Numerical Root Finding Algorithms

Contents

Articles

References

Article Licenses

Root-finding algorithm

A **root-finding algorithm** is a numerical method, or algorithm, for finding a value x such that $f(x) = 0$, for a given function f. Such an x is called a root of the function f.

This article is concerned with finding scalar, real or complex roots, approximated as floating point numbers. Finding integer roots or exact algebraic roots are separate problems, whose algorithms have little in common with those discussed here. (See: Diophantine equation as for integer roots)

Finding a root of $f(x) - g(x) = 0$ is the same as solving the equation $f(x) = g(x)$. Here, x is called the *unknown* in the equation. Conversely, any equation can take the canonical form $f(x) = 0$, so equation solving is the same thing as computing (or *finding*) a root of a function.

Numerical root-finding methods use iteration, producing a sequence of numbers that hopefully converge towards a limit (the so called "fixed point") which is a root. The first values of this series are *initial guesses*. The method computes subsequent values based on the old ones and the function f.

The behaviour of root-finding algorithms is studied in numerical analysis. Algorithms perform best when they take advantage of known characteristics of the given function. Thus an algorithm to find isolated real roots of a low-degree polynomial in one variable may bear little resemblance to an algorithm for complex roots of a "black-box" function which is not even known to be differentiable. Questions include ability to separate close roots, robustness in achieving reliable answers despite inevitable numerical errors, and rate of convergence.

Specific algorithms

Bisection Method

The simplest root-finding algorithm is the bisection method. It works when f is a continuous function and it requires previous knowledge of two initial guesses, a and b, such that $f(a)$ and $f(b)$ have opposite signs. Although it is reliable, it converges slowly, gaining one bit of accuracy with each iteration.

Newton's Method (and similar derivative-based methods)

Newton's method assumes the function f to have a continuous derivative. Newton's method may not converge if started too far away from a root. However, when it does converge, it is faster than the bisection method, and is usually quadratic. Newton's method is also important because it readily generalizes to higher-dimensional problems. Newton-like methods with higher order of convergence are the Householder's methods. The first one after Newton's method is Halley's method with cubic order of convergence.

Secant Method

Replacing the derivative in Newton's method with a finite difference, we get the secant method. This method does not require the computation (nor the existence) of a derivative, but the price is slower convergence (the order is approximately 1.6). A generalization of the secant method in higher dimensions is Broyden's method.

False Position (Regula Falsi)

The false position method, also called the *regula falsi* method, is like the secant method. However, instead of retaining the last two points, it makes sure to keep one point on either side of the root. The false position method is faster than the bisection method and more robust than the secant method, but requires the two starting points to bracket the root. Ridders' method is a variant on the false-position method that also evaluates the function at the midpoint of the interval, giving faster convergence with similar robustness.

Interpolation

The secant method also arises if one approximates the unknown function *f* by linear interpolation. When quadratic interpolation is used instead, one arrives at Muller's method. It converges faster than the secant method. A particular feature of this method is that the iterates x_n may become complex.

Inverse Interpolation

This can be avoided by interpolating the inverse of *f*, resulting in the inverse quadratic interpolation method. Again, convergence is asymptotically faster than the secant method, but inverse quadratic interpolation often behaves poorly when the iterates are not close to the root.

Combinations of Methods

Brent's Method

Brent's method is a combination of the bisection method, the secant method and inverse quadratic interpolation. At every iteration, Brent's method decides which method out of these three is likely to do best, and proceeds by doing a step according to that method. This gives a robust and fast method, which therefore enjoys considerable popularity.

Finding roots of polynomials

Much attention has been given to the special case that the function *f* is a polynomial; there exist root-finding algorithms exploiting the polynomial nature of *f*. For a univariate polynomial of degree less than five, there are closed form solutions such as the quadratic formula which produce all roots. However, even this degree-two solution should be used with care to ensure numerical stability. Even more care must be taken with the degree-three and degree-four solutions because of their complexity. Higher-degree polynomials have no such general solution, according to the Abel–Ruffini theorem (1824, 1799).

Birge-Vieta's method combines Horner's method of polynomial evaluation with Newton-Raphson to provide a computational speed-up.

For real roots, Sturm's theorem and Descartes' rule of signs provide guides to locating and separating roots. This plus interval arithmetic combined with Newton's method yields robust and fast algorithms.

The algorithm for isolating the roots, using Descartes' rule of signs and Vincent's theorem, had been originally called *modified Uspensky's algorithm* by its inventors Collins and Akritas[1]. After going through names like "Collins-Akritas method" and "Descartes' method" (too confusing if ones considers Fourier's article[2]), it was finally François Boulier, of Lille University, who gave it the name *Vincent-Collins-Akritas* (VCA) method[3], p. 24, based on the fact that "Uspensky's method" does not exist[4] and neither does "Descartes' method"[5]. This algorithm has been improved by Rouillier and Zimmerman[6], and the resulting implementation is, to the date, the fastest bisection method. It has the same worst case complexity as Sturm algorithm, but is almost always much faster. It is the default algorithm of Maple root-finding function *fsolve*. Another method based on Vincent's theorem is the *Vincent–Akritas–Strzeboński* (VAS) method[7]; it has been shown[8] that the VAS (continued fractions) method is faster than the fastest implementation of the VCA (bisection) method[6], a fact that was independently confirmed elsewhere[9]; more precisely, for the Mignotte polynomials of high degree VAS is about 50,000 times faster than the fastest implementation of VCA. VAS is the default algorithm for root isolation in Mathematica, Sage, SymPy, Xcas. See Budan's theorem for a description of the historical background of these methods. For a comparison between Sturm's method and VAS use the functions realroot(poly) and time(realroot(poly)) of Xcas. By default, to isolate the real roots of poly realroot uses the VAS method; to use Sturm's method write realroot(sturm, poly). See also the External links for a pointer to an iPhone/iPod/iPad application that does the same thing.

One possibility is to form the companion matrix of the polynomial. Since the eigenvalues of this matrix coincide with the roots of the polynomial, one can use any eigenvalue algorithm to find the roots of the polynomial. For instance the classical Bernoulli's method to find the root greater in modulus, if it exists, turns out to be the power method applied to the companion matrix. The inverse power method, which finds some smallest root first, is what drives the Jenkins–Traub method and gives it its numerical stability and fast convergence even in the presence of multiple or clustered roots.

Laguerre's method, as well as Halley's method, use second order derivatives and complex arithmetics, including the complex square root, to exhibit cubic convergence for simple roots, dominating the quadratic convergence displayed by Newton's method.

Bairstow's method uses Newton's method to find quadratic factors of a polynomial with real coefficients. It can determine both real and complex roots of a real polynomial using only real arithmetic.

The simple Durand–Kerner and the slightly more complicated Aberth method simultaneously finds all the roots using only simple complex number arithmetic.

The splitting circle method is useful for finding the roots of polynomials of high degree to arbitrary precision; it has almost optimal complexity in this setting. Another method with this style is the Dandelin–Gräffe method (actually due to Lobachevsky) which factors the polynomial.

Wilkinson's polynomial illustrates that high precision may be necessary when computing the roots of a polynomial given its coefficients: the problem of finding the roots from the coefficients is in general ill-conditioned.

Finding multiple roots of polynomials

If $p(x)$ is a polynomial with a multiple root at r, then finding the value of r can be difficult (inefficient or impossible) for many of the standard root-finding algorithms. Fortunately, there is a technique especially for this case, provided that p is given explicitly as a polynomial in one variable with exact coefficients.

Algorithm

1. First, we need to determine whether $p(x)$ has a multiple root. If $p(x)$ has a multiple root at r, then its derivative $p'(x)$ will also have a root at r (one fewer than $p(x)$ has there). So we calculate the greatest common divisor of the polynomials $p(x)$ and $p'(x)$; adjust the leading coefficient to be one and call it $g(x)$. (See Sturm's theorem.) If $g(x) = 1$, then $p(x)$ has no multiple roots and we can safely use those other root-finding algorithms which work best when there are no multiple roots, and then exit.
2. Now suppose that there is a multiple root. Notice that $g(x)$ will have a root of the same multiplicity at r that $p'(x)$ has and the degree of the polynomial $g(x)$ will generally be much less than that of $p(x)$. Recursively call this routine, i.e. go back to step #1 above, using $g(x)$ in place of $p(x)$. Now suppose that we have found the roots of $g(x)$, i.e. we have factored it.
3. Since r has been found, we can factor $(x-r)$ out of $p(x)$ repeatedly until we have removed all of the roots at r. Repeat this for any other multiple roots until there are no more multiple roots. Then the quotient, i.e. the remaining part of $p(x)$, can be factored in the usual way with one of the other root-finding algorithms. Exit.

Example

Suppose $p(x) = x^3+x^2-5x+3$ is the function whose roots we want to find. We calculate $p'(x) = 3x^2+2x-5$. Now divide $p'(x)$ into $p(x)$ to get $p(x) = p'(x)\cdot((1/3)x+(1/9))+((-32/9)x+(32/9))$. Divide the remainder by $-32/9$ to get $x-1$ which is monic. Divide $x-1$ into $p'(x)$ to get $p'(x) = (x-1)\cdot(3x+5)+0$. Since the remainder is zero, $g(x) = x-1$. So the multiple root of $p(x)$ is $r = 1$. Dividing $p(x)$ by $(x-1)^2$, we get $p(x) = (x+3)(x-1)^2$ so the other root is -3, a single root.

Direct algorithm for multiple root elimination

There is a direct method of eliminating multiple (or repeated) roots from polynomials with exact coefficients (integers, rational numbers, Gaussian integers or rational complex numbers).

Suppose a is a root of polynomial P, with multiplicity $m>1$. Then a will be a root of the formal derivative P', with multiplicity $m-1$. However, P' may have additional roots that are not roots of P. For example, if $P(x)=(x-1)^3(x-3)^3$, then $P'(x)=6(x-1)^2(x-2)(x-3)^2$. So 2 is a root of P', but not of P.

Define G to be the greatest common divisor of P and P'. (Say, $G(x) = (x-1)^2(x-3)^2$).

Finally, G divides P exactly, so form the quotient $Q = P/G$. (Say, $Q(x)= (x-1)(x-3)$). The roots of Q are the roots of P, with multiple roots reduced down to single roots.

As P is a polynomial with exact coefficients, then if the algorithm is executed using exact arithmetic, Q will also be a polynomial with exact coefficients.

Obviously degree(Q) < degree(P). If degree(Q) ≤ 4 then the multiple roots of P may be found algebraically. It is then possible to determine the multiplicities of those roots in P algebraically.

Iterative root-finding algorithms perform well in the absence of multiple roots.

References

[1] Collins, George E.; Alkiviadis G. Akritas (1976). *Polynomial Real Root Isolation Using Descartes' Rule of Signs* (http://doi.acm.org/10.1145/800205.806346). SYMSAC '76, Proceedings of the third ACM symposium on Symbolic and algebraic computation. Yorktown Heights, NY, USA: ACM. pp. 272-275. .

[2] Fourier, Jean Baptiste Joseph (1820). "Sur l'usage du théorème de Descartes dans la recherche des limites des racines" (http://ia600309.us.archive.org/22/items/bulletindesscien20soci/bulletindesscien20soci.pdf). *Bulletin des Sciences, par la Société Philomatique de Paris*: 156-165. .

[3] Boulier, François (2010). *Systèmes polynomiaux : que signifie « résoudre » ?* (http://www.lifl.fr/~boulier/polycopies/resoudre.pdf). Université Lille 1. .

[4] Akritas, Alkiviadis G. (1986). *There's no "Uspensky's Method"* (http://dl.acm.org/citation.cfm?id=32457). In: Proceedings of the fifth ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '86, Waterloo, Ontario, Canada), pp. 88–90. .

[5] Akritas, Alkiviadis G. (2008). *There is no "Descartes' method"* (http://inf-server.inf.uth.gr/~akritas/articles/71.pdf). In: M.J.Wester and M. Beaudin (Eds), Computer Algebra in Education, AullonaPress, USA, pp. 19-35. .

[6] F. Rouillier and P. Zimmerman, *Efficient isolation of polynomial's real roots*, Journal of Computational and Applied Mathematics **162** (2004)

[7] Akritas, Alkiviadis G.; A.W. Strzeboński, P.S. Vigklas (2008). "Improving the performance of the continued fractions method using new bounds of positive roots" (http://www.lana.lt/journal/30/Akritas.pdf). *Nonlinear Analysis: Modelling and Control* **13**: 265–279. .

[8] Akritas, Alkiviadis G.; Adam W. Strzeboński (2005). "A Comparative Study of Two Real Root Isolation Methods" (http://www.lana.lt/journal/19/Akritas.pdf). *Nonlinear Analysis: Modelling and Control* **10** (4): 297-304. .

[9] Tsigaridas, P.E.; I.Z. Emiris, (2006). "Univariate polynomial real root isolation: Continued fractions revisited" (http://www.springerlink.com/content/c70468755x403481/). *LNCS* **4168**: 817-828. .

- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Chapter 9. Root Finding and Nonlinear Sets of Equations" (http://apps.nrbook.com/empanel/index.html#pg=442). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

External links

- Root-finding algorithms Java code By Behzad Torkian (http://www.torkian.info/Site/Research/Entries/2008/2/28_Root-finding_algorithm_Java_Code_(_Secant,_Bisection,_Newton_).html)
- Animations for Fixed Point Iteration (http://math.fullerton.edu/mathews/a2001/Animations/RootFinding/FixedPoint/FixedPoint.html)
- GAMS: Roots of polynomials with real coefficients (http://gams.nist.gov/serve.cgi/Class/F1a1/)
- Free online polynomial root finder for both real and complex coefficients (http://www.hvks.com/Numerical/websolver.php)
- Kehagias, Spyros: RealRoots, a free App for iPhone, iPod Touch and iPad to compare Sturm's method and VAS http://itunes.apple.com/gr/app/realroots/id483609988?mt=8

Multiplicity (mathematics)

In mathematics, the **multiplicity** of a member of a multiset is the number of times it appears in the multiset. For example, the number of times a given polynomial equation has a root at a given point.

The notion of multiplicity is important to be able to count correctly without specifying exceptions (for example, *double roots* counted twice). Hence the expression, "counted with (sometimes implicit) multiplicity".

If multiplicity is ignored, this may be emphasized by counting the number of **distinct** elements, as in "the number of distinct roots". However, whenever a set (as opposed to multiset) is formed, multiplicity is automatically ignored, without requiring use of the term "distinct".

Multiplicity of a prime factor

In the prime factorization, for example,

$$60 = 2 \times 2 \times 3 \times 5$$

the multiplicity of the prime factor 2 is 2, while the multiplicity of each of the prime factors 3 and 5 is 1. Thus, 60 has 4 prime factors, but only 3 distinct prime factors.

Multiplicity of a root of a polynomial

Let F be a field and $p(x)$ be a polynomial in one variable and coefficients in F. An element $a \in F$ is called a root of multiplicity k of $p(x)$ if there is a polynomial $s(x)$ such that $s(a) \neq 0$ and $p(x) = (x - a)^k s(x)$. If $k = 1$, then a is called a *simple root*.

For instance, the polynomial $p(x) = x^3 + 2x^2 - 7x + 4$ has 1 and -4 as roots, and can be written as $p(x) = (x + 4)(x - 1)^2$. This means that 1 is a root of multiplicity 2, and -4 is a 'simple' root (of multiplicity 1). Multiplicity can be thought of as "How many times does the solution appear in the original equation?".

The discriminant of a polynomial is zero if and only if the polynomial has a multiple root.

Behavior of a polynomial function near a root in relation to its multiplicity

Let $f(x)$ be a polynomial function. Then, if f is graphed on a Cartesian coordinate system, its graph will cross the x-axis at real zeros of odd multiplicity and will touch but not cross the x-axis at real zeros of even multiplicity. In addition, if $f(x)$ has a zero with a multiplicity greater than 1, the graph will be tangent to the x-axis, in other words it will have slope 0 there.

In complex analysis

Let z_0 be a root of a holomorphic function f, and let n be the least positive integer such that, the n^{th} derivative of f evaluated at z_0 differs from zero. Then the power series of f about z_0 begins with the n^{th} term, and f is said to have a root of multiplicity (or "order") n. If $n = 1$, the root is called a simple root (Krantz 1999, p. 70).

We can also define the multiplicity of the zeroes and poles of a meromorphic function thus: If we have a meromorphic function $f = g/h$, take the Taylor expansions of g and h about a point z_0, and find the first non-zero term in each (denote the order of the terms m and n respectively). if $m = n$, then the point has non-zero value. If $m > n$, then the point is a zero of multiplicity $m - n$. If $m < n$, then the point has a pole of multiplicity $n - m$.

References

- Krantz, S. G. *Handbook of Complex Variables*. Boston, MA: Birkhäuser, 1999. ISBN 0-8176-4011-8.

Descartes' rule of signs

In mathematics, **Descartes' rule of signs**, first described by René Descartes in his work *La Géométrie*, is a technique for determining the number of positive or negative real roots of a polynomial.

The rule gives us an upper bound number of positive or negative roots of a polynomial. It is not a complete criterion, i.e. it does not tell the exact number of positive or negative roots.

Descartes' rule of signs

Positive roots

The rule states that if the terms of a single-variable polynomial with real coefficients are ordered by descending variable exponent, then the number of positive roots of the polynomial is either equal to the number of sign differences between consecutive nonzero coefficients, or is less than it by a multiple of 2. Multiple roots of the same value are counted separately.

Negative roots

As a corollary of the rule, the number of negative roots is the number of sign changes after multiplying the coefficients of odd-power terms by −1, or fewer than it by a multiple of 2. This procedure is equivalent to substituting the negation of the variable for the variable itself: For example, to find the number of negative roots of $f(x) = ax^3 + bx^2 + cx + d$, we equivalently ask how many positive roots there are for $-x$ in $f(-x) = a(-x)^3 + b(-x)^2 + c(-x) + d = -ax^3 + bx^2 - cx + d \equiv g(x)$. Using Descartes' rule of signs on $g(x)$ gives the number of positive roots x_i of g, and since $g(x) = f(-x)$ it gives the number of positive roots $(-x_i)$ of f, which is the same as the number of negative roots x_i of f.

Example

The polynomial

$$+x^3 + x^2 - x - 1$$

has one sign change between the second and third terms (the sequence of pairs of successive signs is ++, +−, −−). Therefore it has exactly one positive root. Note that the leading sign needs to be considered although in this particular example it does not affect the answer. To find the number of negative roots, change the signs of the coefficients of the terms with odd exponents, to obtain a second polynomial

$$-x^3 + x^2 + x - 1.$$

This polynomial has two sign changes (the sequence of pairs of successive signs is −+, ++, +−), meaning that this second polynomial has two or zero positive roots; thus the original polynomial has two or zero negative roots.

In fact, the factorization of the first polynomial is

$$(x+1)^2(x-1),$$

so the roots are −1 (twice) and 1.

The factorization of the second polynomial is

$$-(x-1)^2(x+1),$$

So here, the roots are 1 (twice) and −1, the negation of the roots of the original polynomial.

Complex roots

Since any n^{th} degree polynomial has exactly n roots, the minimum number of complex roots is equal to

$$n - (p+q),$$

where p denotes the maximum number of positive roots, q denotes the maximum number of negative roots (both of which can be found using Descartes' rule of signs), and n denotes the degree of the equation. A simple example is the polynomial

$$x^2 + b.$$

If $b > 0$, this has no sign changes, and the polynomial does not change when odd-powered terms (of which there are none in this example) have their coefficients multiplied by −1. Thus the maximum number of positive roots is zero, as is the maximum number of negative roots; so the minimum (and in this case exact) number of complex roots is $n - (p+q) = 2 - (0+0) = 2$.

Special case

The subtraction of only multiples of 2 from the maximal number of positive roots occurs because the polynomial may have complex roots, which always come in pairs since the rule applies to polynomials whose coefficients are real. Thus if the polynomial is known to have all real roots, this rule allows one to find the exact number of positive and negative roots. Since it is easy to determine the multiplicity of zero as a root, the sign of all roots can be determined in this case.

Generalizations

If the real polynomial P has k real positive roots counted with multiplicity, then for every $a > 0$ there are at least k changes of sign in the sequence of coefficients of the Taylor series of the function $e^{ax}P(x)$.[1]

In the 1970s Askold Georgevich Khovanskiĭ developed the theory of fewnomials that generalises Descartes' rule.[2] The rule of signs can be thought of as stating that the number of real roots of a polynomial is dependent on the polynomial's complexity, and that this complexity is proportional to the number of monomials it has, not its degree. Khovanskiĭ showed that this holds true not just for polynomials but for algebraic combinations of many

transcendental functions, the so-called Pfaffian functions.

Notes

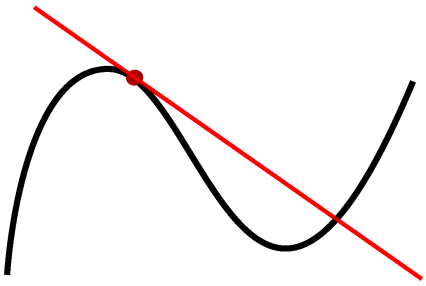
[1] Vladimir P. Kostov, *A mapping defined by the Schur-Szegő composition*, Comptes Rendus Acad. Bulg. Sci. tome 63, No. 7, 2010, 943 - 952.
[2] A. G. Khovanskii, *Fewnomials*, Princeton University Press (1991) ISBN 0-8218-4547-0.

