_{i+1} are such that $|x_{i+1} - x_i| < \varepsilon x$ for given εx, then the root-finding process has converged.

(6) It is presumed that if an interpolating polynomial is made to fit the object function across a given subinterval within a specified relative tolerance εP, then reliable judgments of whether the function exhibits monotone or convex behavior within that subinterval can be made via examination of the polynomial.

The physical events represented by the zeros of metric functions in the MPG seem to adhere to these criteria. The numeric guards and tolerances above provide the means for adaptation and accommodation to physical and mathematical properties of the events.

III. Single Root Searches

Except in polynomial problems of degree less than four, root finding invariably requires an iterative approach. There are three basic strategies, here called *bounding*, *polishing*, and *search*. Typical root finders discussed in numerical analysis texts or available in subroutine libraries find only a single root when invoked. They also may fail to find a root, even when one exists.

Root bounding first determines an interval within which at least one root assuredly exists, and then successively narrows the interval, still maintaining at least one root within the interval, until an estimated root value satisfies convergence criteria. A root is said to be *bracketed* in the interval $[a, b]$ if the function is continuous in this interval and $f(a)f(b) < 0$. The intermediate value theorem then guarantees, because $f(a)$ and $f(b)$ have different signs, that the function value zero must be attained at some value of the variable within the interval. Bisection, Regula Falsi, Weijngaarten-Dekker-Brent, and Ridders methods, discussed in the References, are all methods that converge on a bracketed root.

Root polishing begins with an initial estimate of where the root might lie, and iteratively seeks successively improved estimates until convergence criteria are met. The Newton-Raphson, secant, and Taylor-series reversion methods, also discussed in the References, are classic root polishers. Having an appropriate initial estimate is key to the success of the method. The method of determining this value is typically a heuristic based on some knowledge of the problem.

Exhaustive search segments the search interval into small enough pieces that each may be presumed to contain no more than a single root (by way of special knowledge of the character of the problem). Each of these is examined separately for the presence of a root using bracketing, if it applies, or root polishing otherwise. This is essentially the method used by the MPG's predecessor, the Network Support Subsystem Metric Prediction software. *Adaptive*

search is a specialization of the exhaustive search that subdivides each subinterval only to the point necessary to determine the presence or absence of a root.

The more difficult and uncertain case of root finding therefore occurs when the bounding subinterval contains an even number of roots. The function values at the interval endpoints in this case are nonzero and have the same algebraic sign, as signified by $f(a)f(b) > 0$. The facts of this situation are:

(1) A continuous function that varies monotonically from the value at one endpoint to that at the other has no roots in the interval.

(2) A function that is convex with respect to zero between the endpoints again has no roots.

(3) A function that is concave with respect to zero over the interval also has no roots unless (a) an extremum within the interval itself is a root (of even multiplicity), or (b) the extremum value is opposite in sign to that of the endpoints. In the latter case, the extremum brackets roots in the subintervals on either side.

(4) If there are multiple extrema within the interval, the one most in the direction of zero (with respect to endpoints) nearest zero may be examined as in (3) above.

However, the process of locating extrema of an arbitrary function can prove time-consuming, especially if derivative information is absent. An alternative is to develop criteria that indicate, "with high reliability," whether extrema are, or are not, present within a given interval.

One such method is the generation of a Lagrange interpolation polynomial fitting the interval. The Appendix contains the details of an optimized cubic polynomial fit to the two endpoints and values at the 0.293 and 0.707 points of the interval.

An interpolated value at a point x_i, denoted $\hat{f}_i$, may be compared with the actual function value f_i at that point. If they do not agree within a "reasonable" deviation, then the polynomial may be deemed not to fit the function well enough to attest the existence and approximate the location of extrema. Reasonable, in this case, can be quantified to require that

$$\frac{\left| f_i - \hat{f}_i \right|}{\max\left(\left\{ \left| f_0 \right|, \left| f_a \right|, \left| f_i \right|, \left| f_b \right|, \left| f_1 \right| \right\}\right)} < \varepsilon L \tag{1}$$

for a prespecified allowable deviation[5] εL. The comparison points are chosen to be those where the optimum cubic interpolation formula nominally exhibits the greatest deviation from the function, which are 0.117, 0.5, and 0.883 (see Appendix for details). Obviously, as εL is made smaller, then the subintervals that qualify also become smaller, but more trustworthy approximations result.