External links

- Descartes' Rule of Signs (http://www.cut-the-knot.org/fta/ROS2.shtml) — Proof of the Rule

Derivative

In calculus, a branch of mathematics, the **derivative** is a measure of how a function changes as its input changes. Loosely speaking, a derivative can be thought of as how much one quantity is changing in response to changes in some other quantity; for example, the derivative of the position of a moving object with respect to time is the object's instantaneous velocity.

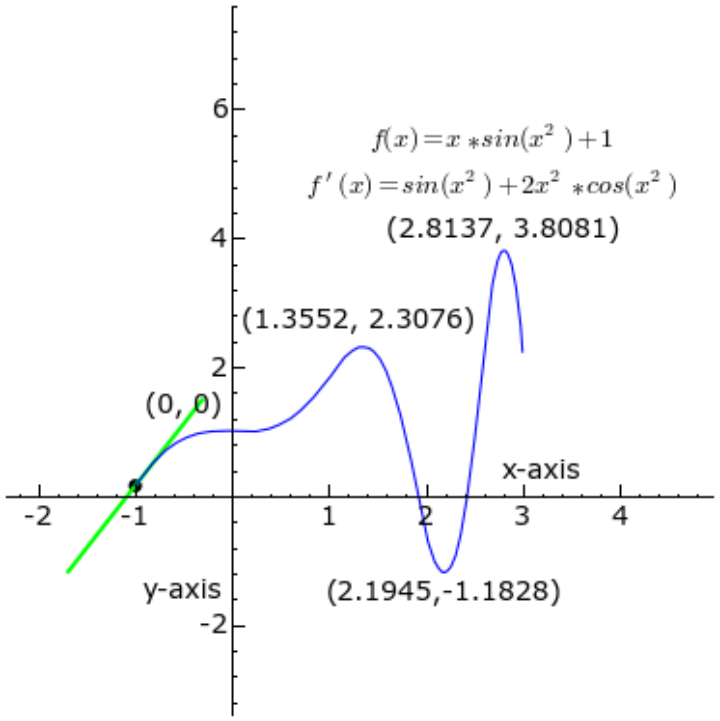


The graph of a function, drawn in black, and a tangent line to that function, drawn in red. The slope of the tangent line is equal to the derivative of the function at the marked point.

The derivative of a function at a chosen input value describes the best linear approximation of the function near that input value. For a real-valued function of a single real variable, the derivative at a point equals the slope of the tangent line to the graph of the function at that point. In higher dimensions, the derivative of a function at a point is a linear transformation called the linearization.[1] A closely related notion is the differential of a function.

The process of finding a derivative is called **differentiation**. The reverse process is called **antidifferentiation**. The fundamental theorem of calculus states that antidifferentiation is the same as integration. Differentiation and integration constitute the two fundamental operations in single-variable calculus.

Differentiation and the derivative



At each point, the derivative of $f(x) = 1 + x \sin x^2$ is the slope of a line that is tangent to the curve. The line is always tangent to the blue curve; its slope is the derivative. Note derivative is positive where green, negative where red, and zero where black.

Differentiation is a method to compute the rate at which a dependent output y changes with respect to the change in the independent input x. This rate of change is called the **derivative** of y with respect to x. In more precise language, the dependence of y upon x means that y is a function of x. This functional relationship is often denoted $y = f(x)$, where f denotes the function. If x and y are real numbers, and if the graph of y is plotted against x, the derivative measures the slope of this graph at each point.

The simplest case is when y is a linear function of x, meaning that the graph of y against x is a straight line. In this case, $y = f(x) = m\ x + b$, for real numbers m and b, and the slope m is given by

$$m = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x},$$

where the symbol Δ (the uppercase form of the Greek letter Delta) is an abbreviation for "change in." This formula is true because

$$y + \Delta y = f(x + \Delta x) = m\ (x + \Delta x) + b = m\ x + b + m\ \Delta x = y + m\Delta x.$$

It follows that $\Delta y = m\ \Delta x$.

This gives an exact value for the slope of a straight line. If the function f is not linear (i.e. its graph is not a straight line), however, then the change in y divided by the change in x varies: differentiation is a method to find an exact value for this rate of change at any given value of x.

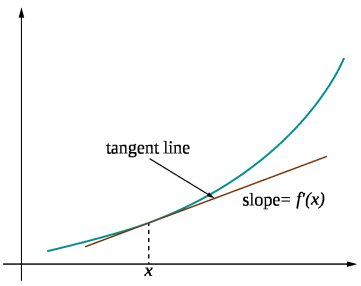


Figure 1. The tangent line at $(x, f(x))$

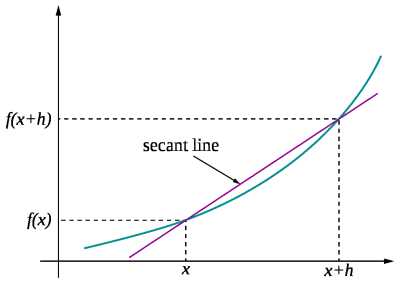


Figure 2. The secant to curve $y=f(x)$ determined by points $(x, f(x))$ and $(x+h, f(x+h))$

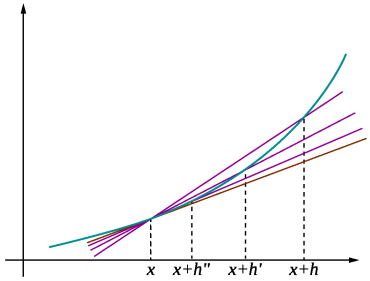


Figure 3. The tangent line as limit of secants

The idea, illustrated by Figures 1-3, is to compute the rate of change as the limiting value of the ratio of the differences Δy / Δx as Δx becomes infinitely small.

In Leibniz's notation, such an infinitesimal change in x is denoted by dx, and the derivative of y with respect to x is written

$$\frac{dy}{dx}$$

suggesting the ratio of two infinitesimal quantities. (The above expression is read as "the derivative of y with respect to x", "d y by d x", or "d y over d x". The oral form "d y d x" is often used conversationally, although it may lead to confusion.)

The most common approach[2] to turn this intuitive idea into a precise definition uses limits, but there are other methods, such as non-standard analysis.[3]

Definition via difference quotients

Let f be a real valued function. In classical geometry, the tangent line to the graph of the function f at a real number a was the unique line through the point $(a, f(a))$ that did *not* meet the graph of f transversally, meaning that the line did not pass straight through the graph. The derivative of y with respect to x at a is, geometrically, the slope of the tangent line to the graph of f at a. The slope of the tangent line is very close to the slope of the line through $(a, f(a))$ and a nearby point on the graph, for example $(a + h, f(a + h))$. These lines are called secant lines. A value of h close to zero gives a good approximation to the slope of the tangent line, and smaller values (in absolute value) of h will, in general, give better approximations. The slope m of the secant line is the difference between the y values of these points divided by the difference between the x values, that is,

$$m = \frac{\Delta f(x)}{\Delta x} = \frac{f(x+h) - f(x)}{(x+h) - (x)} = \frac{f(x+h) - f(x)}{h}.$$

This expression is Newton's difference quotient. The derivative is the value of the difference quotient as the secant lines approach the tangent line. Formally, the **derivative** of the function f at a is the limit

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

of the difference quotient as h approaches zero, if this limit exists. If the limit exists, then f is **differentiable** at a. Here $f'(a)$ is one of several common notations for the derivative (see below).

Equivalently, the derivative satisfies the property that

$$\lim_{h \to 0} \frac{f(a+h) - f(a) - f'(a) \cdot h}{h} = 0,$$

which has the intuitive interpretation (see Figure 1) that the tangent line to f at a gives the *best linear approximation*

$$f(a+h) \approx f(a) + f'(a)h$$

to f near a (i.e., for small h). This interpretation is the easiest to generalize to other settings (see below).

Substituting 0 for h in the difference quotient causes division by zero, so the slope of the tangent line cannot be found directly using this method. Instead, define $Q(h)$ to be the difference quotient as a function of h:

$$Q(h) = \frac{f(a+h) - f(a)}{h}.$$

$Q(h)$ is the slope of the secant line between $(a, f(a))$ and $(a + h, f(a + h))$. If f is a continuous function, meaning that its graph is an unbroken curve with no gaps, then Q is a continuous function away from $h = 0$. If the limit $\lim_{h \to 0} Q(h)$ exists, meaning that there is a way of choosing a value for $Q(0)$ that makes the graph of Q a continuous function, then the function f is differentiable at a, and its derivative at a equals $Q(0)$.

In practice, the existence of a continuous extension of the difference quotient $Q(h)$ to $h = 0$ is shown by modifying the numerator to cancel h in the denominator. This process can be long and tedious for complicated functions, and many shortcuts are commonly used to simplify the process.

Example

The squaring function $f(x) = x^2$ is differentiable at $x = 3$, and its derivative there is 6. This result is established by calculating the limit as h approaches zero of the difference quotient of $f(3)$:

$$f'(3) = \lim_{h \to 0} \frac{f(3+h) - f(3)}{h} = \lim_{h \to 0} \frac{(3+h)^2 - 3^2}{h} = \lim_{h \to 0} \frac{9 + 6h + h^2 - 9}{h} = \lim_{h \to 0} \frac{6h + h^2}{h} = \lim_{h \to 0} (6 + h).$$

The last expression shows that the difference quotient equals $6 + h$ when $h \neq 0$ and is undefined when $h = 0$, because of the definition of the difference quotient. However, the definition of the limit says the difference quotient does not need to be defined when $h = 0$. The limit is the result of letting h go to zero, meaning it is the value that $6 + h$ tends to as h becomes very small:

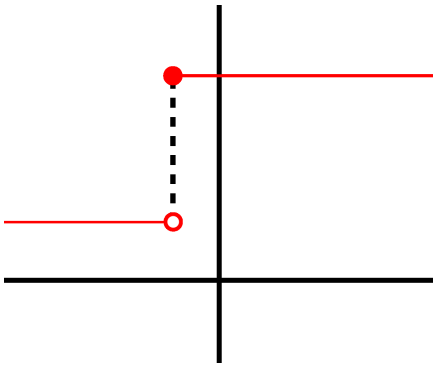
$$\lim_{h \to 0} (6 + h) = 6 + 0 = 6.$$

Hence the slope of the graph of the squaring function at the point (3, 9) is 6, and so its derivative at $x = 3$ is $f'(3) = 6$.

More generally, a similar computation shows that the derivative of the squaring function at $x = a$ is $f'(a) = 2a$.

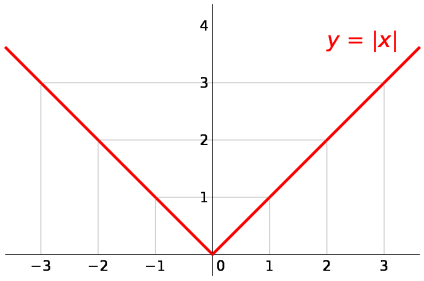
Continuity and differentiability

If $y = f(x)$ is differentiable at a, then f must also be continuous at a. As an example, choose a point a and let f be the step function that returns a value, say 1, for all x less than a, and returns a different value, say 10, for all x greater than or equal to a. f cannot have a derivative at a. If h is negative, then $a + h$ is on the low part of the step, so the secant line from a to $a + h$ is very steep, and as h tends to zero the slope tends to infinity. If h is positive, then $a + h$ is on the high part of the step, so the secant line from a to $a + h$ has slope zero. Consequently the secant lines do not approach any single slope, so the limit of the difference quotient does not exist.[4]



This function does not have a derivative at the marked point, as the function is not continuous there.

However, even if a function is continuous at a point, it may not be differentiable there. For example, the absolute value function $y = |x|$ is continuous at $x = 0$, but it is not differentiable there. If h is positive, then the slope of the secant line from 0 to h is one, whereas if h is negative, then the slope of the secant line from 0 to h is negative one. This can be seen graphically as a "kink" or a "cusp" in the graph at $x = 0$. Even a function with a smooth graph is not differentiable at a point where its tangent is vertical: For instance, the function $y = x^{1/3}$ is not differentiable at $x = 0$.



The absolute value function is continuous, but fails to be differentiable at $x = 0$ since the tangent slopes do not approach the same value from the left as they do from the right.

In summary: for a function f to have a derivative it is *necessary* for the function f to be continuous, but continuity alone is not *sufficient*.

Most functions that occur in practice have derivatives at all points or at almost every point. Early in the history of calculus, many mathematicians assumed that a continuous function was differentiable at most points. Under mild conditions, for example if the function is a monotone function or a Lipschitz function, this is true. However, in 1872 Weierstrass found the first example of a function that is continuous everywhere but differentiable nowhere. This example is now known as the Weierstrass function. In 1931, Stefan Banach proved that the set of functions that have a derivative at some point is a meager set in the space of all continuous functions.[5] Informally, this means that hardly any continuous functions have a derivative at even one point.

The derivative as a function

Let f be a function that has a derivative at every point a in the domain of f. Because every point a has a derivative, there is a function that sends the point a to the derivative of f at a. This function is written $f'(x)$ and is called the *derivative function* or the *derivative* of f. The derivative of f collects all the derivatives of f at all the points in the domain of f.

Sometimes f has a derivative at most, but not all, points of its domain. The function whose value at a equals $f'(a)$ whenever $f'(a)$ is defined and elsewhere is undefined is also called the derivative of f. It is still a function, but its domain is strictly smaller than the domain of f.

Using this idea, differentiation becomes a function of functions: The derivative is an operator whose domain is the set of all functions that have derivatives at every point of their domain and whose range is a set of functions. If we

denote this operator by D, then $D(f)$ is the function $f'(x)$. Since $D(f)$ is a function, it can be evaluated at a point a. By the definition of the derivative function, $D(f)(a) = f'(a)$.

For comparison, consider the doubling function $f(x) = 2x$; f is a real-valued function of a real number, meaning that it takes numbers as inputs and has numbers as outputs:

$$1 \mapsto 2,$$
$$2 \mapsto 4,$$
$$3 \mapsto 6.$$

The operator D, however, is not defined on individual numbers. It is only defined on functions:

$$D(x \mapsto 1) = (x \mapsto 0),$$
$$D(x \mapsto x) = (x \mapsto 1),$$
$$D(x \mapsto x^2) = (x \mapsto 2 \cdot x).$$

Because the output of D is a function, the output of D can be evaluated at a point. For instance, when D is applied to the squaring function,

$$x \mapsto x^2,$$

D outputs the doubling function,

$$x \mapsto 2x,$$

which we named $f(x)$. This output function can then be evaluated to get $f(1) = 2$, $f(2) = 4$, and so on.

Higher derivatives

Let f be a differentiable function, and let $f'(x)$ be its derivative. The derivative of $f'(x)$ (if it has one) is written $f''(x)$ and is called the **second derivative of** f. Similarly, the derivative of a second derivative, if it exists, is written $f'''(x)$ and is called the **third derivative of** f. These repeated derivatives are called *higher-order derivatives*.

If $x(t)$ represents the position of an object at time t, then the higher-order derivatives of x have physical interpretations. The second derivative of x is the derivative of $x'(t)$, the velocity, and by definition this is the object's acceleration. The third derivative of x is defined to be the jerk, and the fourth derivative is defined to be the jounce.

A function f need not have a derivative, for example, if it is not continuous. Similarly, even if f does have a derivative, it may not have a second derivative. For example, let

$$f(x) = \begin{cases} +x^2, & \text{if } x \geq 0 \\ -x^2, & \text{if } x \leq 0. \end{cases}$$

Calculation shows that f is a differentiable function whose derivative is

$$f'(x) = \begin{cases} +2x, & \text{if } x \geq 0 \\ -2x, & \text{if } x \leq 0. \end{cases}$$

$f'(x)$ is twice the absolute value function, and it does not have a derivative at zero. Similar examples show that a function can have k derivatives for any non-negative integer k but no $(k + 1)$-order derivative. A function that has k successive derivatives is called ***k* times differentiable**. If in addition the kth derivative is continuous, then the function is said to be of differentiability class C^k. (This is a stronger condition than having k derivatives. For an example, see differentiability class.) A function that has infinitely many derivatives is called **infinitely differentiable** or **smooth**.

On the real line, every polynomial function is infinitely differentiable. By standard differentiation rules, if a polynomial of degree n is differentiated n times, then it becomes a constant function. All of its subsequent derivatives are identically zero. In particular, they exist, so polynomials are smooth functions.

The derivatives of a function *f* at a point *x* provide polynomial approximations to that function near *x*. For example, if *f* is twice differentiable, then

$$f(x+h) \approx f(x) + f'(x)h + \tfrac{1}{2}f''(x)h^2$$

in the sense that

$$\lim_{h\to 0} \frac{f(x+h) - f(x) - f'(x)h - \frac{1}{2}f''(x)h^2}{h^2} = 0.$$

If *f* is infinitely differentiable, then this is the beginning of the Taylor series for *f*.

Inflection point

A point where the second derivative of a function changes sign is called an **inflection point**.[6] At an inflection point, the second derivative may be zero, as in the case of the inflection point x=0 of the function $y=x^3$, or it may fail to exist, as in the case of the inflection point x=0 of the function $y=x^{1/3}$. At an inflection point, a function switches from being a convex function to being a concave function or vice versa.

Notations for differentiation

Leibniz's notation

The notation for derivatives introduced by Gottfried Leibniz is one of the earliest. It is still commonly used when the equation $y = f(x)$ is viewed as a functional relationship between dependent and independent variables. Then the first derivative is denoted by

$$\frac{dy}{dx}, \quad \frac{df}{dx}(x), \quad \text{or} \quad \frac{d}{dx}f(x),$$

and was once thought of as an infinitesimal quotient. Higher derivatives are expressed using the notation

$$\frac{d^n y}{dx^n}, \quad \frac{d^n f}{dx^n}(x), \quad \text{or} \quad \frac{d^n}{dx^n}f(x)$$

for the *n*th derivative of $y = f(x)$ (with respect to x). These are abbreviations for multiple applications of the derivative operator. For example,

$$\frac{d^2y}{dx^2} = \frac{d}{dx}\left(\frac{dy}{dx}\right).$$

With Leibniz's notation, we can write the derivative of y at the point $x = a$ in two different ways:

$$\left.\frac{dy}{dx}\right|_{x=a} = \frac{dy}{dx}(a).$$

Leibniz's notation allows one to specify the variable for differentiation (in the denominator). This is especially relevant for partial differentiation. It also makes the chain rule easy to remember:[7]

$$\frac{dy}{dx} = \frac{dy}{du}\cdot\frac{du}{dx}.$$

Lagrange's notation

Sometimes referred to as **prime notation**,[8] one of the most common modern notations for differentiation is due to Joseph-Louis Lagrange and uses the prime mark, so that the derivative of a function $f(x)$ is denoted $f'(x)$ or simply f'. Similarly, the second and third derivatives are denoted

$$(f')' = f'' \text{ and } (f'')' = f'''.$$

To denote the number of derivatives beyond this point, some authors use Roman numerals in superscript, whereas others place the number in parentheses:

$$f^{\mathrm{iv}} \text{ or } f^{(4)}.$$

The latter notation generalizes to yield the notation $f^{(n)}$ for the nth derivative of f — this notation is most useful when we wish to talk about the derivative as being a function itself, as in this case the Leibniz notation can become cumbersome.

Newton's notation

Newton's notation for differentiation, also called the dot notation, places a dot over the function name to represent a time derivative. If $y = f(t)$, then

$$\dot{y} \text{ and } \ddot{y}$$

denote, respectively, the first and second derivatives of y with respect to t. This notation is used exclusively for time derivatives, meaning that the independent variable of the function represents time. It is very common in physics and in mathematical disciplines connected with physics such as differential equations. While the notation becomes unmanageable for high-order derivatives, in practice only very few derivatives are needed.

Euler's notation

Euler's notation uses a differential operator D, which is applied to a function f to give the first derivative Df. The second derivative is denoted D^2f, and the nth derivative is denoted D^nf.

If $y = f(x)$ is a dependent variable, then often the subscript x is attached to the D to clarify the independent variable x. Euler's notation is then written

$$D_x y \text{ or } D_x f(x),$$

although this subscript is often omitted when the variable x is understood, for instance when this is the only variable present in the expression.

Euler's notation is useful for stating and solving linear differential equations.

Computing the derivative

The derivative of a function can, in principle, be computed from the definition by considering the difference quotient, and computing its limit. In practice, once the derivatives of a few simple functions are known, the derivatives of other functions are more easily computed using *rules* for obtaining derivatives of more complicated functions from simpler ones.

Derivatives of elementary functions

Most derivative computations eventually require taking the derivative of some common functions. The following incomplete list gives some of the most frequently used functions of a single real variable and their derivatives.

- *Derivatives of powers*: if

$$f(x) = x^r,$$

where r is any real number, then

$$f'(x) = rx^{r-1},$$

wherever this function is defined. For example, if $f(x) = x^{1/4}$, then

$$f'(x) = (1/4)x^{-3/4},$$

and the derivative function is defined only for positive x, not for $x = 0$. When $r = 0$, this rule implies that $f'(x)$ is zero for $x \neq 0$, which is almost the constant rule (stated below).