[5] The polynomial relative accuracy previously designated as εP is now recast as εL, where L designates the allowed error in Lagrange interpolation. In the `EveryDvRoot` algorithm, this quantity is denoted εH, to signify the allowed error in Hermite interpolation.

If the tolerance criteria are not met, the search interval can be subdivided into subintervals, and each analyzed in turn. A natural point for subdivision is the midpoint, since the function value has already been calculated here. This process can continue, then, until the accuracy of the fit is deemed "trustworthy" for indicating the presence of extrema. The judgments listed earlier can then be made as to the presence of a root in the subinterval.

If the polynomial fit has an extremum whose value is beyond zero, then the function can be evaluated at this point to ascertain whether it truly brackets roots on either side. If it does, a bracketing root finder may be used to extract the roots.

If the extremum or function value at this point is "reasonably close" to zero, then a polishing root finder can be called upon to seek a root in this vicinity. Suggested initial points include x_0, x_a, x_b, x_e, x_m and x_1. If one trial location fails, others may be tried. If no roots are found, then none may be presumed to exist in the interval.

The "reasonably close" condition in this case means that the extremum value lies within an uncertainty range about zero due to the allowed error in the Lagrange fit. The fit error εL may be degraded by a factor, here designated as `ERZeroPad`, so as not to reject an interval that might possess a root that a root polisher could find. A padding factor of two or three should prove sufficient. Larger pads elicit needless searches under normal circumstances.

The conditions under which a root can be missed in the process described above are:

(1) The curve-fitting error computation mistakenly reports the cubic polynomial is trustworthy when it is not, and the interval is later rejected. This can be controlled, to a large extent, by choice of εL.

(2) An interval was discarded upon failing the "reasonably close" test, when, in fact, a root was present. This can also be controlled by choice of εL and the zero pad threshold.

(3) The polishing root finder could not locate the root. Little can be done when this happens.

(4) The value chosen for `xGuard` on the basis of having $|f(r \pm \texttt{xGuard}| > \varepsilon f$ may be larger than the actual minimum distance between roots. Reducing εf may help in this situation. If it does not, the function does not meet the assumed criteria on which the algorithm is predicated to operate.

The conditions under which a root may be in error or falsely reported are:

(1) The value of εf may be too large.

(2) The value of εx may be too large.

The values of these two latter parameters are generally chosen based on the numerical precision that can be attained when computing $f(x)$ or its roots, and on the root precision required.

IV. The `EveryRoot` Method

Given an interval $[a, b]$ and a function $f(x)$, the object is to locate values r such that $f(r) = 0$, and to find *all* such values within the given interval. Root multiplicity is unimportant in MPG applications. The `EveryRoot` algorithm described below applies the strategies described above in finding an isolated root, but continues to search for others until it is reasonably assured that all in the given interval have been found. Having just found a root r, the method seeks to find additional roots in the subintervals $[a, r - \texttt{xGuard}]$ and $[r + \texttt{xGuard}, b]$, where guards have been erected on both sides of the extracted root to assure that the just-found root is not found again in later actions.

Inputs include (1) the function $f(x)$ whose roots are to be found; (2) the interval $[x_A, x_B]$ to be searched; (3) the `xGuard` value; (4) the root tolerance εx; (5) the function zero tolerance εf; (5) the polynomial fit accuracy εL; (6) the maximum number of iterations `maxIterations` to be used in bracketing and root polishing; and (7) the maximum number of roots `maxRoots` to be found.

The method maintains two data structures: the root list and the unsearched interval list. At any stage of the search, the first of these contains the list of roots found so far, while the other contains a list of packets, each containing the bounds of intervals that are yet to be processed, together with the function values at these boundaries. For discussion purposes, if subinterval bounds are $[x_0, x_1]$, then the corresponding packet is designated $\{x_0, x_1, f_0, f_1\}$. If the function values have not yet been evaluated, denote the packet as $\{x_0, x_1, -, -\}$.

Initially, the root list is empty and the unsearched interval list contains the single packet spanning the input search interval. The processing steps are numbered below for reference.

(1) Create an initial packet $\{x_A, x_B, -, -\}$ and insert it into the unsearched interval list.

(2) If the unprocessed interval list is empty or if the length of the root list is `maxRoots`, no further action is required, so proceed to step 19.

(3) Retrieve a packet $\{x_0, x_1, f_0, f_1\}$ from the unsearched interval list and examine it to determine what action is to be taken next.

(4) If the length of the interval is less than `xGuard`, discard it and continue the process back at step 2.

(5) If one or more function values in the packet are as yet unevaluated, calculate function values to complete the interval packet.

(6) If the magnitude of the function value at one of the subinterval endpoints is greater than εf, then proceed to step 8.

(7) Here, one of the endpoints is deemed to be a root, so insert it into the root list. Shorten the subinterval by `xGuard` at the appropriate end (the least shortened interval size permitted is zero). Modified endpoints are not permitted to lie outside the current interval. Indicate the function value at this point as unevaluated ("–"). Insert the modified packet back into the unsearched interval list, and continue the process back at step 2.

(8) At this point, endpoint function values are nonzero. If $f_0 f_1 > 0$, then the interval does not bracket a root, so proceed to step 11.

(9) Use one of the bracketing root finders (e.g., Brent) to locate a root, say at $x = r$. Limit the number of iterations that may be used to `maxIterations`.

(10) Insert the just-found root r into the root list, create packets $\{x_0, r - \texttt{xGuard}, f_0, -\}$ and $\{r + \texttt{xGuard}, x_1, -, f_1\}$ for each of the subintervals in the manner of step 7 above, insert each of these into the unsearched interval list, and proceed back to step 2.

(11) At this point, the interval endpoints are of the same sign. Compute the so-called ridge error values, or differences between the function and the Lagrange interpolation polynomial at its likely maximum error points (see the Appendix for particulars), normalized as in Equation (1). If all three ridge errors satisfy the εL criterion, designate $f_{\max} = \max\left(\{|f_0|, |f_a|, |f_i|, |f_b|, |f_1|\}\right)$ and proceed to step 13.

(12) Otherwise, the Lagrange fit is not yet deemed trustworthy, so split the current interval at its midpoint by creating the two packets $\{x_0, x_m, f_0, f_m\}$ and $\{x'_m, x_1, f'_m, f_1\}$, where $x'_m = x_m$ and $f'_m = f_m$ unless $|f_m| < \varepsilon f$, in which case $x'_m = x_m + \texttt{xGuard}$ (à la step 7) and $f'_m = -$. Insert these two packets in the unsearched interval list and proceed back to step 2.

(13) Now the Lagrange fit is deemed trustworthy enough to indicate that any extrema of the Lagrange polynomial within the interval correspond to actual extrema of the function. Solve for the real roots of the derivative of the Lagrange interpolation polynomial (see Appendix for particulars) that may lie within the current interval, if any. These correspond to extrema within the interval. If there are no extrema, the function is monotone in this interval. In this case, the interval may be discarded, so continue the process back at step 2 above.

(14) Otherwise, evaluate the concavity at each extremum within the interval (see Appendix for details). If there is no concavity, the interval is deemed not to contain a root and may be discarded, so proceed back to step 2 above.

(15) The interval now is deemed to have a concavity and may possibly contain a root. Compute the polynomial coefficients and evaluate the polynomial $\hat{f_{ex}}$ at this extremum point (see Appendix for details). If $f_0 \hat{f_{ex}} > 0$ and $|\hat{f_{ex}}| > \texttt{ERZeroPad}\, f_{\max} \varepsilon L$, then the polynomial extremum still has the same sign as the endpoints and is too far from zero to give a reasonable chance of there being a root in the vicinity, so discard the interval and proceed back to step 2.

(16) Evaluate the function at the extremum point, $f_{ex} = F(y_{ex})$. If $f_0 f_{ex} \leq 0$, then a root has been bracketed, so split the current interval at this extremum point, creating two packets $\{x_0, x_{ex}, f_0, f_{ex}\}$ and $\{x'_{ex}, x_1, f'_{ex}, f_1\}$ in the manner of step 12 above (except at the extremum, rather than the interval midpoint). Enter these into the unsearched interval list, and proceed back to step 2.

(17) Otherwise, the extremum point is near enough to zero to warrant further search. Use a polishing root finder (e.g., secant method) to search for it. Limit the number of iterations to `maxIterations` at each trial. Use a number of initial conditions, if necessary. If a root is found at $x = r$ within the interval, go to step 10 above.

(18) Otherwise, no root is deemed present, so discard the interval and go to back to step 2.

(19) All roots in the interval have been found. Sort and return the root list.

V. Example

A *Mathematica* implementation of the `EveryRoot` algorithm was engaged[6] in the search for times during which the planet Mars is occulted by Moon as observed by DSS-14 over a 10-year period. For this purpose, a low-precision simulator[7] of body positions within the solar system was employed, based on mean orbital elements published in the *Explanatory Supplement to the Astronomical Almanac*.