- *Exponential and logarithmic functions*:

$$\frac{d}{dx}e^x = e^x.$$
$$\frac{d}{dx}a^x = \ln(a)a^x.$$
$$\frac{d}{dx}\ln(x) = \frac{1}{x}, \qquad x > 0.$$
$$\frac{d}{dx}\log_a(x) = \frac{1}{x\ln(a)}.$$

- *Trigonometric functions*:

$$\frac{d}{dx}\sin(x) = \cos(x).$$
$$\frac{d}{dx}\cos(x) = -\sin(x).$$
$$\frac{d}{dx}\tan(x) = \sec^2(x) = \frac{1}{\cos^2(x)} = 1 + \tan^2(x).$$

- *Inverse trigonometric functions*:

$$\frac{d}{dx}\arcsin(x) = \frac{1}{\sqrt{1-x^2}}.$$
$$\frac{d}{dx}\arccos(x) = -\frac{1}{\sqrt{1-x^2}}.$$
$$\frac{d}{dx}\arctan(x) = \frac{1}{1+x^2}.$$

Rules for finding the derivative

In many cases, complicated limit calculations by direct application of Newton's difference quotient can be avoided using differentiation rules. Some of the most basic rules are the following.

- *Constant rule*: if $f(x)$ is constant, then

$$f' = 0.$$

- *Sum rule*:

$(af + bg)' = af' + bg'$ for all functions f and g and all real numbers a and b.

- *Product rule*:

$(fg)' = f'g + fg'$ for all functions f and g. By extension, this means that the derivative of a constant times a function is the constant times the derivative of the function: $\frac{d}{dx}\pi x^2 = 2\pi x.$

- *Quotient rule*:

$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$ for all functions f and g at all inputs where $g \neq 0$.

- *Chain rule*: If $f(x) = h(g(x))$, then

$$f'(x) = h'(g(x)) \cdot g'(x).$$

Example computation

The derivative of

$$f(x) = x^4 + \sin(x^2) - \ln(x)e^x + 7$$

is

$$\begin{aligned} f'(x) &= 4x^{(4-1)} + \frac{d\left(x^2\right)}{dx}\cos(x^2) - \frac{d\left(\ln x\right)}{dx}e^x - \ln x\frac{d\left(e^x\right)}{dx} + 0 \\ &= 4x^3 + 2x\cos(x^2) - \frac{1}{x}e^x - \ln(x)e^x. \end{aligned}$$

Here the second term was computed using the chain rule and third using the product rule. The known derivatives of the elementary functions x^2, x^4, sin(x), ln(x) and exp(x) = e^x, as well as the constant 7, were also used.

Derivatives in higher dimensions

Derivatives of vector valued functions

A vector-valued function $\mathbf{y}(t)$ of a real variable sends real numbers to vectors in some vector space $\mathbf{R}^n$. A vector-valued function can be split up into its coordinate functions $y_1(t)$, $y_2(t)$, …, $y_n(t)$, meaning that $\mathbf{y}(t) = (y_1(t), ..., y_n(t))$. This includes, for example, parametric curves in $\mathbf{R}^2$ or $\mathbf{R}^3$. The coordinate functions are real valued functions, so the above definition of derivative applies to them. The derivative of $\mathbf{y}(t)$ is defined to be the vector, called the tangent vector, whose coordinates are the derivatives of the coordinate functions. That is,

$$\mathbf{y}'(t) = (y_1'(t), \ldots, y_n'(t)).$$

Equivalently,

$$\mathbf{y}'(t) = \lim_{h \to 0} \frac{\mathbf{y}(t+h) - \mathbf{y}(t)}{h},$$

if the limit exists. The subtraction in the numerator is subtraction of vectors, not scalars. If the derivative of $\mathbf{y}$ exists for every value of t, then $\mathbf{y}'$ is another vector valued function.

If $\mathbf{e}_1$, …, $\mathbf{e}_n$ is the standard basis for $\mathbf{R}^n$, then $\mathbf{y}(t)$ can also be written as $y_1(t)\mathbf{e}_1 + \ldots + y_n(t)\mathbf{e}_n$. If we assume that the derivative of a vector-valued function retains the linearity property, then the derivative of $\mathbf{y}(t)$ must be

$$y_1'(t)\mathbf{e}_1 + \cdots + y_n'(t)\mathbf{e}_n$$

because each of the basis vectors is a constant.

This generalization is useful, for example, if $\mathbf{y}(t)$ is the position vector of a particle at time t; then the derivative $\mathbf{y}'(t)$ is the velocity vector of the particle at time t.

Partial derivatives

Suppose that f is a function that depends on more than one variable. For instance,

$$f(x, y) = x^2 + xy + y^2.$$

f can be reinterpreted as a family of functions of one variable indexed by the other variables:

$$f(x, y) = f_x(y) = x^2 + xy + y^2.$$

In other words, every value of x chooses a function, denoted f_x, which is a function of one real number.[9] That is,

$$x \mapsto f_x,$$

$$f_x(y) = x^2 + xy + y^2.$$

Once a value of x is chosen, say a, then $f(x,y)$ determines a function f_a that sends y to $a^2 + ay + y^2$:

$$f_a(y) = a^2 + ay + y^2.$$

In this expression, a is a *constant*, not a *variable*, so f_a is a function of only one real variable. Consequently the definition of the derivative for a function of one variable applies:

$$f'_a(y) = a + 2y.$$

The above procedure can be performed for any choice of a. Assembling the derivatives together into a function gives a function that describes the variation of f in the y direction:

$$\frac{\partial f}{\partial y}(x, y) = x + 2y.$$

This is the partial derivative of f with respect to y. Here ∂ is a rounded d called the **partial derivative symbol**. To distinguish it from the letter d, ∂ is sometimes pronounced "der", "del", or "partial" instead of "dee".

In general, the **partial derivative** of a function $f(x_1, \ldots, x_n)$ in the direction x_i at the point $(a_1 \ldots, a_n)$ is defined to be:

$$\frac{\partial f}{\partial x_i}(a_1, \ldots, a_n) = \lim_{h \to 0} \frac{f(a_1, \ldots, a_i + h, \ldots, a_n) - f(a_1, \ldots, a_i, \ldots, a_n)}{h}.$$

In the above difference quotient, all the variables except x_i are held fixed. That choice of fixed values determines a function of one variable

$$f_{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n}(x_i) = f(a_1, \ldots, a_{i-1}, x_i, a_{i+1}, \ldots, a_n),$$

and, by definition,

$$\frac{df_{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n}}{dx_i}(a_i) = \frac{\partial f}{\partial x_i}(a_1, \ldots, a_n).$$

In other words, the different choices of a index a family of one-variable functions just as in the example above. This expression also shows that the computation of partial derivatives reduces to the computation of one-variable derivatives.

An important example of a function of several variables is the case of a scalar-valued function $f(x_1, \ldots x_n)$ on a domain in Euclidean space $\mathbf{R}^n$ (e.g., on $\mathbf{R}^2$ or $\mathbf{R}^3$). In this case f has a partial derivative $\partial f/\partial x_j$ with respect to each variable x_j. At the point a, these partial derivatives define the vector

$$\nabla f(a) = \left(\frac{\partial f}{\partial x_1}(a), \ldots, \frac{\partial f}{\partial x_n}(a) \right).$$

This vector is called the **gradient** of f at a. If f is differentiable at every point in some domain, then the gradient is a vector-valued function ∇f that takes the point a to the vector $\nabla f(a)$. Consequently the gradient determines a vector field.

Directional derivatives

If f is a real-valued function on $\mathbf{R}^n$, then the partial derivatives of f measure its variation in the direction of the coordinate axes. For example, if f is a function of x and y, then its partial derivatives measure the variation in f in the x direction and the y direction. They do not, however, directly measure the variation of f in any other direction, such as along the diagonal line $y = x$. These are measured using directional derivatives. Choose a vector

$$\mathbf{v} = (v_1, \ldots, v_n).$$

The **directional derivative** of f in the direction of $\mathbf{v}$ at the point $\mathbf{x}$ is the limit

$$D_{\mathbf{v}} f(\mathbf{x}) = \lim_{h \to 0} \frac{f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x})}{h}.$$

In some cases it may be easier to compute or estimate the directional derivative after changing the length of the vector. Often this is done to turn the problem into the computation of a directional derivative in the direction of a unit vector. To see how this works, suppose that $\mathbf{v} = \lambda\mathbf{u}$. Substitute $h = k/\lambda$ into the difference quotient. The difference quotient becomes:

$$\frac{f(\mathbf{x}+(k/\lambda)(\lambda\mathbf{u}))-f(\mathbf{x})}{k/\lambda}=\lambda\cdot\frac{f(\mathbf{x}+k\mathbf{u})-f(\mathbf{x})}{k}.$$

This is λ times the difference quotient for the directional derivative of f with respect to $\mathbf{u}$. Furthermore, taking the limit as h tends to zero is the same as taking the limit as k tends to zero because h and k are multiples of each other. Therefore $D_{\mathbf{v}}(f) = \lambda D_{\mathbf{u}}(f)$. Because of this rescaling property, directional derivatives are frequently considered only for unit vectors.

If all the partial derivatives of f exist and are continuous at $\mathbf{x}$, then they determine the directional derivative of f in the direction $\mathbf{v}$ by the formula:

$$D_{\mathbf{v}}f(\boldsymbol{x})=\sum_{j=1}^{n}v_j\frac{\partial f}{\partial x_j}.$$

This is a consequence of the definition of the total derivative. It follows that the directional derivative is linear in $\mathbf{v}$, meaning that $D_{\mathbf{v}+\mathbf{w}}(f) = D_{\mathbf{v}}(f) + D_{\mathbf{w}}(f)$.

The same definition also works when f is a function with values in $\mathbf{R}^{\mathrm{m}}$. The above definition is applied to each component of the vectors. In this case, the directional derivative is a vector in $\mathbf{R}^{\mathrm{m}}$.

Total derivative, total differential and Jacobian matrix

When f is a function from an open subset of $\mathbf{R}^n$ to $\mathbf{R}^m$, then the directional derivative of f in a chosen direction is the best linear approximation to f at that point and in that direction. But when $n > 1$, no single directional derivative can give a complete picture of the behavior of f. The total derivative, also called the (**total**) **differential**, gives a complete picture by considering all directions at once. That is, for any vector $\mathbf{v}$ starting at $\mathbf{a}$, the linear approximation formula holds:

$$f(\mathbf{a}+\mathbf{v})\approx f(\mathbf{a})+f'(\mathbf{a})\mathbf{v}.$$

Just like the single-variable derivative, $f'(\mathbf{a})$ is chosen so that the error in this approximation is as small as possible.

If n and m are both one, then the derivative $f'(a)$ is a number and the expression $f'(a)v$ is the product of two numbers. But in higher dimensions, it is impossible for $f'(\mathbf{a})$ to be a number. If it were a number, then $f'(\mathbf{a})\mathbf{v}$ would be a vector in $\mathbf{R}^n$ while the other terms would be vectors in $\mathbf{R}^m$, and therefore the formula would not make sense. For the linear approximation formula to make sense, $f'(\mathbf{a})$ must be a function that sends vectors in $\mathbf{R}^n$ to vectors in $\mathbf{R}^m$, and $f'(\mathbf{a})\mathbf{v}$ must denote this function evaluated at $\mathbf{v}$.

To determine what kind of function it is, notice that the linear approximation formula can be rewritten as

$$f(\mathbf{a}+\mathbf{v})-f(\mathbf{a})\approx f'(\mathbf{a})\mathbf{v}.$$

Notice that if we choose another vector $\mathbf{w}$, then this approximate equation determines another approximate equation by substituting $\mathbf{w}$ for $\mathbf{v}$. It determines a third approximate equation by substituting both $\mathbf{w}$ for $\mathbf{v}$ and $\mathbf{a} + \mathbf{v}$ for $\mathbf{a}$. By subtracting these two new equations, we get

$$f(\mathbf{a}+\mathbf{v}+\mathbf{w})-f(\mathbf{a}+\mathbf{v})-f(\mathbf{a}+\mathbf{w})+f(\mathbf{a})\approx f'(\mathbf{a}+\mathbf{v})\mathbf{w}-f'(\mathbf{a})\mathbf{w}.$$

If we assume that $\mathbf{v}$ is small and that the derivative varies continuously in $\mathbf{a}$, then $f'(\mathbf{a} + \mathbf{v})$ is approximately equal to $f'(\mathbf{a})$, and therefore the right-hand side is approximately zero. The left-hand side can be rewritten in a different way using the linear approximation formula with $\mathbf{v} + \mathbf{w}$ substituted for $\mathbf{v}$. The linear approximation formula implies:

$$\begin{aligned}0&\approx f(\mathbf{a}+\mathbf{v}+\mathbf{w})-f(\mathbf{a}+\mathbf{v})-f(\mathbf{a}+\mathbf{w})+f(\mathbf{a})\\&=(f(\mathbf{a}+\mathbf{v}+\mathbf{w})-f(\mathbf{a}))-(f(\mathbf{a}+\mathbf{v})-f(\mathbf{a}))-(f(\mathbf{a}+\mathbf{w})-f(\mathbf{a}))\\&\approx f'(\mathbf{a})(\mathbf{v}+\mathbf{w})-f'(\mathbf{a})\mathbf{v}-f'(\mathbf{a})\mathbf{w}.\end{aligned}$$

This suggests that $f'(\mathbf{a})$ is a linear transformation from the vector space $\mathbf{R}^n$ to the vector space $\mathbf{R}^m$. In fact, it is possible to make this a precise derivation by measuring the error in the approximations. Assume that the error in these linear approximation formula is bounded by a constant times $\|\mathbf{v}\|$, where the constant is independent of $\mathbf{v}$ but

depends continuously on **a**. Then, after adding an appropriate error term, all of the above approximate equalities can be rephrased as inequalities. In particular, $f'(\mathbf{a})$ is a linear transformation up to a small error term. In the limit as **v** and **w** tend to zero, it must therefore be a linear transformation. Since we define the total derivative by taking a limit as **v** goes to zero, $f'(\mathbf{a})$ must be a linear transformation.

In one variable, the fact that the derivative is the best linear approximation is expressed by the fact that it is the limit of difference quotients. However, the usual difference quotient does not make sense in higher dimensions because it is not usually possible to divide vectors. In particular, the numerator and denominator of the difference quotient are not even in the same vector space: The numerator lies in the codomain $\mathbf{R}^m$ while the denominator lies in the domain $\mathbf{R}^n$. Furthermore, the derivative is a linear transformation, a different type of object from both the numerator and denominator. To make precise the idea that $f'(\mathbf{a})$ is the best linear approximation, it is necessary to adapt a different formula for the one-variable derivative in which these problems disappear. If $f : \mathbf{R} \to \mathbf{R}$, then the usual definition of the derivative may be manipulated to show that the derivative of f at a is the unique number $f'(a)$ such that

$$\lim_{h\to 0} \frac{f(a+h) - f(a) - f'(a)h}{h} = 0.$$

This is equivalent to

$$\lim_{h\to 0} \frac{|f(a+h) - f(a) - f'(a)h|}{|h|} = 0$$

because the limit of a function tends to zero if and only if the limit of the absolute value of the function tends to zero. This last formula can be adapted to the many-variable situation by replacing the absolute values with norms.

The definition of the **total derivative** of f at **a**, therefore, is that it is the unique linear transformation $f'(\mathbf{a}) : \mathbf{R}^n \to \mathbf{R}^m$ such that

$$\lim_{\|\mathbf{h}\|\to 0} \frac{\|f(\mathbf{a}+\mathbf{h}) - f(\mathbf{a}) - f'(\mathbf{a})\mathbf{h}\|}{\|\mathbf{h}\|} = 0.$$

Here **h** is a vector in $\mathbf{R}^n$, so the norm in the denominator is the standard length on $\mathbf{R}^n$. However, $f'(\mathbf{a})\mathbf{h}$ is a vector in $\mathbf{R}^m$, and the norm in the numerator is the standard length on $\mathbf{R}^m$. If v is a vector starting at a, then $f'(\mathbf{a})\mathbf{v}$ is called the pushforward of **v** by f and is sometimes written $f_*\mathbf{v}$.

If the total derivative exists at **a**, then all the partial derivatives and directional derivatives of f exist at **a**, and for all **v**, $f'(\mathbf{a})\mathbf{v}$ is the directional derivative of f in the direction **v**. If we write f using coordinate functions, so that $f = (f_1, f_2, \ldots, f_m)$, then the total derivative can be expressed using the partial derivatives as a matrix. This matrix is called the **Jacobian matrix** of f at **a**:

$$f'(\mathbf{a}) = \mathrm{Jac}_{\mathbf{a}} = \left(\frac{\partial f_i}{\partial x_j}\right)_{ij}.$$

The existence of the total derivative $f'(\mathbf{a})$ is strictly stronger than the existence of all the partial derivatives, but if the partial derivatives exist and are continuous, then the total derivative exists, is given by the Jacobian, and depends continuously on **a**.

The definition of the total derivative subsumes the definition of the derivative in one variable. That is, if f is a real-valued function of a real variable, then the total derivative exists if and only if the usual derivative exists. The Jacobian matrix reduces to a 1×1 matrix whose only entry is the derivative $f'(x)$. This 1×1 matrix satisfies the property that $f(a + h) - f(a) - f'(a)h$ is approximately zero, in other words that

$$f(a+h) \approx f(a) + f'(a)h.$$

Up to changing variables, this is the statement that the function $x \mapsto f(a) + f'(a)(x-a)$ is the best linear approximation to f at a.

The total derivative of a function does not give another function in the same way as the one-variable case. This is because the total derivative of a multivariable function has to record much more information than the derivative of a

single-variable function. Instead, the total derivative gives a function from the tangent bundle of the source to the tangent bundle of the target.

The natural analog of second, third, and higher-order total derivatives is not a linear transformation, is not a function on the tangent bundle, and is not built by repeatedly taking the total derivative. The analog of a higher-order derivative, called a jet, cannot be a linear transformation because higher-order derivatives reflect subtle geometric information, such as concavity, which cannot be described in terms of linear data such as vectors. It cannot be a function on the tangent bundle because the tangent bundle only has room for the base space and the directional derivatives. Because jets capture higher-order information, they take as arguments additional coordinates representing higher-order changes in direction. The space determined by these additional coordinates is called the jet bundle. The relation between the total derivative and the partial derivatives of a function is paralleled in the relation between the kth order jet of a function and its partial derivatives of order less than or equal to k.

Generalizations

The concept of a derivative can be extended to many other settings. The common thread is that the derivative of a function at a point serves as a linear approximation of the function at that point.

- An important generalization of the derivative concerns complex functions of complex variables, such as functions from (a domain in) the complex numbers $\mathbf{C}$ to $\mathbf{C}$. The notion of the derivative of such a function is obtained by replacing real variables with complex variables in the definition. If $\mathbf{C}$ is identified with $\mathbf{R}^2$ by writing a complex number z as $x + i\,y$, then a differentiable function from $\mathbf{C}$ to $\mathbf{C}$ is certainly differentiable as a function from $\mathbf{R}^2$ to $\mathbf{R}^2$ (in the sense that its partial derivatives all exist), but the converse is not true in general: the complex derivative only exists if the real derivative is *complex linear* and this imposes relations between the partial derivatives called the Cauchy Riemann equations — see holomorphic functions.
- Another generalization concerns functions between differentiable or smooth manifolds. Intuitively speaking such a manifold M is a space that can be approximated near each point x by a vector space called its tangent space: the prototypical example is a smooth surface in $\mathbf{R}^3$. The derivative (or differential) of a (differentiable) map $f\colon M \to N$ between manifolds, at a point x in M, is then a linear map from the tangent space of M at x to the tangent space of N at $f(x)$. The derivative function becomes a map between the tangent bundles of M and N. This definition is fundamental in differential geometry and has many uses — see pushforward (differential) and pullback (differential geometry).
- Differentiation can also be defined for maps between infinite dimensional vector spaces such as Banach spaces and Fréchet spaces. There is a generalization both of the directional derivative, called the Gâteaux derivative, and of the differential, called the Fréchet derivative.
- One deficiency of the classical derivative is that not very many functions are differentiable. Nevertheless, there is a way of extending the notion of the derivative so that all continuous functions and many other functions can be differentiated using a concept known as the weak derivative. The idea is to embed the continuous functions in a larger space called the space of distributions and only require that a function is differentiable "on average".
- The properties of the derivative have inspired the introduction and study of many similar objects in algebra and topology — see, for example, differential algebra.
- The discrete equivalent of differentiation is finite differences. The study of differential calculus is unified with the calculus of finite differences in time scale calculus.
- Also see arithmetic derivative.

Notes

[1] Differential calculus, as discussed in this article, is a very well-established mathematical discipline for which there are many sources. Almost all of the material in this article can be found in Apostol 1967, Apostol 1969, and Spivak 1994.

[2] Spivak 1994, chapter 10.

[3] See Differential (infinitesimal) for an overview. Further approaches include the Radon–Nikodym theorem, and the universal derivation (see Kähler differential).

[4] Despite this, it is still possible to take the derivative in the sense of distributions. The result is nine times the Dirac measure centered at *a*.

[5] Banach, S. (1931), "Uber die Baire'sche Kategorie gewisser Funktionenmengen", *Studia. Math.* (3): 174–179.. Cited by Hewitt, E and Stromberg, K (1963), *Real and abstract analysis*, Springer-Verlag, Theorem 17.8

[6] Apostol 1967, §4.18

[7] In the formulation of calculus in terms of limits, the *du* symbol has been assigned various meanings by various authors. Some authors do not assign a meaning to *du* by itself, but only as part of the symbol *du*/*dx*. Others define *dx* as an independent variable, and define *du* by $du = dx \cdot f'(x)$. In non-standard analysis *du* is defined as an infinitesimal. It is also interpreted as the exterior derivative of a function *u*. See differential (infinitesimal) for further information.

[8] (http://web.mit.edu/wwmath/calculus/differentiation/notation.html)

[9] This can also be expressed as the adjointness between the product space and function space constructions.

References

Print

- Anton, Howard; Bivens, Irl; Davis, Stephen (February 2, 2005), *Calculus: Early Transcendentals Single and Multivariable* (8th ed.), New York: Wiley, ISBN 978-0-471-47244-5
- Apostol, Tom M. (June 1967), *Calculus, Vol. 1: One-Variable Calculus with an Introduction to Linear Algebra*, **1** (2nd ed.), Wiley, ISBN 978-0-471-00005-1
- Apostol, Tom M. (June 1969), *Calculus, Vol. 2: Multi-Variable Calculus and Linear Algebra with Applications*, **1** (2nd ed.), Wiley, ISBN 978-0-471-00007-5
- Courant, Richard; John, Fritz (December 22, 1998), *Introduction to Calculus and Analysis, Vol. 1*, Springer-Verlag, ISBN 978-3-540-65058-4
- Eves, Howard (January 2, 1990), *An Introduction to the History of Mathematics* (6th ed.), Brooks Cole, ISBN 978-0-03-029558-4
- Larson, Ron; Hostetler, Robert P.; Edwards, Bruce H. (February 28, 2006), *Calculus: Early Transcendental Functions* (4th ed.), Houghton Mifflin Company, ISBN 978-0-618-60624-5
- Spivak, Michael (September 1994), *Calculus* (3rd ed.), Publish or Perish, ISBN 978-0-914098-89-8
- Stewart, James (December 24, 2002), *Calculus* (5th ed.), Brooks Cole, ISBN 978-0-534-39339-7
- Thompson, Silvanus P. (September 8, 1998), *Calculus Made Easy* (Revised, Updated, Expanded ed.), New York: St. Martin's Press, ISBN 978-0-312-18548-0

Online books

- Crowell, Benjamin (2003), *Calculus* (http://www.lightandmatter.com/calc/)
- Garrett, Paul (2004), *Notes on First-Year Calculus* (http://www.math.umn.edu/~garrett/calculus/), University of Minnesota
- Hussain, Faraz (2006), *Understanding Calculus* (http://www.understandingcalculus.com/)
- Keisler, H. Jerome (2000), *Elementary Calculus: An Approach Using Infinitesimals* (http://www.math.wisc.edu/~keisler/calc.html)
- Mauch, Sean (2004), *Unabridged Version of Sean's Applied Math Book* (http://www.its.caltech.edu/~sean/book/unabridged.html)
- Sloughter, Dan (2000), *Difference Equations to Differential Equations* (http://synechism.org/drupal/de2de/)
- Strang, Gilbert (1991), *Calculus* (http://ocw.mit.edu/ans7870/resources/Strang/strangtext.htm)

- Stroyan, Keith D. (1997), *A Brief Introduction to Infinitesimal Calculus* (http://www.math.uiowa.edu/~stroyan/InfsmlCalculus/InfsmlCalc.htm)
- Wikibooks, *Calculus* (http://en.wikibooks.org/wiki/Calculus)

Web pages

- Khan Academy: Derivative lesson 1 (http://www.khanacademy.org/video/calculus--derivatives-1--new-hd-version?playlist=Calculus)
- Weisstein, Eric W. " Derivative. (http://mathworld.wolfram.com/Derivative.html)" From MathWorld
- Derivatives of Trigonometric functions (http://www.ugrad.math.ubc.ca/coursedoc/math100/notes/derivative/trig2.html), UBC
- Solved Problems in Derivatives (http://calculus.solved-problems.com/category/derivative/)

Rate of convergence

In numerical analysis, the speed at which a convergent sequence approaches its limit is called the **rate of convergence**. Although strictly speaking, a limit does not give information about any finite first part of the sequence, this concept is of practical importance if we deal with a sequence of successive approximations for an iterative method, as then typically fewer iterations are needed to yield a useful approximation if the rate of convergence is higher. This may even make the difference between needing ten or a million iterations.

Similar concepts are used for discretization methods. The solution of the discretized problem converges to the solution of the continuous problem as the grid size goes to zero, and the speed of convergence is one of the factors of the efficiency of the method. However, the terminology in this case is different from the terminology for iterative methods.

Series acceleration is a collection of techniques for improving the rate of convergence of a series discretization . Such acceleration is commonly accomplished with sequence transformations.

Convergence speed for iterative methods

Basic definition

Suppose that the sequence $\{x_k\}$ converges to the number L.

We say that this sequence **converges linearly** to L, if there exists a number $\mu \in (0, 1)$ such that

\lim_{k\to \infty} \frac{|x_{k+1}-L|}{|x_k-L|} = \mu.

The number μ is called the *rate of convergence*.

If the sequences converges, and

- $\mu = 0$, then the sequence is said to **converge superlinearly**.
- $\mu = 1$, then the sequence is said to **converge sublinearly**.

If the sequences *converges sublinearly* and additionally

$$\lim_{k\to\infty} \frac{|x_{k+2} - x_{k+1}|}{|x_{k+1} - x_k|} = 1,$$

then it is said the sequence $\{x_k\}$ **converges logarithmically** to L.

The next definition is used to distinguish superlinear rates of convergence. We say that the sequence **converges with order *q* to L** for $q>1$[1] if

$$\lim_{k\to\infty} \frac{|x_{k+1} - L|}{|x_k - L|^q} = \mu \,|\, \mu > 0.$$

In particular, convergence with order

- 2 is called **quadratic convergence**,
- 3 is called **cubic convergence**,
- etc.

This is sometimes called *Q-linear convergence*, *Q-quadratic convergence*, etc., to distinguish it from the definition below. The Q stands for "quotient," because the definition uses the quotient between two successive terms.

Extended definition

The drawback of the above definitions is that these do not catch some sequences which still converge reasonably fast, but whose "speed" is variable, such as the sequence $\{b_k\}$ below. Therefore, the definition of rate of convergence is sometimes extended as follows.

Under the new definition, the sequence $\{x_k\}$ converges with at least order q if there exists a sequence $\{\varepsilon_k\}$ such that

$$|x_k - L| \leq \varepsilon_k^q \quad \text{for all } k,$$

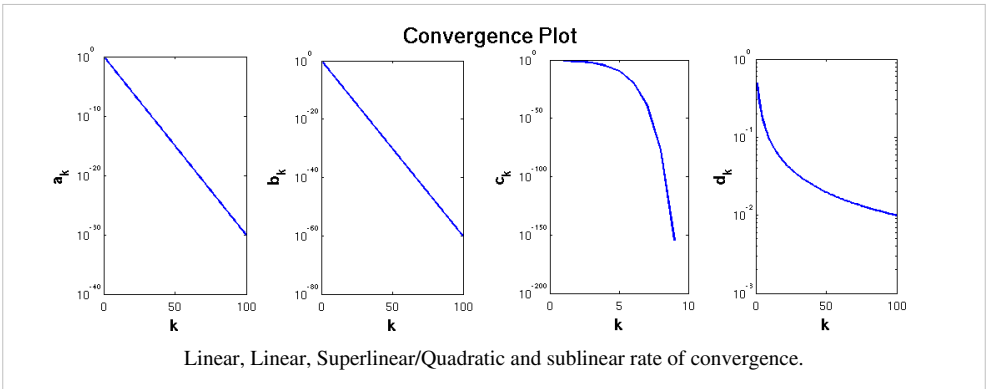
and the sequence $\{\varepsilon_k\}$ converges to zero with order q according to the above "simple" definition. To distinguish it from that definition, this is sometimes called *R-linear convergence*, *R-quadratic convergence*, etc. (with the R standing for "root").

Examples

Consider the following sequences:

$$\begin{array}{lllllllll}
a_0 = 1, & a_1 = \frac{1}{2}, & a_2 = \frac{1}{4}, & a_3 = \frac{1}{8}, & a_4 = \frac{1}{16}, & a_5 = \frac{1}{32}, & \ldots, & a_k = \frac{1}{2^k}, & \ldots \\
b_0 = 1, & b_1 = 1, & b_2 = \frac{1}{4}, & b_3 = \frac{1}{4}, & b_4 = \frac{1}{16}, & b_5 = \frac{1}{16}, & \ldots, & b_k = \frac{1}{4^{\lfloor \frac{k}{2} \rfloor}}, & \ldots \\
c_0 = \frac{1}{2}, & c_1 = \frac{1}{4}, & c_2 = \frac{1}{16}, & c_3 = \frac{1}{256}, & c_4 = \frac{1}{65\,536}, & & \ldots, & c_k = \frac{1}{2^{2^k}}, & \ldots \\
d_0 = 1, & d_1 = \frac{1}{2}, & d_2 = \frac{1}{3}, & d_3 = \frac{1}{4}, & d_4 = \frac{1}{5}, & d_5 = \frac{1}{6}, & \ldots, & d_k = \frac{1}{k+1}, & \ldots
\end{array}$$

The sequence $\{a_k\}$ converges linearly to 0 with rate 1/2. More generally, the sequence $C\mu^k$ converges linearly with rate μ if |μ| < 1. The sequence $\{b_k\}$ also converges linearly to 0 with rate 1/2 under the extended definition, but not under the simple definition. The sequence $\{c_k\}$ converges superlinearly. In fact, it is quadratically convergent. Finally, the sequence $\{d_k\}$ converges sublinearly.



Linear, Linear, Superlinear/Quadratic and sublinear rate of convergence.

Convergence speed for discretization methods

A similar situation exists for discretization methods. The important parameter here for the convergence speed is not the iteration number k but it depends on the number of grid points and grid spacing. In this case, the number of grid points in a discretization process is inversely proportional to the grid spacing here it denoted as n.

In this case, a sequence x_n is said to converge to L with order p if there exists a constant C such that

$$|x_n - L| < Cn^{-p} \text{ for all } n.$$

This is written as $|x_n - L| = O(n^{-p})$ using the big O notation.

This is the relevant definition when discussing methods for numerical quadrature or the solution of ordinary differential equations.

Examples

The sequence $\{d_k\}$ with $d_k = 1 / (k+1)$ was introduced above. This sequence converges with order 1 according to the convention for discretization methods.

The sequence $\{a_k\}$ with $a_k = 2^{-k}$, which was also introduced above, converges with order p for every number p. It is said to converge exponentially using the convention for discretization methods. However, it only converges linearly (that is, with order 1) using the convention for iterative methods.

Acceleration of convergence

Many methods exist to increase the rate of convergence of a given sequence, i.e. to transform a given sequence into one converging faster to the same limit. Such techniques are in general known as "series acceleration". The goal of the transformed sequence is to be much less "expensive" to calculate than the original sequence. One example of series acceleration is Aitken's delta-squared process.

References

[1] q may be non-integer. For example, the secant method has, in the case of convergence to a regular root, convergence order $\varphi=1.681...$.

Literature

The simple definition is used in

- Michelle Schatzman (2002), *Numerical analysis: a mathematical introduction*, Clarendon Press, Oxford. ISBN 0-19-850279-6.

The extended definition is used in

- Kendell A. Atkinson (1988), *An introduction to numerical analysis* (2nd ed.), John Wiley and Sons. ISBN 0-471-50023-2.
- http://web.mit.edu/rudin/www/MukherjeeRuSc11COLT.pdf
- Walter Gautschi (1997), *Numerical analysis: an introduction,* Birkhäuser, Boston. ISBN 0-8176-3895-4.
- Endre Süli and David Mayers (2003), *An introduction to numerical analysis,* Cambridge University Press. ISBN 0-521-00794-1.

Logarithmic convergence is used in

- Van Tuyl, Andrew H. (1994). "Acceleration of convergence of a family of logarithmically convergent sequences" (http://www.ams.org/journals/mcom/1994-63-207/S0025-5718-1994-1234428-2/S0025-5718-1994-1234428-2.pdf). *Mathematics of Computation* **63** (207): 229–246.

The Big O definition is used in

- Richard L. Burden and J. Douglas Faires (2001), *Numerical Analysis* (7th ed.), Brooks/Cole. ISBN 0-534-38216-9

The terms *Q-linear* and *R-linear* are used in; The Big O definition when using Taylor series is used in

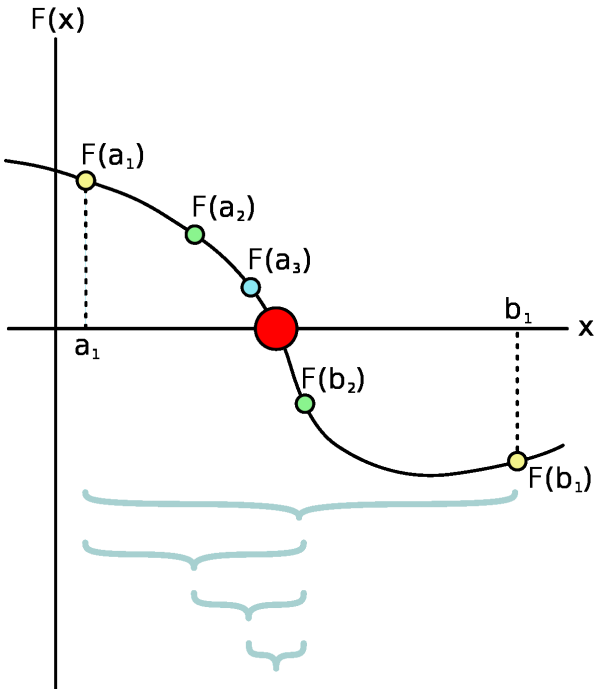
- Nocedal, Jorge; Wright, Stephen J. (2006). *Numerical Optimization* (2nd ed.). Berlin, New York: Springer-Verlag. pp. 619+620. ISBN 978-0-387-30303-1.

One may also study the following paper for Q-linear and R-linear:

- Potra, F. A. (1989). "On Q-order and R-order of convergence". *J. Optim. Th. Appl.* **63** (3): 415–431. doi:10.1007/BF00939805.

Bisection method

The **bisection method** in mathematics is a root-finding method which repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. It is a very simple and robust method, but it is also relatively slow. Because of this, it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods.[1] The method is also called the **binary search method**[2]



A few steps of the bisection method applied over the starting range $[a_1;b_1]$. The bigger red dot is the root of the function.

The method

The method is applicable when we wish to solve the equation $f(x) = 0$ for the real variable x, where f is a continuous function defined on an interval $[a, b]$ and $f(a)$ and $f(b)$ have opposite signs. In this case a and b are said to bracket a root since, by the intermediate value theorem, the f must have at least one root in the interval (a, b).

At each step the method divides the interval in two by computing the midpoint $c = (a+b) / 2$ of the interval and the value of the function $f(c)$ at that point. Unless c is itself a root (which is very unlikely, but possible) there are now two possibilities: either $f(a)$ and $f(c)$ have opposite signs and bracket a root, or $f(c)$ and $f(b)$ have opposite signs and bracket a root. The method selects the subinterval that is a bracket as a new interval to be used in the next step. In this way the interval that contains a zero of f is reduced in width by 50% at each step. The process is continued until the interval is sufficiently small.

Explicitly, if $f(a)$ and $f(c)$ are opposite signs, then the method sets c as the new value for b, and if $f(b)$ and $f(c)$ are opposite signs then the method sets c as the new a. (If $f(c)=0$ then c may be taken as the solution and the process stops.) In both cases, the new $f(a)$ and $f(b)$ have opposite signs, so the method is applicable to this smaller interval.[3]

Analysis

The method is guaranteed to converge to a root of *f* if *f* is a continuous function on the interval [*a*, *b*] and *f*(*a*) and *f*(*b*) have opposite signs. The absolute error is halved at each step so the method converges linearly, which is comparatively slow.

Specifically, if $p_1 = (a+b)/2$ is the midpoint of the initial interval, and p_n is the midpoint of the interval in the *n*th step, then the difference between p_n and a solution *p* is bounded by[4]

$$|p_n - p| \leq \frac{|b-a|}{2^n}.$$

This formula can be used to determine in advance the number of iterations that the bisection method would need to converge to a root to within a certain tolerance.

Pseudocode

The method may be written in Pseudocode as follows:[5]

```
INPUT: Function f, endpoint values a, b, tolerance TOL, maximum iterations NMAX
CONDITIONS: a < b, either f(a) < 0 and f(b) > 0 or f(a) > 0 and f(b) < 0
OUTPUT: value which differs from a root of f(x)=0 by less than TOL

N ← 1
While N ≤ NMAX { limit iterations to prevent infinite loop
  c ← (a + b)/2 new midpoint
  If (f(c) = 0 or (b − a)/2 < TOL then { solution found
    Output(c)
    Stop
  }
  N ← N + 1 increment step counter
  If sign(f(c)) = sign(f(a)) then a ← c else b ← c new interval
}
Output("Method failed.") max number of steps exceeded
```

References

[1] Burden & Faires 1985, p. 31
[2] Burden & Fairies 1985, p. 28
[3] Burden & Faires 1985, p. 28 for section
[4] Burden & Faires 1985, p. 31, Theorem 2.1
[5] Burden & Faires 1985, p. 29

- Burden, Richard L.; Faires, J. Douglas (1985), "2.1 The Bisection Algorithm", *Numerical Analysis* (3rd ed.), PWS Publishers, ISBN 0-87150-857-5.
- Corliss, George (1977), "Which root does the bisection algorithm find?", *SIAM Review* **19** (2): 325–327, doi:10.1137/1019044, ISSN 1095-7200.
- Kaw, Autar; Kalu, Egwu (2008), *Numerical Methods with Applications* (http://numericalmethods.eng.usf.edu/topics/textbook_index.html) (1st ed.)

External links

- Weisstein, Eric W., " Bisection (http://mathworld.wolfram.com/Bisection.html)" from MathWorld.
- (http://www.torkian.info/Site/Research/Entries/2008/2/28_Root-finding_algorithm_Java_Code_(_Secant,_Bisection,_Newton_).html) Java code by Behzad Torkian
- Bisection Method (http://numericalmethods.eng.usf.edu/topics/bisection_method.html) Notes, PPT, Mathcad, Maple, Matlab, Mathematica from Holistic Numerical Methods Institute (http://numericalmethods.eng.usf.edu)
- Module for the Bisection Method by John H. Mathews (http://math.fullerton.edu/mathews/n2003/BisectionMod.html)
- Online root finding of a polynomial-Bisection method (http://catc.ac.ir/mazlumi/jscodes/bisection.php) by Farhad Mazlumi

Newton's method

In numerical analysis, **Newton's method** (also known as the **Newton–Raphson method**), named after Isaac Newton and Joseph Raphson, is a method for finding successively better approximations to the roots (or zeroes) of a real-valued function. The algorithm is first in the class of Householder's methods, succeeded by Halley's method. The method can also be extended to complex functions and to systems of equations.

The Newton-Raphson method in one variable is implemented as follows:

Given a function f defined over the reals x, and its derivative f', we begin with a first guess x_0 for a root of the function f. Provided the function is reasonably well-behaved a better approximation x_1 is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

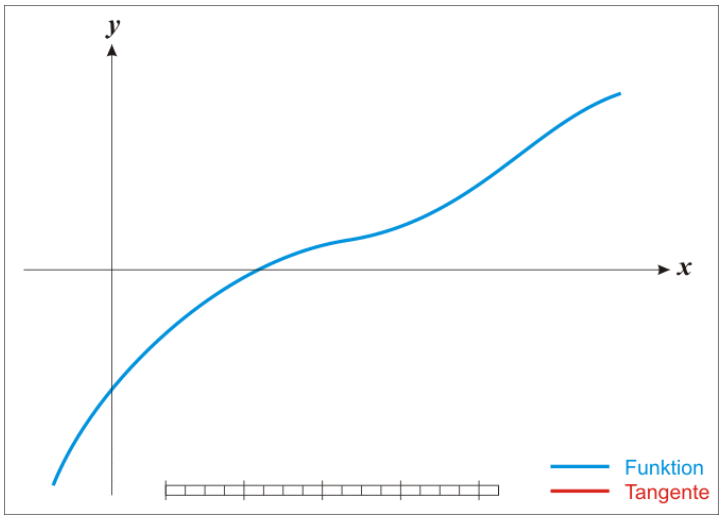
Geometrically, $(x_1, 0)$ is the intersection with the x-axis of a line tangent to f at $(x_0, f(x_0))$.

The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently accurate value is reached.

Description



The function f is shown in blue and the tangent line is in red. We see that x_{n+1} is a better approximation than x_n for the root x of the function f.

The idea of the method is as follows: one starts with an initial guess which is reasonably close to the true root, then the function is approximated by its tangent line (which can be computed using the tools of calculus), and one computes the x-intercept of this tangent line (which is easily done with elementary algebra). This x-intercept will typically be a better approximation to the function's root than the original guess, and the method can be iterated.

Suppose $f : [a, b] \to \mathbf{R}$ is a differentiable function defined on the interval $[a, b]$ with values in the real numbers $\mathbf{R}$. The formula for converging on the root can be easily derived. Suppose we have some current approximation x_n. Then we can derive the formula for a better approximation, x_{n+1} by referring to the diagram on the right. We know from the definition of the derivative at a given point that it is the slope of a tangent at that point.