According to this simulator, there are 131 conjunctions in the interval between January 2000 and January 2010, but there are only four occultations, at conjunctions 32, 63, 86, and 104. In the 87672-hr test interval, the total occultation time was only about 3 hr, a fraction of about 3.4×10^{-5}. The maximum occultation interval was 70 min and the shortest was about 18 min. The occultation metric function ranges from a maximum negative value of about –3.14 to a maximum positive value (when occultations are in effect) in the range 0.00026 to 0.004. The dynamic range of the metric, as measured by the negative-to-positive peak ratio when roots were found, is over 12000:1.

`EveryRoot` found all the occultation entry and exit events. Its input parameters were `xGuard` $= \varepsilon x = 30$ s, $\varepsilon f = 10^{-10}$, $\varepsilon L = 0.01$, `maxIterations` $= 30$, and `maxRoots` $= 30{,}000$. It required, on average, about 148 samples of the metric function per conjunction, or about 5.3 samples per day, which averages about 4.5 hr between samples. This average sample interval is about 15 times longer than the shortest occultation period. Clearly, more samples were being expended in searching potential occultation regions than in searching more remote regions, so these figures attest to the adaptability of the algorithm to focus on the important portions of a trajectory.

In actual MPG tests conducted by Jonathan Walther, an 11-hr min-max search for occultations of a Mars orbiter (Mars Reconnaissance Orbiter) by the Moon required 65 samples. If this rate were to be extrapolated to cover the same 10-year span, about 518,000 samples would be required. This represents a factor of 26 times the number required by the `EveryRoot` search.

This is a significant advantage, not only in execution efficiency, but also in simplicity, since the containment interval approach used in the current MPG, primarily for robustness, but for efficiency as well, is not needed.

---

[6] The details of this experiment area reported in the *Mathematica* study `MarsMoonOccultationEvents.nb` found in the MPG *Mathematica* Archives in the Services Preparation Subsystem Portal (internal JPL site).

[7] Details of this simulation are reported in the *Mathematica* study `SpacecraftSolarSystemSimulator.nb` found in the MPG *Mathematica* Archives in the Services Preparation Subsystem Portal (internal JPL site).

VI. Conclusion

This article presents a root finder capable of finding all roots of a function within a given interval subject to very lenient assumptions, which are (1) the function is continuous within the interval; (2) adjacent roots are separated at least by a known amount; (3) a function value less than a given magnitude is considered to be at a root value; (4) the value of the function at the minimum separation distance from a root will not be less than (2) unless there is another separate root there; (5) if two root estimates during an iteration differ by less than a prespecified value, then the search process is considered to have converged; and (6) an optimized cubic polynomial, fit to the function over a given subinterval, that satisfies a given relative error condition at three optimally chosen internal points, reliably indicates whether the function has extrema within the interval.

As will be the case with all root finders, there will be functions that are not sufficiently well behaved that all roots can be found, that is, those that fail the assumptions above. The robustness of the method depends on choosing the three parameters that control the search in such a way as to make the performance acceptable. Such considerations, then, may be application-dependent, and may have to be made on a case-by-case basis.

In validation tests[8] (not shown here), the root finder was subjected to functions having a wide dynamic range, roots of even multiplicity, sparse roots, and closely spaced roots. Although the method, as developed here, assumes that the function is continuous, when it was given discontinuous functions, there was some loss of efficiency while searching near the discontinuities. But in all cases, all roots were located when input parameters were consistent with function behavior.

Like the MPG curve-fitting object, the `EveryRoot` algorithm adapts to the dynamics of the given object function. Functions that are smooth over long intervals require fewer partitions, and hence, fewer samples, than do more dynamically changing functions. However, the interpolation accuracy required merely to ascertain the character of extrema within an interval is orders of magnitude less than that required to fit a function such as the predicted downlink frequency characteristic, for example. The root finder is potentially thus much faster curve fitting. Moreover, its role is direct (i.e., to find roots), whereas the role of curve fitting in event detection was more indirect.

Acknowledgment

The author would like to acknowledge the competent and critical review of this article and the suggestions for its improvement given by W. Van Snyder.

[8] Validation cases appear in the Mathematica study `EveryRoot.nb`, found in the MPG *Mathematica* Archives in the Services Preparation Subsystem Portal (internal JPL site).