That is

$$f'(x_n) = \frac{\Delta y}{\Delta x} = \frac{f(x_n) - 0}{x_n - x_{n+1}}.$$

Here, f' denotes the derivative of the function f. Then by simple algebra we can derive

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

We start the process off with some arbitrary initial value x_0. (The closer to the zero, the better. But, in the absence of any intuition about where the zero might lie, a "guess and check" method might narrow the possibilities to a reasonably small interval by appealing to the intermediate value theorem.) The method will usually converge, provided this initial guess is close enough to the unknown zero, and that $f'(x_0) \neq 0$. Furthermore, for a zero of multiplicity 1, the convergence is at least quadratic (see rate of convergence) in a neighbourhood of the zero, which intuitively means that the number of correct digits roughly at least doubles in every step. More details can be found in the analysis section below.

The Householder's methods are similar but have higher order for even faster convergence. However, the extra computations required for each step can slow down the overall performance relative to Newton's method, particularly if f or its derivatives are computationally expensive to evaluate.

History

Newton's method was described by Isaac Newton in *De analysi per aequationes numero terminorum infinitas* (written in 1669, published in 1711 by William Jones) and in *De metodis fluxionum et serierum infinitarum* (written in 1671, translated and published as *Method of Fluxions* in 1736 by John Colson). However, his description differs substantially from the modern description given above: Newton applies the method only to polynomials. He does not compute the successive approximations x_n, but computes a sequence of polynomials and only at the end, he arrives at an approximation for the root x. Finally, Newton views the method as purely algebraic and fails to notice the connection with calculus. Isaac Newton probably derived his method from a similar but less precise method by

Vieta. The essence of Vieta's method can be found in the work of the Persian mathematician, Sharaf al-Din al-Tusi, while his successor Jamshīd al-Kāshī used a form of Newton's method to solve $x^P - N = 0$ to find roots of N (Ypma 1995). A special case of Newton's method for calculating square roots was known much earlier and is often called the Babylonian method.

Newton's method was used by 17th century Japanese mathematician Seki Kōwa to solve single-variable equations, though the connection with calculus was missing.

Newton's method was first published in 1685 in *A Treatise of Algebra both Historical and Practical* by John Wallis. In 1690, Joseph Raphson published a simplified description in *Analysis aequationum universalis*. Raphson again viewed Newton's method purely as an algebraic method and restricted its use to polynomials, but he describes the method in terms of the successive approximations x_n instead of the more complicated sequence of polynomials used by Newton. Finally, in 1740, Thomas Simpson described Newton's method as an iterative method for solving general nonlinear equations using fluxional calculus, essentially giving the description above. In the same publication, Simpson also gives the generalization to systems of two equations and notes that Newton's method can be used for solving optimization problems by setting the gradient to zero.

Arthur Cayley in 1879 in *The Newton-Fourier imaginary problem* was the first who noticed the difficulties in generalizing the Newton's method to complex roots of polynomials with degree greater than 2 and complex initial values. This opened the way to the study of the theory of iterations of rational functions.

Practical considerations

Newton's method is an extremely powerful technique—in general the convergence is quadratic: as the method converges on the root, the difference between the root and the approximation is squared (the number of accurate digits roughly doubles) at each step. However, there are some difficulties with the method.

Difficulty in calculating derivative of a function

Newton's method requires that the derivative be calculated directly. An analytical expression for the derivative may not be easily obtainable and could be expensive to evaluate. In these situations, it may be appropriate to approximate the derivative by using the slope of a line through two nearby points on the function. Using this approximation would result in something like the secant method whose convergence is slower than that of Newton's method.

Failure of the method to converge to the root

It is important to review the proof of quadratic convergence of Newton's Method before implementing it. Specifically, one should review the assumptions made in the proof. For situations where the method fails to converge, it is because the assumptions made in this proof are not met.

Overshoot

If the first derivative is not well behaved in the neighborhood of a particular root, the method may overshoot, and diverge from that root.

Stationary point

If a stationary point of the function is encountered, the derivative is zero and the method will terminate due to division by zero.

Poor initial estimate

A large error in the initial estimate can contribute to non-convergence of the algorithm.

Mitigation of non-convergence

In a robust implementation of Newton's method, it is common to place limits on the number of iterations, bound the solution to an interval known to contain the root, and combine the method with a more robust root finding method.

Slow convergence for roots of multiplicity > 1

If the root being sought has multiplicity greater than one, the convergence rate is merely linear (errors reduced by a constant factor at each step) unless special steps are taken. When there are two or more roots that are close together then it may take many iterations before the iterates get close enough to one of them for the quadratic convergence to be apparent. However, if the multiplicity m of the root is known, one can use the following modified algorithm that preserves the quadratic convergence rate:

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}.$$ [1]

Analysis

Suppose that the function f has a zero at α, i.e., $f(\alpha) = 0$.

If f is continuously differentiable and its derivative is nonzero at α, then there exists a neighborhood of α such that for all starting values x_0 in that neighborhood, the sequence $\{x_n\}$ will converge to α.

If the function is continuously differentiable and its derivative is not 0 at α and it has a second derivative at α then the convergence is quadratic or faster. If the second derivative is not 0 at α then the convergence is merely quadratic. If the third derivative exists and is bounded in a neighborhood of α, then:

$$\Delta x_{i+1} = \frac{f''(\alpha)}{2f'(\alpha)}(\Delta x_i)^2 + O[\Delta x_i]^3,$$

where $\Delta x_i \triangleq x_i - \alpha$.

If the derivative is 0 at α, then the convergence is usually only linear. Specifically, if f is twice continuously differentiable, $f'(\alpha) = 0$ and $f''(\alpha) \neq 0$, then there exists a neighborhood of α such that for all starting values x_0 in that neighborhood, the sequence of iterates converges linearly, with rate $\log_{10} 2$ (Süli & Mayers, Exercise 1.6). Alternatively if $f'(\alpha) = 0$ and $f'(x) \neq 0$ for $x \neq 0$, x in a neighborhood U of α, α being a zero of multiplicity r, and if $f \in C^r(U)$ then there exists a neighborhood of α such that for all starting values x_0 in that neighborhood, the sequence of iterates converges linearly.

However, even linear convergence is not guaranteed in pathological situations.

In practice these results are local, and the neighborhood of convergence is not known in advance. But there are also some results on global convergence: for instance, given a right neighborhood U_+ of α, if f is twice differentiable in U_+ and if $f' \neq 0,\ f \cdot f'' > 0$ in U_+, then, for each x_0 in U_+ the sequence x_k is monotonically decreasing to α.

Proof of quadratic convergence for Newton's iterative method

According to Taylor's theorem, any function *f*(*x*) which has a continuous second derivative can be represented by an expansion about a point that is close to a root of f(x). Suppose this root is α. Then the expansion of f(α) about x_n is:

$$f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + R_1 \quad \textbf{(1)}$$

where the Lagrange form of the Taylor series expansion remainder is

$$R_1 = \frac{1}{2!} f''(\xi_n)(\alpha - x_n)^2,$$

where ξ_n is in between x_n and α.

Since α is the root, (**1**) becomes:

$$0 = f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + \frac{1}{2} f''(\xi_n)(\alpha - x_n)^2 \quad \textbf{(2)}$$

Dividing equation (**2**) by $f'(x_n)$ and rearranging gives

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = \frac{-f''(\xi_n)}{2f'(x_n)} (\alpha - x_n)^2 \quad \textbf{(3)}$$

Remembering that x_{n+1} is defined by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad \textbf{(4)}$$

one finds that

$$\underbrace{\alpha - x_{n+1}}_{\epsilon_{n+1}} = \frac{-f''(\xi_n)}{2f'(x_n)} (\underbrace{\alpha - x_n}_{\epsilon_n})^2.$$

That is,

$$\epsilon_{n+1} = \frac{-f''(\xi_n)}{2f'(x_n)} {\epsilon_n}^2. \quad \textbf{(5)}$$

Taking absolute value of both sides gives

$$|\epsilon_{n+1}| = \frac{|f''(\xi_n)|}{2|f'(x_n)|} {\epsilon_n}^2 \quad \textbf{(6)}$$

Equation (**6**) shows that the rate of convergence is quadratic if following conditions are satisfied:

1. $f'(x) \neq 0; \forall x \in I$, where I is the interval $[\alpha - r, \alpha + r]$ for some $r \geq |(\alpha - x_0)|$;
2. $f''(x)$ is finite , $\forall x \in I$;
3. x_0 *sufficiently* close to the root α

The term *sufficiently* close in this context means the following:

(a) Taylor approximation is accurate enough such that we can ignore higher order terms,

(b) $\frac{1}{2} \left| \frac{f''(x_n)}{f'(x_n)} \right| < C \left| \frac{f''(\alpha)}{f'(\alpha)} \right|$, for some $C < \infty$,

(c) $C \left| \frac{f''(\alpha)}{f'(\alpha)} \right| \epsilon_n < 1$, for $n \in \mathbb{Z}^+ \cup \{0\}$ and C satisfying condition (b).

Finally, (**6**) can be expressed in the following way:

$$|\epsilon_{n+1}| \leq M{\epsilon^2}_n$$

where M is the supremum of the variable coefficient of ${\epsilon^2}_n$ on the interval I defined in the condition 1, that is:

$$M = \sup_{x \in I} \frac{1}{2} \left| \frac{f''(x)}{f'(x)} \right|.$$

The initial point x_0 has to be chosen such that conditions 1 through 3 are satisfied, where the third condition requires that $M\,|\epsilon_0| < 1.$

Basins of attraction

The basins of attraction—the regions of the real number line such that within each region iteration from any point leads to one particular root—can be infinite in number and arbitrarily small. For example,[2] for the function $f(x) = x^3 - 2x^2 - 11x + 12$, the following initial conditions are in successive basins of attraction:

2.35287527 converges to 4;

2.35284172 converges to −3;

2.35283735 converges to 4;

2.352836327 converges to −3;

2.352836323 converges to 1.

Failure analysis

Newton's method is only guaranteed to converge if certain conditions are satisfied. If the assumptions made in the proof of quadratic convergence are met, the method will converge. For the following subsections, failure of the method to converge indicates that the assumptions made in the proof were not met.

Bad starting points

In some cases the conditions on the function that are necessary for convergence are satisfied, but the point chosen as the initial point is not in the interval where the method converges. This can happen, for example, if the function whose root is sought approaches zero asymptotically as *x* goes to ∞ or $-\infty$. In such cases a different method, such as bisection, should be used to obtain a better estimate for the zero to use as an initial point.

Iteration point is stationary

Consider the function:

$$f(x) = 1 - x^2.$$

It has a maximum at *x*=0 and solutions of *f(x)* = 0 at *x* = ±1. If we start iterating from the stationary point x_0=0 (where the derivative is zero), x_1 will be undefined, since the tangent at (0,1) is parallel to the *x*-axis:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0 - \frac{1}{0}.$$

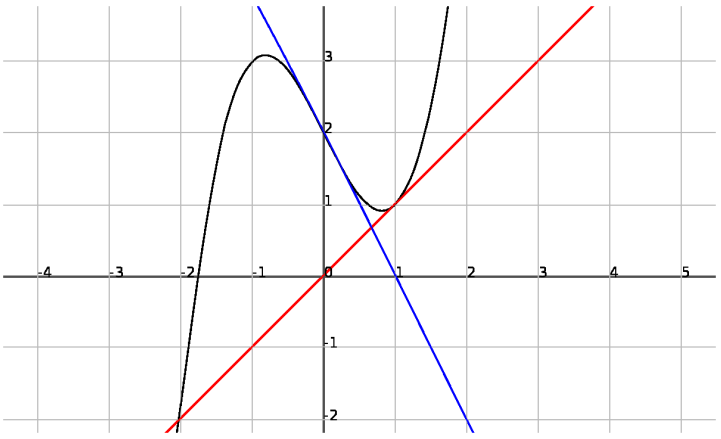
The same issue occurs if, instead of the starting point, any iteration point is stationary. Even if the derivative is small but not zero, the next iteration will be a far worse approximation.

Starting point enters a cycle

For some functions, some starting points may enter an infinite cycle, preventing convergence. Let

$$f(x) = x^3 - 2x + 2$$

and take 0 as the starting point. The first iteration produces 1 and the second iteration returns to 0 so the sequence will alternate between the two without converging to a root. In fact, this 2-cycle is stable: there are neighborhoods around 0 and around 1 from which all points iterate asymptotically to the 2-cycle (and hence not to the root of the function). In general, the behavior of the sequence can be very complex (see Newton fractal).



The tangent lines of $x^3 - 2x + 2$ at 0 and 1 intersect the x-axis at 1 and 0 respectively, illustrating why Newton's method oscillates between these values for some starting points.

Derivative issues

If the function is not continuously differentiable in a neighborhood of the root then it is possible that Newton's method will always diverge and fail, unless the solution is guessed on the first try.

Derivative does not exist at root

A simple example of a function where Newton's method diverges is the cube root, which is continuous and infinitely differentiable, except for $x = 0$, where its derivative is undefined (this, however, does not affect the algorithm, since it will never require the derivative if the solution is already found):

$$f(x) = \sqrt[3]{x}.$$

For any iteration point x_n, the next iteration point will be:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^{\frac{1}{3}}}{\frac{1}{3}x_n^{\frac{1}{3}-1}} = x_n - 3\,x_n = -2\,x_n.$$

The algorithm overshoots the solution and lands on the other side of the y-axis, farther away than it initially was; applying Newton's method actually doubles the distances from the solution at each iteration.

In fact, the iterations diverge to infinity for every $f(x) = |x|^\alpha$, where $0 < \alpha < \frac{1}{2}$. In the limiting case of $\alpha = \frac{1}{2}$(square root), the iterations will alternate indefinitely between points x_0 and $-x_0$, so they do not converge in this case either.

Discontinuous derivative

If the derivative is not continuous at the root, then convergence may fail to occur in any neighborhood of the root. Consider the function

$$f(x) = \begin{cases} 0 & \text{if } x = 0, \\ x + x^2 \sin\left(\frac{2}{x}\right) & \text{if } x \neq 0. \end{cases}$$

Its derivative is:

$$f'(x) = \begin{cases} 1 & \text{if } x = 0, \\ 1 + 2x \sin\left(\frac{2}{x}\right) - 2\cos\left(\frac{2}{x}\right) & \text{if } x \neq 0. \end{cases}$$

Within any neighborhood of the root, this derivative keeps changing sign as x approaches 0 from the right (or from the left) while $f(x) \geq x - x^2 > 0$ for $0 < x < 1$.

So $f(x)/f'(x)$ is unbounded near the root, and Newton's method will diverge almost everywhere in any neighborhood of it, even though:

- the function is differentiable (and thus continuous) everywhere;
- the derivative at the root is nonzero;
- f is infinitely differentiable except at the root; and
- the derivative is bounded in a neighborhood of the root (unlike $f(x)/f'(x)$).

Non-quadratic convergence

In some cases the iterates converge but do not converge as quickly as promised. In these cases simpler methods converge just as quickly as Newton's method.

Zero derivative

If the first derivative is zero at the root, then convergence will not be quadratic. Indeed, let

$$f(x) = x^2$$

then $f'(x) = 2x$ and consequently $x - f(x)/f'(x) = x/2$. So convergence is not quadratic, even though the function is infinitely differentiable everywhere.

Similar problems occur even when the root is only "nearly" double. For example, let

$$f(x) = x^2(x - 1000) + 1.$$

Then the first few iterates starting at $x_0 = 1$ are 1, 0.500250376, 0.251062828, 0.127507934, 0.067671976, 0.041224176, 0.032741218, 0.031642362; it takes six iterations to reach a point where the convergence appears to be quadratic.

No second derivative

If there is no second derivative at the root, then convergence may fail to be quadratic. Indeed, let

$$f(x) = x + x^{\frac{4}{3}}.$$

Then

$$f'(x) = 1 + \frac{4}{3}x^{\frac{1}{3}}.$$

And

$$f''(x) = \frac{4}{9}x^{\frac{-2}{3}}$$

except when $x = 0$ where it is undefined. Given x_n,

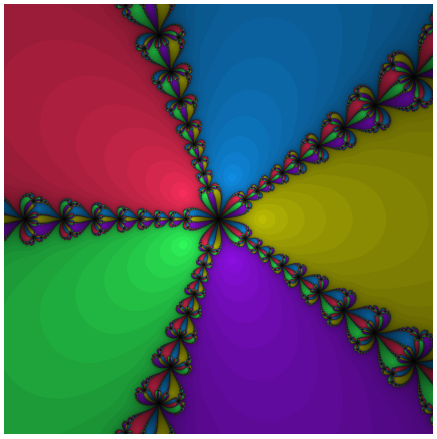
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = \frac{\frac{1}{3}x_n{}^{\frac{4}{3}}}{(1 + \frac{4}{3}x_n{}^{\frac{1}{3}})}$$

which has approximately 4/3 times as many bits of precision as x_n has. This is less than the 2 times as many which would be required for quadratic convergence. So the convergence of Newton's method (in this case) is not quadratic, even though: the function is continuously differentiable everywhere; the derivative is not zero at the root; and f is infinitely differentiable except at the desired root.

Generalizations

Complex functions

When dealing with complex functions, Newton's method can be directly applied to find their zeroes. Each zero has a basin of attraction in the complex plane, the set of all starting values that cause the method to converge to that particular zero. These sets can be mapped as in the image shown. For many complex functions, the boundaries of the basins of attraction are fractals.



Basins of attraction for $x^5 - 1 = 0$; darker means more iterations to converge.

In some cases there are regions in the complex plane which are not in any of these basins of attraction, meaning the iterates do not converge. For example,[3] if one uses a real initial condition to seek a root of $x^2 + 1$, all subsequent iterates will be real numbers and so the iterations cannot converge to either root, since both roots are non-real. In this case almost all initial conditions lead to chaotic behavior, while some initial conditions iterate either to infinity or to repeating cycles of any finite length.

Nonlinear systems of equations

k variables, k functions

One may also use Newton's method to solve systems of k (non-linear) equations, which amounts to finding the zeroes of continuously differentiable functions $F : \mathbf{R}^k \to \mathbf{R}^k$. In the formulation given above, one then has to left multiply with the inverse of the k-by-k Jacobian matrix $J_F(x_n)$ instead of dividing by $f'(x_n)$.

Rather than actually computing the inverse of this matrix, one can save time by solving the system of linear equations

$$J_F(x_n)(x_{n+1} - x_n) = -F(x_n)$$

for the unknown $x_{n+1} - x_n$.

k variables, > k equations

The k-dimensional Newton's method can be used to solve systems of >*k* (non-linear) equations as well if the algorithm uses the generalized inverse of the non-square Jacobian matrix $J^+ = ((J^T J)^{-1})J^T$ instead of the inverse of J. If the nonlinear system has no solution, the methods attempts to find a solution in the non-linear least squares sense. See Gauss–Newton algorithm for more information.

Nonlinear equations in a Banach space

Another generalization is Newton's method to find a root of a functional *F* defined in a Banach space. In this case the formulation is

$$X_{n+1} = X_n - [F'(X_n)]^{-1} F(X_n),$$

where $F'(X_n)$ is the Fréchet derivative computed at X_n. One needs the Fréchet derivative to be boundedly invertible at each X_n in order for the method to be applicable. A condition for existence of and convergence to a root is given by the Newton–Kantorovich theorem.

Nonlinear equations over *p*-adic numbers

In *p*-adic analysis, the standard method to show a polynomial equation in one variable has a *p*-adic root is Hensel's lemma, which uses the recursion from Newton's method on the *p*-adic numbers. Because of the more stable behavior of addition and multiplication in the *p*-adic numbers compared to the real numbers (specifically, the unit ball in the *p*-adics is a ring), convergence in Hensel's lemma can be guaranteed under much simpler hypotheses than in the classical Newton's method on the real line.

Applications

Minimization and maximization problems

Newton's method can be used to find a minimum or maximum of a function. The derivative is zero at a minimum or maximum, so minima and maxima can be found by applying Newton's method to the derivative. The iteration becomes:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

Digital division

An important and somewhat surprising application is Newton–Raphson division, which can be used to quickly find the reciprocal of a number using only multiplication and subtraction.

Solving transcendental equations

Many transcendental equations can be solved using Newton's method. Given the equation

$$g(x) = h(x),$$

with *g(x)* and/or *h(x)* a transcendental function, one writes

$$f(x) = g(x) - h(x).$$

The values of *x* that solves the original equation are then the roots of *f(x)*, which may be found via Newton's method.

Examples

Square root of a number

Consider the problem of finding the square root of a number. There are many methods of computing square roots, and Newton's method is one.

For example, if one wishes to find the square root of 612, this is equivalent to finding the solution to

$$x^2 = 612$$

The function to use in Newton's method is then,

$$f(x) = x^2 - 612$$

with derivative,

$$f'(x) = 2x.$$

With an initial guess of 10, the sequence given by Newton's method is

$$\begin{aligned}
x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} &&= 10 - \frac{10^2 - 612}{2 \cdot 10} &&= 35.6 \\
x_2 &= x_1 - \frac{f(x_1)}{f'(x_1)} &&= 35.6 - \frac{35.6^2 - 612}{2 \cdot 35.6} &&= \underline{2}6.395505617978\ldots \\
x_3 &= \vdots &&= \vdots &&= \underline{24.}790635492455\ldots \\
x_4 &= \vdots &&= \vdots &&= \underline{24.738}688294075\ldots \\
x_5 &= \vdots &&= \vdots &&= \underline{24.738633753}767\ldots
\end{aligned}$$

Where the correct digits are underlined. With only a few iterations one can obtain a solution accurate to many decimal places.

Solution of $\cos(x) = x^3$

Consider the problem of finding the positive number x with $\cos(x) = x^3$. We can rephrase that as finding the zero of $f(x) = \cos(x) - x^3$. We have $f'(x) = -\sin(x) - 3x^2$. Since $\cos(x) \le 1$ for all x and $x^3 > 1$ for $x > 1$, we know that our zero lies between 0 and 1. We try a starting value of $x_0 = 0.5$. (Note that a starting value of 0 will lead to an undefined result, showing the importance of using a starting point that is close to the zero.)

$$\begin{aligned}
x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} &&= 0.5 - \frac{\cos(0.5) - (0.5)^3}{-\sin(0.5) - 3(0.5)^2} &&= 1.112141637097 \\
x_2 &= x_1 - \frac{f(x_1)}{f'(x_1)} &&= \vdots &&= \underline{0.}909672693736 \\
x_3 &= \vdots &&= \vdots &&= \underline{0.86}7263818209 \\
x_4 &= \vdots &&= \vdots &&= \underline{0.865477135298} \\
x_5 &= \vdots &&= \vdots &&= \underline{0.865474033111} \\
x_6 &= \vdots &&= \vdots &&= \underline{0.865474033102}
\end{aligned}$$

The correct digits are underlined in the above example. In particular, x_6 is correct to the number of decimal places given. We see that the number of correct digits after the decimal point increases from 2 (for x_3) to 5 and 10, illustrating the quadratic convergence.

References

[1] "Accelerated and Modified Newton Methods" (http://math.fullerton.edu/mathews/n2003/newtonacceleratemod.html). .

[2] Dence, Thomas, "Cubics, chaos and Newton's method", *Mathematical Gazette* 81, November 1997, 403-408.

[3] Strang, Gilbert, "A chaotic search for *i*", *"The College Mathematics Journal* 22, January 1991, pp. 3-12 (esp. p. 6).

- Tjalling J. Ypma, Historical development of the Newton-Raphson method, *SIAM Review* **37** (4), 531–551, 1995. doi:10.1137/1037125.
- Bonnans, J. Frédéric; Gilbert, J. Charles; Lemaréchal, Claude; Sagastizábal, Claudia A. (2006). *Numerical optimization: Theoretical and practical aspects* (http://www.springer.com/mathematics/applications/book/978-3-540-35445-1). Universitext (Second revised ed. of translation of 1997 French ed.). Berlin: Springer-Verlag. pp. xiv+490. doi:10.1007/978-3-540-35447-5. ISBN 3-540-35445-X. MR2265882.
- P. Deuflhard, *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms.* Springer Series in Computational Mathematics, Vol. 35. Springer, Berlin, 2004. ISBN 3-540-21099-7.
- C. T. Kelley, *Solving Nonlinear Equations with Newton's Method*, no 1 in Fundamentals of Algorithms, SIAM, 2003. ISBN 0-89871-546-6.
- J. M. Ortega, W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables.* Classics in Applied Mathematics, SIAM, 2000. ISBN 0-89871-461-3.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Chapter 9. Root Finding and Nonlinear Sets of Equations Importance Sampling" (http://apps.nrbook.com/empanel/index.html#pg=442). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.. See especially Sections 9.4 (http://apps.nrbook.com/empanel/index.html#pg=456), 9.6 (http://apps.nrbook.com/empanel/index.html#pg=473), and 9.7 (http://apps.nrbook.com/empanel/index.html#pg=477).
- Endre Süli and David Mayers, *An Introduction to Numerical Analysis*, Cambridge University Press, 2003. ISBN 0-521-00794-1.
- Kaw, Autar; Kalu, Egwu (2008). *Numerical Methods with Applications* (1st ed.).

External links

- Weisstein, Eric W., " Newton's Method (http://mathworld.wolfram.com/NewtonsMethod.html)" from MathWorld.
- Newton's method (http://www.lightandmatter.com/html_books/calc/ch04/ch04.html#Section4.1) -- section from an online textbook
- Newton-Raphson online calculator (http://www.maccery.com/maths/newton_raphson.php)
- Animations for Newton's method (http://math.fullerton.edu/mathews/a2001/Animations/RootFinding/NewtonMethod/NewtonMethod.html) by Prof. John H. Mathews
- Animations for Newton's method (http://animation.yihui.name/compstat:newton_s_method) by Yihui Xie using the R package animation (http://cran.r-project.org/package=animation)
- Newton-Raphson Method Notes, PPT, Mathcad, Maple, Matlab, Mathematica (http://numericalmethods.eng.usf.edu/topics/newton_raphson.html) at Holistic Numerical Methods Institute (http://numericalmethods.eng.usf.edu)
- Module for Newton’s Method by John H. Mathews (http://math.fullerton.edu/mathews/n2003/Newton'sMethodMod.html)
- Worked example (http://plus.maths.org/issue9/puzzle/solution.html)
- The Newton-Raphson algorithm (http://acumensoftwareinc.com/TechNotes/NewtonRaphson/html/) coded in C++ as a template class which takes a function object
- Newton's Method for finding roots - Source provides for C++ function and examples (http://www.amcgowan.ca/blog/computer-science/newtons-method-for-finding-roots/)
- Java code by Behzad Torkian (http://www.torkian.info/Site/Research/Entries/2008/2/28_Root-finding_algorithm_Java_Code_(_Secant,_Bisection,_Newton_).html)

- Matlab implementation of Newton's method (http://www.mathworks.com/matlabcentral/fileexchange/29370-newton-method-in-n-dimensions)
- Online root finding of a polynomial-Newton's method (http://catc.ac.ir/mazlumi/jscodes/newton.php) by Farhad Mazlumi

Householder's method

In numerical analysis, the class of **Householder's methods** are root-finding algorithms used for functions of one real variable with continuous derivatives up to some order *d*+1, where *d* will be the order of the Householder's method.

The algorithm is iterative and it has rate of convergence of *d*+1.

Method

Like any root-finding method, the Householder's method is a numerical algorithm for solving the nonlinear equation *f*(*x*) = 0. In this case, the function *f* has to be a function of one real variable. The method consists of a sequence of iterations [1]:

$$x_{n+1} = x_n + d\,\frac{(1/f)^{(d-1)}(x_n)}{(1/f)^{(d)}(x_n)}$$

beginning with an initial guess x_0.

If *f* is a *(d+1)* times continuously differentiable function and *a* is a zero of *f* but not of its derivative, then, in a neighborhood of *a*, the iterates x_n satisfy:

$|x_{n+1} - a| \leq K \cdot |x_n - a|^{d+1}$, for some $K > 0$.

This means that the iterates converge to the zero if the initial guess is sufficiently close, and that the convergence has rate *(d+1)*.

Motivation

An approximate idea of the origin of the Householder's method derives from the geometric series. Let the real-valued, continuously differentiable function *f(x)* have a simple zero at *x=a*, that is a root *f(a)=0* of multiplicity one, which is equivalent to $f'(a) \neq 0$. Then *1/f(x)* has a singularity that simple pole (also of multiplicity one) at *a*, and close to *a* the behavior of *1/f(x)* is dominated by the factor *1/(x-a)*. Approximatively one gets

$$\frac{1}{f(x)} = \frac{1}{f(x) - f(a)} = \frac{x-a}{f(x)-f(a)} \cdot \frac{-1}{a(1-x/a)} \approx \frac{-1}{af'(a)} \cdot \sum_{k=0}^{\infty} \left(\frac{x}{a}\right)^k .$$

Here $f'(a) \neq 0$because *a* is a simple zero of *f(x)*. The coefficient of degree *d* has the value $C\,a^{-d}$. Thus, one can now reconstruct the zero *a* by dividing the coefficient of degree *d-1* by the coefficient of degree *d*. Since this geometric series is an approximation to the Taylor expansion of *1/f(x)*, one can get estimates of the zero of *f(x)* – now without prior knowledge of the location of this zero – by dividing the corresponding coefficients of the Taylor expansion of *1/f(x)* or, more generally, *1/f(b+x)*. From that one gets, for any integer *d*, and if the corresponding derivatives exist,

$$a \approx b + \frac{(1/f)^{(d-1)}(b)}{(d-1)!}\;\frac{d!}{(1/f)^{(d)}(b)} = b + d\,\frac{(1/f)^{(d-1)}(b)}{(1/f)^{(d)}(b)}.$$

Alternative motivation

Suppose *x=a* is a simple root. Then near *x=a*, *(1/f)(x)* is a meromorphic function. Suppose we have the Taylor expansion:

$$(1/f)(x) = \sum_{d=0}^{\infty} \frac{(1/f)^{(d)}(b)}{d!}(x-b)^d.$$

By König's theorem, we have:

$$a - b = \lim_{d\to\infty} \frac{\frac{(1/f)^{(d-1)}(b)}{(d-1)!}}{\frac{(1/f)^{(d)}(b)}{d!}} = d\frac{(1/f)^{(d-1)}(b)}{(1/f)^{(d)}(b)}.$$

This suggests that the Householder's iteration might be a good convergent iteration. The actual proof of the convergence is also based on this idea.

The methods of lower order

The Householder's method of order *1* is just Newton's method, since:

$$\begin{aligned} x_{n+1} &= x_n + 1\,\frac{(1/f)(x_n)}{(1/f)^{(1)}(x_n)} \\ &= x_n + \frac{1}{f(x_n)} \cdot \left(\frac{-f'(x_n)}{f(x_n)^2}\right)^{-1} \\ &= x_n - \frac{f(x_n)}{f'(x_n)}. \end{aligned}$$

For the Householder's method of order *2* one gets Halley's method, since the identities

$$(1/f)'(x) = -\frac{f'(x)}{f(x)^2}$$

and

$$(1/f)''(x) = -\frac{f''(x)}{f(x)^2} + 2\frac{f'(x)^2}{f(x)^3}$$

result in

$$\begin{aligned} x_{n+1} &= x_n + 2\,\frac{(1/f)'(x_n)}{(1/f)''(x_n)} \\ &= x_n + \frac{-2f(x_n)\,f'(x_n)}{-f(x_n)f''(x_n)+2f'(x_n)^2} \\ &= x_n - \frac{f(x_n)f'(x_n)}{f'(x_n)^2-\frac{1}{2}f(x_n)f''(x_n)} \\ &= x_n + h_n\,\frac{1}{1+\frac{1}{2}(f''/f')(x_n)\,h_n}. \end{aligned}$$

In the last line, $h_n = -\frac{f(x_n)}{f'(x_n)}$ is the update of the Newton iteration at the point x_n. This line was added to demonstrate where the difference to the simple Newton's method lies.
The third order method is obtained from the identity of the third order derivative of *1/f*

$$(1/f)'''(x) = -\frac{f'''(x)}{f(x)^2} + 6\frac{f'(x)\,f''(x)}{f(x)^3} - 6\frac{f'(x)^3}{f(x)^4}$$

and has the formula

$$\begin{aligned} x_{n+1} &= x_n + 3\,\frac{(1/f)''(x_n)}{(1/f)'''(x_n)} \\ &= x_n - \frac{6f(x_n)\,f'(x_n)^2-3f(x_n)^2\,f''(x_n)}{6f'(x)^3-6f(x_n)f'(x_n)\,f''(x)+f(x_n)^2\,f'''(x_n)} \\ &= x_n + h_n\frac{1+\frac{1}{2}(f''/f')(x_n)\,h_n}{1+(f''/f')(x_n)\,h_n+\frac{1}{6}(f'''/f')(x_n)\,h_n^2} \end{aligned}$$

and so on...

Example

The first problem solved by Newton with the Newton-Raphson-Simpson method was the polynomial equation $y^3 - 2y - 5 = 0$. He observed that there should be a solution close to 2. Replacing y=x+2 transforms the equation into

$$0 = f(x) = -1 + 10x + 6x^2 + x^3.$$

The Taylor series of the reciprocal function starts with

$$\begin{aligned} 1/f(x) = \quad & -1 - 10\,x - 106\,x^2 - 1121\,x^3 - 11856\,x^4 - 125392\,x^5 \\ & -1326177\,x^6 - 14025978\,x^7 - 148342234\,x^8 - 1568904385\,x^9 \\ & -16593123232\,x^{10} + O(x^{11}) \end{aligned}$$

The result of applying Householder's methods of various orders at *x=0* is also obtained by dividing neighboring coefficients of the latter power series. For the first orders one gets the following values after just one iteration step: For an example, in the case of the 3rd order, $x_1 = 0.0 + 106/1121 = 0.09455842997324$.

d	x_1
1	**0.1**000000000000000000000000000000
2	**0.094**339622641509433962264150943396
3	**0.09455**842997323818019625334522747 6
4	**0.094551**282051282051282051282051282
5	**0.0945514**8653821615414061503126196 3
6	**0.09455148**143875214243649226309911 9
7	**0.094551481543**74689593837948412581 3
8	**0.0945514815423**367562335619133253 71
9	**0.09455148154232**48370868693824193 75
10	**0.094551481542326**78478801765822985

As one can see, there are a little bit more than *d* correct decimal places for each order d.

Let's calculate the x_2, x_3, x_4 values for some lowest order,

$$f = -1 + 10x + 6x^2 + x^3$$
$$f' = 10 + 12x + 3x^2$$
$$f'' = 12 + 6x$$
$$f''' = 6$$

And using following relations,

1st order; $x_{i+1} = x_i - f(x_i)/f'(x_i)$

2nd order; $x_{i+1} = x_i + (-2ff')/(2f'^2 - ff'')$

3rd order; $x_{i+1} = x_i - \dfrac{6ff'^2 - 3f^2f''}{6f'^3 - 6ff'f'' + f^2f'''}$

x	1st (Newton)	2nd (Halley)	3rd order	4th order
x_1	0.1	0.09433962264151	0.09455842997324	0.09455128205128
x_2	0.09456812110419	0.09455148154016421472	0.094551481542326591482567	
x_3	0.09455148169819930297	0.094551481542326591482386 54	0.094551481542326591482386 54057931	
x_4	0.094551481542326591496064 85	0.094551481542326591482386 5405793	0.094551481542326591482386 54057931	
x_5	0.094551481542326591482386 54057931	0.094551481542326591482386 5405793		
x_6	0.094551481542326591482386 54057931			

Derivation

An exact derivation of the Householder's methods starts from the Padé approximation of order *(d+1)* of the function resp. its Taylor development, where the approximant with linear numerator is chosen. If this has been achieved, the update for the next approximation results from computing the unique zero of the numerator.

The Padé approximation has the form

$$f(x+h) = \frac{a_0 + h}{b_0 + b_1 h + \cdots + b_{d-1} h^{d-1}} + O(h^{d+1}).$$

The rational function has a zero at $h = -a_0$.

Just as the Taylor polynomial of degree *d* has *d+1* coefficients that depend on the function *f*, also the Padé approximation always has *d+1* coefficients dependent on *f* and its derivatives. More precisely, in any Pade approximant, the degrees of numerator and denominator polynomials have to add to the order of the approximant. Therefore, $b_d = 0$ has to hold.

One could determine the Padé approximant starting from the Taylor polynomial of *f* using Euclid's algorithm. However, starting from the Taylor polynomial of *1/f* is shorter and leads directly to the given formula. Since

$$(1/f)(x+h) = (1/f)(x) + (1/f)'(x)h + \cdots + (1/f)^{(d-1)}(x)\frac{h^{d-1}}{(d-1)!} + (1/f)^{(d)}(x)\frac{h^d}{d!} + O(h^{d+1})$$

has to be equal to the inverse of the desired rational function, we get after multiplying with $a_0 + h$ in the power h^d the equation

$$0 = b_d = a_0 (1/f)^{(d)}(x)\frac{1}{d!} + (1/f)^{(d-1)}(x)\frac{1}{(d-1)!}.$$

Now, solving the last equation for the zero $h = -a_0$ of the numerator results in

$$\begin{aligned} h = \quad -a_0 &= \frac{\frac{1}{(d-1)!}(1/f)^{(d-1)}(x)}{\frac{1}{d!}(1/f)^{(d)}(x)} \\ &= \quad d\,\frac{(1/f)^{(d-1)}(x)}{(1/f)^{(d)}(x)} \end{aligned}.$$

This implies the iteration formula

$$x_{n+1} = x_n + d\,\frac{(1/f)^{(d-1)}(x_n)}{(1/f)^{(d)}(x_n)}.$$

External links

- Pascal Sebah and Xavier Gourdon (2001). "Newton's method and high order iteration" [2].

References

[1] Householder, Alston Scott (1970). *The Numerical Treatment of a Single Nonlinear Equation*. McGraw-Hill. p. 169. LCCN 79-103908.
[2] http://numbers.computation.free.fr/Constants/Algorithms/newton.html

Halley's method

In numerical analysis, **Halley's method** is a root-finding algorithm used for functions of one real variable with a continuous second derivative, i.e., C^2 functions. It is named after its inventor Edmond Halley, who also discovered Halley's Comet.

The algorithm is second in the class of Householder's methods, right after Newton's method. Like the latter, it produces iteratively a sequence of approximations to the root; their rate of convergence to the root is cubic. Multidimensional versions of this method exist.

Method

Like any root-finding method, Halley's method is a numerical algorithm for solving the nonlinear equation $f(x) = 0$. In this case, the function f has to be a function of one real variable. The method consists of a sequence of iterations:

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)}$$

beginning with an initial guess x_0.

If f is a three times continuously differentiable function and a is a zero of f but not of its derivative, then, in a neighborhood of a, the iterates x_n satisfy:

$$|x_{n+1} - a| \leq K \cdot |x_n - a|^3, \text{ for some } K > 0.$$

This means that the iterates converge to the zero if the initial guess is sufficiently close, and that the convergence is cubic.

The following alternative formulation shows the similarity between Halley's method and Newton's method. The expression $f(x_n)/f'(x_n)$ is computed only once, and it is particularly useful when $f''(x_n)/f'(x_n)$ can be simplified.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \left[1 - \frac{f(x_n)}{f'(x_n)} \cdot \frac{f''(x_n)}{2f'(x_n)}\right]^{-1}$$

A further alternative is as below, in which case the technique is sometimes referred to as **Bailey's method**.[1]

$$x_{n+1} = x_n - f(x_n) / \left[f'(x_n) - \frac{f(x_n)f''(x_n)}{2f'(x_n)}\right]$$

Using any variation, when the second derivative is very close to zero, the iteration is almost the same as under Newton's method.

Derivation

Consider the function

$$g(x) = \frac{f(x)}{\sqrt{|f'(x)|}}.$$

Any root of f which is *not* a root of its derivative is a root of *g*; and any root of *g* is a root of f. Applying Newton's method to *g* gives

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$$

with

$$g'(x) = \frac{2[f'(x)]^2 - f(x)f''(x)}{2f'(x)\sqrt{|f'(x)|}},$$

and the result follows. Notice that if f'(*c*) = 0, then one cannot apply this at *c* because *g*(*c*) would be undefined.

Cubic convergence

Suppose *a* is a root of *f* but not of its derivative. And suppose that the third derivative of *f* exists and is continuous in a neighborhood of *a* and x_n is in that neighborhood. Then Taylor's theorem implies:

$$0 = f(a) = f(x_n) + f'(x_n)(a - x_n) + \frac{f''(x_n)}{2}(a - x_n)^2 + \frac{f'''(\xi)}{6}(a - x_n)^3$$

and also

$$0 = f(a) = f(x_n) + f'(x_n)(a - x_n) + \frac{f''(\eta)}{2}(a - x_n)^2,$$

where ξ and η are numbers lying between *a* and x_n. Multiply the first equation by $2f'(x_n)$ and subtract from it the second equation times $f''(x_n)(a - x_n)$ to give:

$$\begin{aligned}
0 &= 2f(x_n)f'(x_n) + 2[f'(x_n)]^2(a - x_n) \\
&\quad + f'(x_n)f''(x_n)(a - x_n)^2 + \frac{f'(x_n)f'''(\xi)}{3}(a - x_n)^3 \\
&\quad - f(x_n)f''(x_n)(a - x_n) - f'(x_n)f''(x_n)(a - x_n)^2 \\
&\quad - \frac{f''(x_n)f''(\eta)}{2}(a - x_n)^3.
\end{aligned}$$

Canceling $f'(x_n)f''(x_n)(a - x_n)^2$ and re-organizing terms yields:

$$\begin{aligned}
0 = 2f(x_n)f'(x_n) &+ \left(2[f'(x_n)]^2 - f(x_n)f''(x_n)\right)(a - x_n) \\
&+ \left(\frac{f'(x_n)f'''(\xi)}{3} - \frac{f''(x_n)f''(\eta)}{2}\right)(a - x_n)^3.
\end{aligned}$$

Put the second term on the left side and divide through by $2[f'(x_n)]^2 - f(x_n)f''(x_n)$ to get:

$$a - x_n = \frac{-2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)} - \frac{2f'(x_n)f'''(\xi) - 3f''(x_n)f''(\eta)}{6(2[f'(x_n)]^2 - f(x_n)f''(x_n))}(a - x_n)^3.$$

Thus:

$$a - x_{n+1} = -\frac{2f'(x_n)f'''(\xi) - 3f''(x_n)f''(\eta)}{12[f'(x_n)]^2 - 6f(x_n)f''(x_n)}(a - x_n)^3.$$

The limit of the coefficient on the right side as x_n approaches *a* is:

$$-\frac{2f'(a)f'''(a) - 3f''(a)f''(a)}{12[f'(a)]^2}.$$

If we take *K* to be a little larger than the absolute value of this, we can take absolute values of both sides of the formula and replace the absolute value of coefficient by its upper bound near *a* to get:

$$|a - x_{n+1}| \leq K|a - x_n|^3$$

which is what was to be proved.