References

[1] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, 1986.

[2] C. J. F. Ridders, "A New Algorithm for Computing a Single Root of a Real Continuous Function, *IEEE Transactions on Circuits and Systems*, vol. CAS-26, no. 11, November 1979.

[3] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, National Bureau of Standards Applied Math Series 55, U. S. Dept. of Commerce, U. S. Government Printing Office, Washington, DC 20402, 1964, Tenth Printing 1972.

[4] *Explanatory Supplement to the Astronomical Almanac*, University Science Books, Mill Valley CA, 1992.

Appendix

Cubic Polynomial and Extrema Computation

This Appendix derives the properties of the cubic Lagrange interpolation polynomial used by the `EveryRoot` function. The packet defining the current subinterval being processed will be denoted as $\{x_0, x_1, f_0, f_1\}$. For convenience, the interval will be normalized to occupy [0,1] under the representation

$$\begin{aligned} x &= x_0 + hy \\ h &= (x_1 - x_0) \\ y &= \frac{x - x_0}{h} \\ F(y) &= f(x) \end{aligned} \tag{A-1}$$

Generating the Lagrange polynomial requires two more function samples at internal points, here taken to be the symmetric values a and $b = 1 - a$. With $\mathbf{f} = [f_0, f_a, f_b, f_1]^T$ set to the vector of values at the sample points, *Mathematica* derives the polynomial in y over the normalized interval as the quadratic form $L(y) = [1, y, y^2, y^3]\mathbf{M}\,\mathbf{f}$, in which

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{1 + a - a^2}{a(1-a)} & \frac{1}{a(1-2a)} & -\frac{1}{(1-a)(1-2a)} & 1 \\ \frac{2}{a(1-a)} & -\frac{2-a}{a(1-a)(1-2a)} & \frac{1+a}{a(1-a)(1-2a)} & -\frac{1}{a(1-a)} \\ -\frac{1}{a(1-a)} & \frac{1}{a(1-a)(1-2a)} & -\frac{1}{a(1-a)(1-2a)} & \frac{1}{a(1-a)} \end{bmatrix} \tag{A-2}$$

When the function being fit is taken to be the first few terms of the Taylor expansion of $F(x)$, then the error in the Lagrange interpolation polynomial is found to take the form

$$L(y) - F(y) = \frac{y(1-y)\left(y^2 - y + a - a^2\right)}{24} F^{(4)}\left(\tfrac{1}{2}\right) + \cdots \tag{A-3}$$

That is, the interpolation error is nominally dominated by the fourth derivative of the function at the interval midpoint multiplied by a polynomial of degree 4. This coefficient polynomial defines the dominant error characteristic, which is zero at the interval endpoints and at internal points a and b, by design. The largest first-order errors occur at the extrema of this polynomial, called the *ridge-error* points. *Mathematica* computes these to be at $\tfrac{1}{2}\left(1 - \sqrt{1 - 2a + 2a^2}\right)$, $\tfrac{1}{2}$, and $\tfrac{1}{2}\left(1 + \sqrt{1 - 2a + 2a^2}\right)$. The corresponding error polynomial extrema are similarly found to be

$$\left\{ -\frac{a^2(1-a)}{4}, \left[\frac{1-2a}{4}\right]^2, -\frac{a^2(1-a)^2}{4} \right\} \tag{A-4}$$

All three of these extrema are made equal in magnitude when a is chosen to satisfy the condition

$$\left[\frac{1-2a}{4}\right]^2 = \frac{a^2(1-a)^2}{4} \tag{A-5}$$

Collection of terms and simplification of this condition produces the equation

$$\frac{1}{16}(1 - 4a + 8a^3 - 4a^4) = 0 \tag{A-6}$$

The solution of this equation, for which $0 < a < \frac{1}{2}$, can be verified to be $a = 1 - 1/\sqrt{2}$. This value makes the cubic polynomial a mini-max fit, insofar as the dominant error term in the Taylor expansion is concerned.