To summarize, $\Delta x_{i+1} = \frac{3(f'')^2 - 2f'f'''}{12(f')^2}(\Delta x_i)^3 + O[\Delta x_i]^4$,

where $\Delta x_i \triangleq x_i - a$.

References

Notes

[1] See for example the Bond Exchange of South Africa's *Bond Pricing Formula Specifications* (http://www.jse.co.za/Libraries/BESA_Bond_pricing/bond_pricing_formula_-_final.sflb.ashx), where Bailey's method is suggested when solving for a bond's Yield to maturity.

Sources

- T.R. Scavo and J.B. Thoo, On the geometry of Halley's method. *American Mathematical Monthly*, **102**:5 (1995), pp. 417–426. (http://www.jstor.org/pss/2975033)
- This article began as a translation from the equivalent article in French Wikipedia (http://fr.wikipedia.org/w/index.php?title=Itération_de_Halley&oldid=11673690), retrieved 22 January 2007.

External links

- Weisstein, Eric W., " Halley's method (http://mathworld.wolfram.com/HalleysMethod.html)" from MathWorld.
- Halley's Method by John H. Mathews (http://math.fullerton.edu/mathews/n2003/Halley'sMethodMod.html)
- Halley's Method by P. J. Acklam (http://home.online.no/~pjacklam/notes/halley/halley.pdf)
- *Newton's method and high order iterations* (http://numbers.computation.free.fr/Constants/Algorithms/newton.html), Pascal Sebah and Xavier Gourdon, 2001 (the site has a link to a Postscript version for better formula display)

Secant method

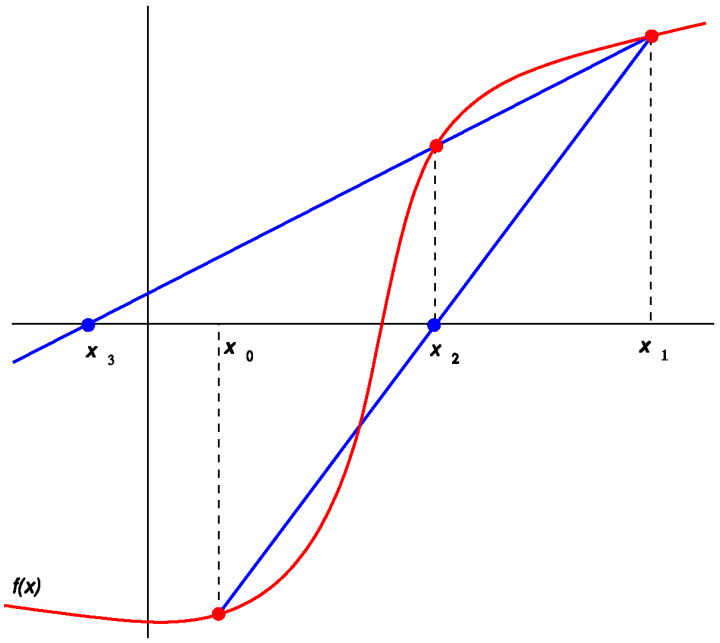
In numerical analysis, the **secant method** is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function *f*. The secant method can be thought of as a finite difference approximation of Newton's method. However, the method was developed independently of Newton's method, and predated the latter by over 3,000 years. [1]

The method

The secant method is defined by the recurrence relation

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

As can be seen from the recurrence relation, the secant method requires two initial values, x_0 and x_1, which should ideally be chosen to lie close to the root.



The first two iterations of the secant method. The red curve shows the function *f* and the blue lines are the secants.

Derivation of the method

Starting with initial values x_0 and x_1, we construct a line through the points $(x_0, f(x_0))$ and $(x_1, f(x_1))$, as demonstrated in the picture on the right. In point-slope form, this line has the equation

$$y = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1) + f(x_1)$$

We find the root of this line – the value of x such that $y = 0$ – by solving the following equation for x :

$$0 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1) + f(x_1)$$

The solution is

$$x = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

We then use this new value of x as x_2 and repeat the process using x_1 and x_2 instead of x_0 and x_1. We continue this process, solving for x_3, x_4, etc., until we reach a sufficiently high level of precision (a sufficiently small difference between x_n and x_{n-1}).

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

$$x_3 = x_2 - f(x_2)\frac{x_2 - x_1}{f(x_2) - f(x_1)}$$

...

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

Convergence

The iterates x_n of the secant method converge to a root of f, if the initial values x_0 and x_1 are sufficiently close to the root. The order of convergence is α, where

$$\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618$$

is the golden ratio. In particular, the convergence is superlinear, but not quite quadratic.

This result only holds under some technical conditions, namely that f be twice continuously differentiable and the root in question be simple (i.e., with multiplicity 1).

If the initial values are not close enough to the root, then there is no guarantee that the secant method converges. There is no general definition of "close enough", but the criterion has to do with how "wiggly" the function is on the interval $[\ x_0,\ x_1\ ]$. For example, if f is differentiable on that interval and there is a point where $f' = 0$ on the interval, then the algorithm may not converge.

Comparison with other root-finding methods

The secant method does not require that the root remain bracketed like the bisection method does, and hence it does not always converge. The false position method (or *regula falsi*) uses the same formula as the secant method. However, it does not apply the formula on x_{n-1} and x_n, like the secant method, but on x_n and on the last iterate x_k such that $f(x_k)$ and $f(x_n)$ have a different sign. This means that the false position method always converges.

The recurrence formula of the secant method can be derived from the formula for Newton's method

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

by using the finite difference approximation

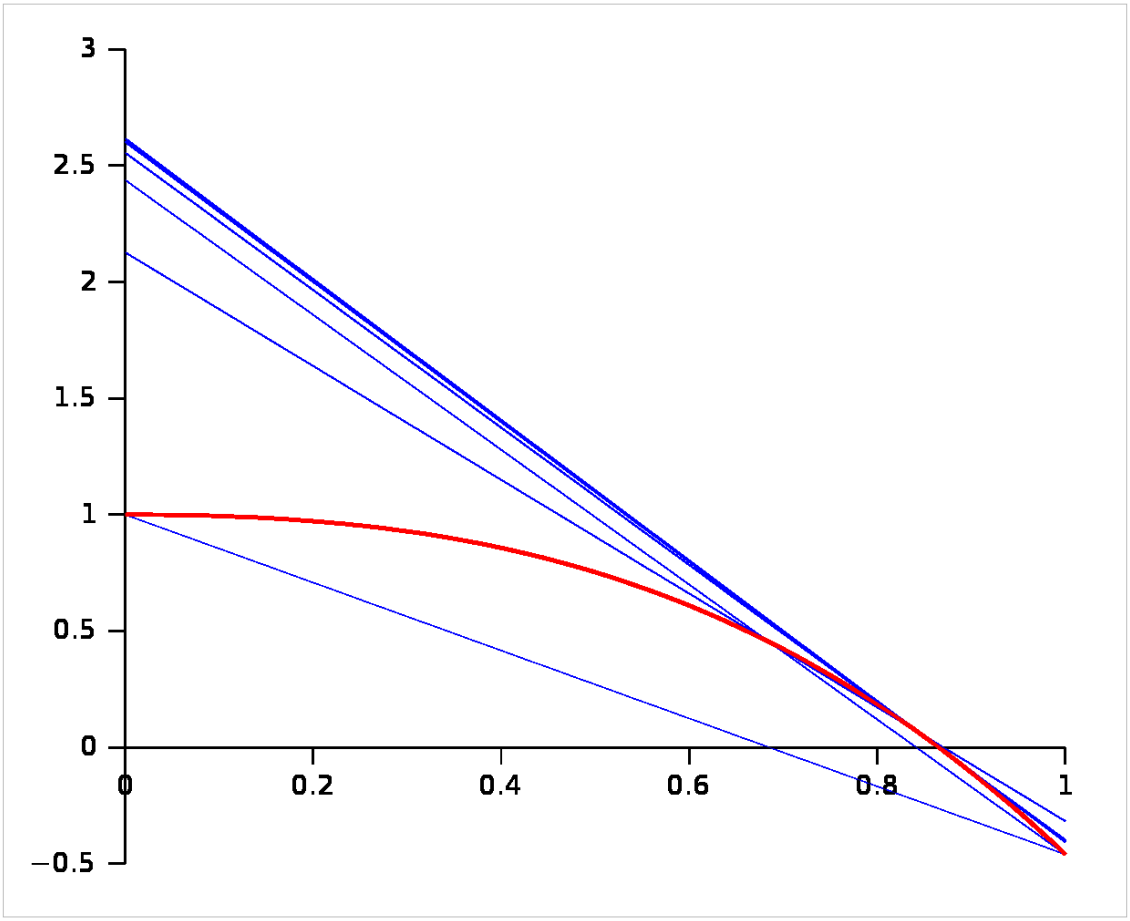
$$f'(x_{n-1}) \approx \frac{f(x_{n-1}) - f(x_{n-2})}{x_{n-1} - x_{n-2}}.$$

If we compare Newton's method with the secant method, we see that Newton's method converges faster (order 2 against α ≈ 1.6). However, Newton's method requires the evaluation of both f and its derivative f' at every step, while the secant method only requires the evaluation of f. Therefore, the secant method may occasionally be faster in practice. For instance, if we assume that evaluating f takes as much time as evaluating its derivative and we neglect all other costs, we can do two steps of the secant method (decreasing the logarithm of the error by a factor $\alpha^2 \approx 2.6$) for the same cost as one step of Newton's method (decreasing the logarithm of the error by a factor 2), so the secant method is faster. If however we consider parallel processing for the evaluation of the derivative, Newton's method proves its worth, being faster in time, though still spending more steps.

Generalizations

Broyden's method is a generalization of the secant method to more than one dimension.

The following graph shows the function f in red and the last secant line in bold blue. In the graph, the x-intercept of the secant line seems to be a good approximation of the root of f.



A computational example

The Secant method is applied to find a root of the function $f(x)=x^2-612$. Here is an implementation in the Matlab language.

```
# From calculation, we expect that the iteration converges at x=24.7386

f=@(x)x^2-612;
x(1)=10;
x(2)=30;
for i=3:7
    x(i)=x(i-1)-f(x(i-1))*(x(i-1)-x(i-2))/(f(x(i-1))-f(x(i-2)));
end
root=x(7)
```

Notes

[1] Papakonstantinou, J., *The Historical Development of the Secant Method in 1-D* (http://citation.allacademic.com/meta/p_mla_apa_research_citation/2/0/0/0/4/p200044_index.html), , retrieved 2011-06-29

References

- Kaw, Autar; Kalu, Egwu (2008), *Numerical Methods with Applications* (http://www.autarkaw.com/books/numericalmethods/index.html) (1st ed.).
- Allen, Myron B.; Isaacson, Eli L.. *Numerical analysis for applied science* (http://books.google.co.uk/books?id=PpB9cjOxQAQC). John Wiley & Sons. pp. 188–195. ISBN 978-0-471-55266-6.

External links

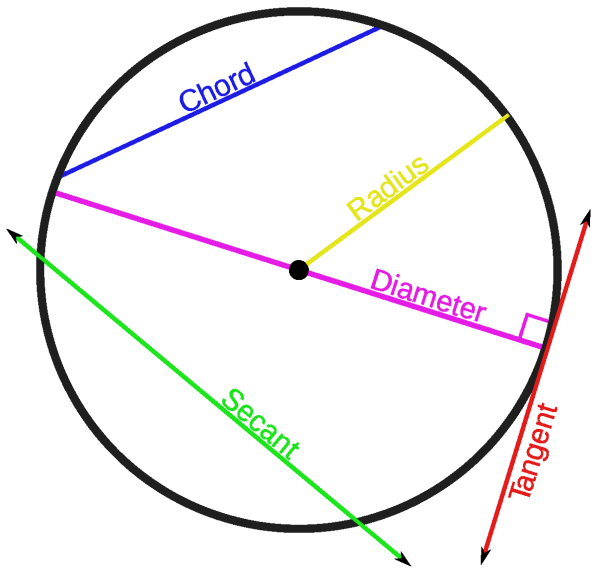
- Java code By Behzad Torkian (http://www.torkian.info/Site/Research/Entries/2008/2/28_Root-finding_algorithm_Java_Code_(_Secant,_Bisection,_Newton_).html)
- Animations for the secant method (http://math.fullerton.edu/mathews/a2001/Animations/RootFinding/SecantMethod/SecantMethod.html)
- Secant Method (http://numericalmethods.eng.usf.edu/topics/secant_method.html) Notes, PPT, Mathcad, Maple, Mathematica, Matlab at Holistic Numerical Methods Institute (http://numericalmethods.eng.usf.edu)
- Weisstein, Eric W., " Secant Method (http://mathworld.wolfram.com/SecantMethod.html)" from MathWorld.
- Module for Secant Method by John H. Mathews (http://math.fullerton.edu/mathews/n2003/SecantMethodMod.html)
- Online root finding of a polynomial-Secant method (http://catc.ac.ir/mazlumi/jscodes/secant.php) by Farhad Mazlumi

Secant line

A **secant line** of a curve is a line that (locally) intersects two points on the curve. The word *secant* comes from the Latin *secare*, *to cut*.

It can be used to approximate the tangent to a curve, at some point *P*. If the secant to a curve is defined by two points, *P* and *Q*, with *P* fixed and *Q* variable, as *Q* approaches *P* along the curve, the direction of the secant approaches that of the tangent at *P*, (assuming that the first-derivative of the curve is continuous at point *P* so that there is only one tangent). As a consequence, one could say that the limit as *Q* approaches *P* of the secant's slope, or direction, is that of the tangent. In calculus, this idea is the basis of the geometric definition of the derivative. A chord is the portion of a secant that lies within the curve.

A secant line on a map is a line where the projection is without distortion.



Common lines and line segments on a circle, including the secant line

External links

- Weisstein, Eric W., "Secant line [1]" from MathWorld.

References

[1] http://mathworld.wolfram.com/SecantLine.html

Broyden's method

In numerical analysis, **Broyden's method** is a quasi-Newton method for the root-finding algorithm in *k* variables. It was originally described by C. G. Broyden in 1965.[1]

Newton's method for solving the equation $f(x) = 0$ uses the Jacobian matrix and determinant, J, at every iteration. However, computing this Jacobian is a difficult and expensive operation. The idea behind Broyden's method is to compute the whole Jacobian only at the first iteration, and to do a rank-one update at the other iterations.

In 1979 Gay proved that when Broyden's method is applied to a linear system of size n x n, it terminates in *2n* steps.[2]

Description of the method

Solving single variable equation

In the secant method, we replace the first derivative $f'(x_n)$ with the finite difference approximation:

$$f'(x_n) \simeq \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}},$$

and proceeds similar to Newton's Method (n is the index for the iterations):

$$x_{n+1} = x_n - \frac{1}{f'(x_n)} f(x_n).$$

Solving a set of nonlinear equations

To solve a set of nonlinear equations

$$\vec{F}(\vec{x}) = \vec{0},$$

where the vector $\vec{F}$ is a function of vector $\vec{x}$ as (if we have k equations):

$$\vec{x} = (x_1, x_2, x_3, ..., x_k)$$

$$\vec{F}(\vec{x}) = (f_1(x_1, x_2, ..., x_k), f_2(x_1, x_2, ..., x_k)..., f_k(x_1, x_2, ..., x_k))$$

For such problems, Broyden gives a generalization of above formula, replacing the derivative $\vec{F}'$ with the Jacobian J. The Jacobian matrix is determined iteratively based on the **secant equation** with the finite difference approximation:

$$J_n \cdot (\vec{x}_n - \vec{x}_{n-1}) \simeq \vec{F}(\vec{x}_n) - \vec{F}(\vec{x}_{n-1}),$$

where n is the index of iterations. However above equation is under determined in more than one dimension. Broyden suggests using the current estimate of the Jacobian matrix J_{n-1} and improving upon it by taking the solution to the secant equation that is a minimal modification to J_{n-1} (minimal in the sense of minimizing the Frobenius norm $\|J_n - J_{n-1}\|_F$):

$$J_n = J_{n-1} + \frac{\Delta\vec{F}_n - J_{n-1}\Delta\vec{x}_n}{\|\Delta\vec{x}_n\|^2}\Delta\vec{x}^T$$

where

$$\Delta\vec{x} = \vec{x}_n - \vec{x}_{n-1}$$

$$\Delta\vec{F} = \vec{F}_n - \vec{F}_{n-1}$$

then we proceed in the Newton direction as:

$$\vec{x}_{n+1} = \vec{x}_n - J_n^{-1}\vec{F}(\vec{x}_n).$$

Broyden also suggested using the Sherman-Morrison formula to update directly the inverse of the Jacobian matrix:

$$J_n^{-1} = J_{n-1}^{-1} + \frac{\Delta\vec{x}_n - J_{n-1}^{-1}\Delta\vec{F}_n}{\Delta\vec{x}_n^T J_{n-1}^{-1}\Delta\vec{F}_n}(\Delta\vec{x}_n^T J_{n-1}^{-1})$$

This method is commonly known as the "good Broyden's method". A similar technique can be derived by using a slightly different modification to J_{n-1}(which minimizes $\left\|J_n^{-1} - J_{n-1}^{-1}\right\|_{\vec{F}}$instead); this yields the so-called "bad Broyden's method" (but see[3]):

$$J_n^{-1} = J_{n-1}^{-1} + \frac{\Delta\vec{x}_n - J_{n-1}^{-1}\Delta\vec{F}_n}{\Delta\vec{F}_n^T\Delta\vec{F}_n}\Delta\vec{F}_n^T$$

Many other quasi-Newton schemes have been suggested in optimization, where one seeks a maximum or minimum by finding the root of the first derivative (gradient in multi dimensions). The Jacobian of the gradient is called Hessian and is symmetric, adding further constraints to its upgrade.

References

[1] Broyden, C. G. (October 1965). "A Class of Methods for Solving Nonlinear Simultaneous Equations". *Mathematics of Computation* (American Mathematical Society) **19** (92): 577–593. doi:10.2307/2003941. JSTOR 2003941.

[2] Gay, D.M. (August 1979). "Some convergence properties of Broyden's method". *SIAM Journal of Numerical Analysis* (SIAM) **16** (4): 623–630. doi:10.1137/0716047.

[3] Kvaalen, Eric (November 1991). "A faster Broyden method". *BIT Numerical Mathematics* (SIAM) **31** (2): 369–372. doi:10.1007/BF01931297.

External links

- Module for Broyden's Method by John H. Mathews (http://math.fullerton.edu/mathews/n2003/BroydenMethodMod.html)

False position method

The **false position method** or **regula falsi method** is a term for problem-solving methods in arithmetic, algebra, and calculus. In simple terms, these methods begin by attempting to evaluate a problem using test ("false") values for the variables, and then adjust the values accordingly.

Two basic types of false position method can be distinguished, *simple false position* and *double false position*. *Simple false position* is aimed at solving problems involving direct proportion. Such problems can be written algebraically in the form: determine *x* such that

$$ax = b\,,$$

if *a* and *b* are known. *Double false position* is aimed at solving more difficult problems that can be written algebraically in the form: determine *x* such that

$$f(x) = b\,,$$

if it is known that

$$f(x_1) = b_1, f(x_2) = b_2.$$

Double false position is mathematically equivalent to linear interpolation; for an affine linear function,

$$f(x) = ax + c\,,$$

it provides the exact solution, while for a nonlinear function *f* it provides an approximation that can be successively improved by iteration.

Arithmetic and Algebra

In problems involving arithmetic or algebra, the **false position method** or **regula falsi** is used to refer to basic trial and error methods of solving problems by substituting test values for the unknown quantities. This is sometimes also referred to as "guess and check". Versions of this method predate the advent of algebra and the use of equations.

For simple false position, the method of solving what we would now write as $ax = b$ begins by using a test input value x', and finding the corresponding output value b' by multiplication: $ax' = b'$. The correct answer is then found by proportional adjustment, $x = x' \cdot b \div b'$. This technique is found in cuneiform tablets from ancient Babylonian mathematics, and possibly in papyri from ancient Egyptian mathematics [1].