The matrix of the quadratic form above for the minimax cubic polynomial is then

$$\mathbf{M}_{opt} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3-2\sqrt{2} & 4+3\sqrt{2} & -2-\sqrt{2} & 1 \\ 4(1+\sqrt{2}) & -10-7\sqrt{2} & 8+5\sqrt{2} & -2(1+\sqrt{2}) \\ -2(1+\sqrt{2}) & 6+4\sqrt{2} & -6-4\sqrt{2} & 2(1+\sqrt{2}) \end{bmatrix} \tag{A-7}$$

The coefficients of the optimized cubic polynomial $F(y) = c_0 + c_1 y + c_2 y^2 + c_3 y^3$ follow from the quadratic form given earlier. They are the vector components

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \mathbf{M}_{opt}\mathbf{f} \tag{A-8}$$

Interpolated values of the Lagrange polynomial may now be computed for any location within the unit interval. The SPICE `POLYDS` utility may be used for this purpose, if desired.

The ridge-error locations are

$$\begin{aligned} y_{e0} &= \tfrac{1}{2}(1 - \sqrt{1 - 2a + 2a^2}) = \tfrac{1}{2}(1 - \sqrt{2 - \sqrt{2}}) = 0.117317 \\ y_m &= \tfrac{1}{2} = 0.50 \\ y_{e1} &= \tfrac{1}{2}(1 + \sqrt{1 - 2a + 2a^2}) = \tfrac{1}{2}(1 + \sqrt{2 - \sqrt{2}}) = 0.882683 \end{aligned} \tag{A-9}$$

The `EveryRoot` algorithm evaluates the interpolation error at the ridge-error points. The values of the optimized cubic polynomial at the these locations can each be expressed as the vector inner product of the four function samples and a constant vector, found by *Mathematica* to be

$$\begin{aligned} \mathbf{v}_{e0} &= \frac{1}{4}\left[(1+\sqrt{2-\sqrt{2}}),\ (1+\sqrt{2+\sqrt{2}}),\ (1-\sqrt{2+\sqrt{2}}),\ (1-\sqrt{2-\sqrt{2}})\right] \\ \mathbf{v}_{em} &= \frac{1}{4}\left[(1-\sqrt{2}),\ (1+\sqrt{2}),\ (1+\sqrt{2}),\ (1-\sqrt{2})\right] \\ \mathbf{v}_{e1} &= \frac{1}{4}\left[(1-\sqrt{2-\sqrt{2}}),\ (1-\sqrt{2+\sqrt{2}}),\ (1+\sqrt{2+\sqrt{2}}),\ (1+\sqrt{2-\sqrt{2}})\right] \end{aligned}$$

Numeric values of these ridge-error vectors are

$$\begin{aligned} \mathbf{v}_{e0} &= [0.441342,\ 0.71194,\ -0.21194,\ 0.0586583] \\ \mathbf{v}_{em} &= [-0.103553,\ 0.603553,\ 0.603553,\ -0.103553] \\ \mathbf{v}_{e1} &= [0.0586583,\ -0.21194,\ 0.71194,\ 0.441342] \end{aligned} \tag{A-10}$$

The interpolation errors at the ridge error points are now computed as

$$\begin{aligned} \varepsilon_{e0} &= \left| f(x_{e0}) - \mathbf{v}_{e0} \cdot \mathbf{f} \right| \\ \varepsilon_{em} &= \left| f(x_m) - \mathbf{v}_{em} \cdot \mathbf{f} \right| \\ \varepsilon_{e1} &= \left| f(x_{e1}) - \mathbf{v}_{e1} \cdot \mathbf{f} \right| \end{aligned} \tag{A-11}$$

where the $f(x_{ei})$ values are those corresponding to the ridge-error points y_{ei} via Equation (A-1).

Once `EveryRoot` has determined that the quality-of-fit criterion has been met, as judged using Equation (1) in algorithm step 11, the coefficients $\{c_0, c_1, c_2, c_3\}$ may be computed from the final set of function samples using Equation (A-8). The extrema can then be determined by finding the roots of the derivative polynomial, whose coefficients are $\{c_1, 2c_2, 3c_3\}$. The SPICE `RQUAD` function may be used for this purpose. Only the real solutions that lie within in the unit interval, if any, are retained.

The curvature along the polynomial is determined by its second derivative, now a linear polynomial whose coefficients are $\{2c_2, 6c_3\}$. At the extremum point y_{ex}, the curvature is thus given by the simple expression $2c_2 + 3c_3 y_{ex}$. An extremum with positive curvature is a local minimum, while one with negative curvature is a local maximum.

Therefore, for an interval whose endpoints are of the same algebraic sign, the Boolean value

$$\texttt{concave} = (f_0(2c_2 + 6c_3 y_{ex}) > 0) \tag{A-12}$$

is true when the cubic polynomial is concave, and false if it is convex, at the extremum.