Likewise, double false position arose in late antiquity as a purely arithmetical algorithm. It was used mostly to solve what are now called affine linear problems by using a pair of test inputs and the corresponding pair of outputs. This algorithm would be memorized and carried out by rote. The oldest surviving document demonstrating knowledge and proficiency in the technique is the Indian mathematical text *Vaishali Ganit* (c. 3rd century BC). In the ancient Chinese mathematical text called *The Nine Chapters on the Mathematical Art* (九章算術), dated from 200 BC to AD 100, most of Chapter 7 was devoted to the algorithm. There, the procedure was justified by concrete arithmetical arguments, then applied creatively to a wide variety of story problems, including one involving what we would call secant lines on a quadratic polynomial. A more typical example is this "joint purchase" problem:

> Now an item is purchased jointly; everyone contributes 8 [coins], the excess is 3; everyone contributes 7, the deficit is 4. Tell: The number of people, the item price, what is each? Answer: 7 people, item price 53. [2]

Between the 9th and 10th centuries, the Egyptian Muslim mathematician Abu Kamil wrote a now-lost treatise on the use of double false position, known as the *Book of the Two Errors* (*Kitāb al-khaṭāʾayn*). The oldest surviving writing on double false position from the Middle East is that of Qusta ibn Luqa (10th century), a Christian Arab mathematician from Baalbek, Lebanon. He justified the technique by a formal, Euclidean-style geometric proof. Within the tradition of medieval Muslim mathematics, double false position was known as *hisāb al-khaṭāʾayn* ("reckoning by two errors"). It was used for centuries, especially in the Maghreb, to solve practical problems such as

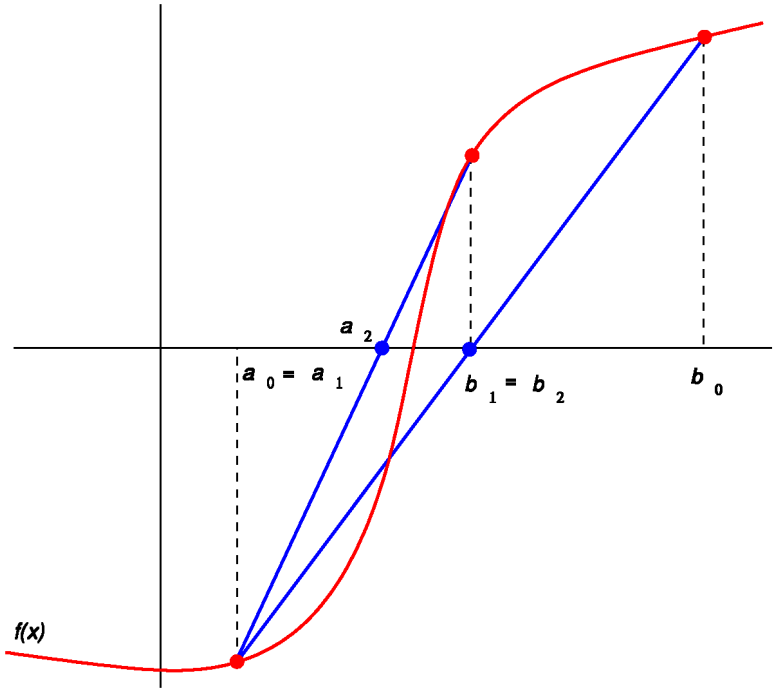
commercial and juridical questions (estate partitions according to rules of Quranic inheritance), as well as purely recreational problems. The algorithm was often memorized with the aid of mnemonics, such as a verse attributed to Ibn al-Yasamin and balance-scale diagrams explained by al-Hassar and Ibn al-Banna, all three being mathematicians of Moroccan origin.[3]

Leonardo of Pisa (Fibonacci) devoted Chapter 13 of his book *Liber Abaci* (AD 1202) to explaining and demonstrating the uses of double false position, terming the method *regulis elchatayn* after the *al-khaṭāʾayn* method that he had learned from Arab sources.[4]

Numerical analysis

In numerical analysis, double false position became a root-finding algorithm that combines features from the bisection method and the secant method.

Like the bisection method, the false position method starts with two points a_0 and b_0 such that $f(a_0)$ and $f(b_0)$ are of opposite signs, which implies by the intermediate value theorem that the function f has a root in the interval $[a_0, b_0]$, assuming continuity of the function f. The method proceeds by producing a sequence of shrinking intervals $[a_k, b_k]$ that all contain a root of f.



The first two iterations of the false position method. The red curve shows the function f and the blue lines are the secants.

At iteration number k, the number

$$c_k = \frac{f(b_k)a_k - f(a_k)b_k}{f(b_k) - f(a_k)}$$

is computed. As explained below, c_k is the root of the secant line through $(a_k, f(a_k))$ and $(b_k, f(b_k))$. If $f(a_k)$ and $f(c_k)$ have the same sign, then we set $a_{k+1} = c_k$ and $b_{k+1} = b_k$, otherwise we set $a_{k+1} = a_k$ and $b_{k+1} = c_k$. This process is repeated until the root is approximated sufficiently well.

The above formula is also used in the secant method, but the secant method always retains the last two computed points, while the false position method retains two points which certainly bracket a root. On the other hand, the only difference between the false position method and the bisection method is that the latter uses $c_k = (a_k + b_k) / 2$.

Finding the root of the secant

Given a_k and b_k, we construct the line through the points $(a_k, f(a_k))$ and $(b_k, f(b_k))$, as demonstrated in the picture on the right. Note that this line is a secant or chord of the graph of the function f. In point-slope form, it can be defined as

$$y - f(b_k) = \frac{f(b_k) - f(a_k)}{b_k - a_k}(x - b_k).$$

We now choose c_k to be the root of this line (substituting for x), and setting $y = 0$ and see that

$$f(b_k) + \frac{f(b_k) - f(a_k)}{b_k - a_k}(c_k - b_k) = 0.$$

Solving this equation gives the above equation for c_k.

Analysis

If the initial end-points a_0 and b_0 are chosen such that $f(a_0)$ and $f(b_0)$ are of opposite signs, then at each step, one of the end-points will get closer to a root of f. If the second derivative of f is of constant sign (so there is no inflection point) in the interval, then one endpoint (the one where f also has the same sign) will remain fixed for all subsequent iterations while the converging endpoint becomes updated. As a result, unlike the bisection method, the width of the bracket does not tend to zero (unless the zero is at an inflection point around which *sign(f)=-sign(f″)*). As a consequence, the linear approximation to $f(x)$, which is used to pick the false position, does not improve in its quality.

One example of this phenomenon is the function

$$f(x) = 2x^3 - 4x^2 + 3x$$

on the initial bracket [−1,1]. The left end, −1, is never replaced (after the first three iterations, f'' is negative on the interval) and thus the width of the bracket never falls below 1. Hence, the right endpoint approaches 0 at a linear rate (the number of accurate digits grows linearly, with a rate of convergence of 2/3).

For discontinuous functions, this method can only be expected to find a point where the function changes sign (for example at *x=0* for *1/x* or the sign function). In addition to sign changes, it is also possible for the method to converge to a point where the limit of the function is zero, even if the function is undefined (or has another value) at that point (for example at *x=0* for the function given by $f(x)=abs(x)-x^2$ when $x \neq 0$ and by *f(0)=5*, starting with the interval [-0.5, 3.0]). It is mathematically possible with discontinuous functions for the method to fail to converge to a zero limit or sign change, but this is not a problem in practice since it would require an infinite sequence of coincidences for both endpoints to get stuck converging to discontinuities where the sign does not change (for example at $x=\pm 1$ in $f(x)=1/(x-1)^2+1/(x+1)^2$). The method of bisection avoids this hypothetical convergence problem.

Illinois algorithm

While it is a misunderstanding to think that the method of false position is a good method, it is equally a mistake to think that it is unsalvageable. The failure mode is easy to detect (the same end-point is retained twice in a row) and easily remedied by next picking a modified false position, such as

$$c_k = \frac{\frac{1}{2}f(b_k)a_k - f(a_k)b_k}{\frac{1}{2}f(b_k) - f(a_k)}$$

or

$$c_k = \frac{f(b_k)a_k - \frac{1}{2}f(a_k)b_k}{f(b_k) - \frac{1}{2}f(a_k)}$$

down-weighting one of the endpoint values to force the next c_k to occur on that side of the function. The factor of 2 above looks like a hack, but it guarantees superlinear convergence (asymptotically, the algorithm will perform two regular steps after any modified step). There are other ways to pick the rescaling which give even better superlinear convergence rates.

The above adjustment to *regula falsi* is sometimes called the **Illinois algorithm**.[5][6] Ford (1995) summarizes and analyzes this and other similar superlinear variants of the method of false position. Judging from the bibliography, modified regula falsi methods were well known in the 1970s and have been subsequently forgotten or misremembered in current textbooks.

Example code

C code was written for clarity instead of efficiency. It was designed to solve the same problem as solved by the Newton's method and secant method code: to find the positive number x where $\cos(x) = x^3$. This problem is transformed into a root-finding problem of the form $f(x) = \cos(x) - x^3 = 0$.

```
#include <stdio.h>
#include <math.h>

double f(double x)
{
   return cos(x) - x*x*x;
}

double FalsiMethod(double s, double t, double e, int m)
{
   int n,side=0;
   double r,fr,fs = f(s),ft = f(t);

   for (n = 1; n <= m; n++)
   {
       r = (fs*t - ft*s) / (fs - ft);
       if (fabs(t-s) < e*fabs(t+s)) break;
       fr = f(r);

       if (fr * ft > 0)
       {
         t = r; ft = fr;
         if (side==-1) fs /= 2;
         side = -1;
       }
       else if (fs * fr > 0)
       {
         s = r;  fs = fr;
         if (side==+1) ft /= 2;
         side = +1;
       }
        else break;
    }
```

```
    return r;
}

int main(void)
{
    printf("%0.15f\n", FalsiMethod(0, 1, 5E-15, 100));
    return 0;
}
```

After running this code, the final answer is approximately 0.865474033101614

Notes

[1] Jean-Luc Chabert, ed., *A History of Algorithms: From the Pebble to the Microchip* (Berlin: Springer, 1999), pp. 86-91.

[2] Shen Kangshen, John N. Crossley and Anthony W.-C. Lun, 1999. *The Nine Chapters on the Mathematical Art: Companion and Commentary*. Oxford: Oxford University Press, p. 358.

[3] Schwartz, R. K (2004). "Issues in the Origin and Development of Hisab al-Khata'ayn (Calculation by Double False Position)". Eighth North African Meeting on the History of Arab Mathematics. Radès, Tunisia. Available online at: http://facstaff.uindy.edu/~oaks/Biblio/COMHISMA8paper.doc and http://www.ub.edu/islamsci/Schwartz.pdf

[4] Schwartz, R. K (2004). "Issues in the Origin and Development of Hisab al-Khata'ayn (Calculation by Double False Position)". Eighth North African Meeting on the History of Arab Mathematics. Radès, Tunisia. Available online at: http://facstaff.uindy.edu/~oaks/Biblio/COMHISMA8paper.doc and http://www.ub.edu/islamsci/Schwartz.pdf

[5] Dahlquist, Germund; Björck, Åke (1974). *Numerical Methods* (http://books.google.com/books?id=armfeHpJIwAC&pg=PA232). pp. 231–232. .

[6] Dowell, M.; Jarratt, P. (1971). "A modified regula falsi method for computing the root of an equation". *BIT* **11** (2): 168–174. doi:10.1007/BF01934364.

References

- J.A. Ford (1995). Improved Algorithms of Illinois-type for the Numerical Solution of Nonlinear Equations (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.8676). *Technical Report CSM-257*. University of Essex Press.
- Richard L. Burden, J. Douglas Faires (2000). *Numerical Analysis*, 7th ed. Brooks/Cole. ISBN 0-534-38216-9.
- L.E. Sigler (2002). *Fibonacci's Liber Abaci, Leonardo Pisano's Book of Calculation*. Springer-Verlag, New York. ISBN 0-387-40737-5.

External links

- The Regula Falsi Method by John H. Mathews (http://math.fullerton.edu/mathews/n2003/RegulaFalsiMod.html)

Ridders' method

In numerical analysis, **Ridders' method** is a root-finding algorithm based on the false position method and the use of an exponential function to successively approximate a root of a function *f*. The method is due to C. Ridders (1979).

Ridders' method is simpler than Brent's method but Press et al. (2007) claim that it usually performs about as well. The formula below converges quadratically when the function is well-behaved, which implies that the number of additional significant digits found at each step approximately doubles; but the function has to be evaluated twice for each step, so the overall order of convergence of the method is √2. If the function is not well-behaved, the root remains bracketed and the length of the bracketing interval at least halves on each iteration, so convergence is guaranteed.

Method

Press et al. (2007) describe the method as follows. Given two values of the independent variable, x_1 and x_2, which are on two different sides of the root being sought, the method begins by evaluating the function at the midpoint x_3 between the two points. One then finds the unique exponential function of the form e^{ax} which, when multiplied by *f*, transforms the function at the three points into a straight line. The false position method is then applied to the transformed values, leading to a new value x_4, between x_1 and x_2, which can be used as one of the two bracketing values in the next step of the iteration. The other bracketing value is taken to be x_3 if this has the opposite sign to x_4, or otherwise whichever or x_1 and x_2 has the opposite sign to x_4.

The method can be summarized by the formula (equation 9.2.4 from Press et al.)

$$x_4 = x_3 + (x_3 - x_1)\frac{\operatorname{sign}[f(x_1) - f(x_2)]f(x_3)}{\sqrt{f(x_3)^2 - f(x_1)f(x_2)}}.$$

References

- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 9.2.1. Ridders' Method" [1]. *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.
- Ridders, C. (1979). "A new algorithm for computing a single root of a real continuous function". *IEEE Transactions on Circuits and Systems* **26**: 979–980. doi:10.1109/TCS.1979.1084580.
- Kiusalaas, Jaan (2010). *Numerical Methods in Engineering with Python* [2] (2nd ed.). Cambridge University Press. pp. 146–150. ISBN 978-0-521-19132-6.

References

[1] http://apps.nrbook.com/empanel/index.html#pg=452

[2] http://books.google.co.uk/books?id=9SG1r8EJawIC&pg=PT156

Linear interpolation

Linear interpolation is a method of curve fitting using linear polynomials. **Lerp** is an abbreviation for *linear interpolation*, which can also be used as a verb (Raymond 2003).

Linear interpolation between two known points

If the two known points are given by the coordinates (x_0,y_0) and (x_1,y_1), the **linear interpolant** is the straight line between these points. For a value x in the interval (x_0,x_1), the value y along the straight line is given from the equation

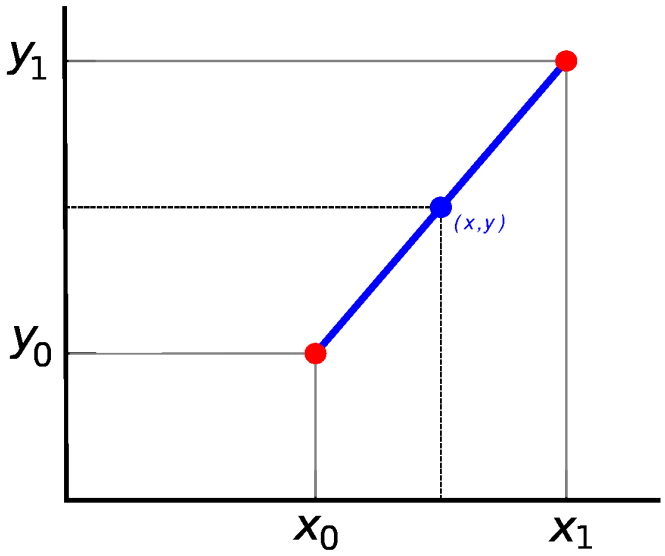
$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

which can be derived geometrically from the figure on the right. It is a special case of polynomial interpolation with $n = 1$.

Solving this equation for y, which is the unknown value at x, gives

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0} = y_0 + \frac{(x - x_0)y_1 - (x - x_0)y_0}{x_1 - x_0}$$

which is the formula for linear interpolation in the interval (x_0,x_1). Outside this interval, the formula is identical to linear extrapolation.

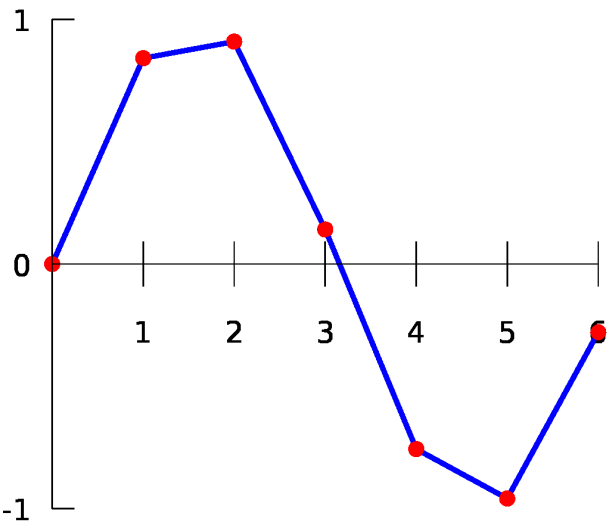


Given the two red points, the blue line is the linear interpolant between the points, and the value y at x may be found by linear interpolation.

This formula can also be understood as a weighted average. The weights are inversely related to the distance from the end points to the unknown point; the closer point has more influence than the farther point. Thus, the weights are $\frac{x - x_0}{x_1 - x_0}$ and $\frac{x_1 - x}{x_1 - x_0}$, which are normalized distances between the unknown point and each of the end points.

Interpolation of a data set

Linear interpolation on a set of data points (x_0, y_0), (x_1, y_1), ..., (x_n, y_n) is defined as the concatenation of linear interpolants between each pair of data points. This results in a continuous curve, with a discontinuous derivative (in general), thus of differentiability class C^0.



Linear interpolation on a data set (red points) consists of pieces of linear interpolants (blue lines).

Linear interpolation as approximation

Linear interpolation is often used to approximate a value of some function *f* using two known values of that function at other points. The *error* of this approximation is defined as

$$R_T = f(x) - p(x)$$

where *p* denotes the linear interpolation polynomial defined above

$$p(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0).$$

It can be proven using Rolle's theorem that if *f* has a continuous second derivative, the error is bounded by

$$|R_T| \leq \frac{(x_1 - x_0)^2}{8} \max_{x_0 \leq x \leq x_1} |f''(x)|.$$

As you see, the approximation between two points on a given function gets worse with the second derivative of the function that is approximated. This is intuitively correct as well: the "curvier" the function is, the worse the approximations made with simple linear interpolation.

Applications

Linear interpolation is often used to fill the gaps in a table. Suppose that one has a table listing the population of some country in 1970, 1980, 1990 and 2000, and that one wanted to estimate the population in 1994. Linear interpolation is an easy way to do this.

The basic operation of linear interpolation between two values is so commonly used in computer graphics that it is sometimes called a **lerp** in that field's jargon. The term can be used as a verb or noun for the operation. e.g. "Bresenham's algorithm lerps incrementally between the two endpoints of the line."

Lerp operations are built into the hardware of all modern computer graphics processors. They are often used as building blocks for more complex operations: for example, a bilinear interpolation can be accomplished in two lerps. Because this operation is cheap, it's also a good way to implement accurate lookup tables with quick lookup for smooth functions without having too many table entries.

Extensions

Accuracy

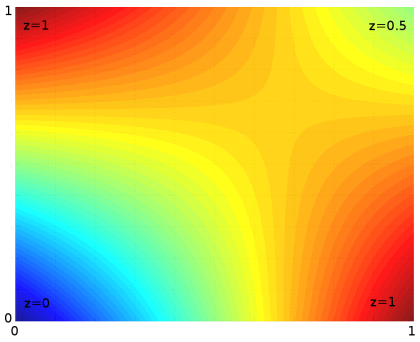
If a C^0 function is insufficient, for example if the process that has produced the data points is known be smoother than C^0, it is common to replace linear interpolation with spline interpolation, or even polynomial interpolation in some cases.

Multivariate

Linear interpolation as described here is for data points in one spatial dimension. For two spatial dimensions, the extension of linear interpolation is called bilinear interpolation, and in three dimensions, trilinear interpolation. Notice, though, that these interpolants are no longer linear functions of the spatial coordinates, rather products of linear functions; this is illustrated by the clearly non-linear example of bilinear interpolation in the figure below. Other extensions of linear interpolation can be applied to other kinds of mesh such as triangular and tetrahedral meshes, including Bézier surfaces. These may be defined as indeed higher dimensional piecewise linear function (see second figure below).

History

Linear interpolation has been used since antiquity for filling the gaps in tables, often with astronomical data. It is believed that it was used by Babylonian astronomers and mathematicians in Seleucid Mesopotamia (last three centuries BC), and by the Greek astronomer and mathematician, Hipparchus (2nd century BC). A description of linear interpolation can be found in the *Almagest* (2nd century AD) by Ptolemy.



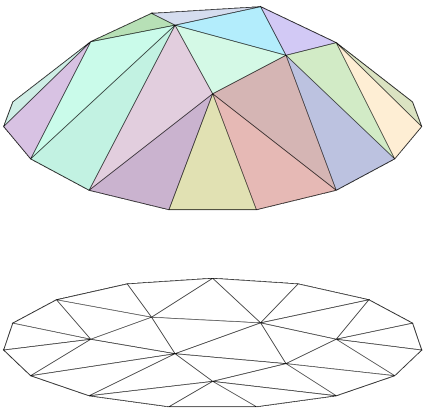
Example of bilinear interpolation on the unit square with the z-values 0, 1, 1 and 0.5 as indicated. Interpolated values in between represented by colour.

References

- Meijering, Erik (2002), "A chronology of interpolation: from ancient astronomy to modern signal and image processing", *Proceedings of the IEEE* **90** (3): 319–342, doi:10.1109/5.993400.
- Raymond, Eric (2003), "LERP" [1], *Jargon File (version 4.4.7)*.

External links

- Equations of the Straight Line [2] at cut-the-knot
- Implementing linear interpolation in Microsoft Excel [3]



A piecewise linear function in two dimensions (top) and the convex polytopes on which it is linear (bottom).

References

[1] http://www.catb.org/jargon/html/L/LERP.html
[2] http://www.cut-the-knot.org/Curriculum/Calculus/StraightLine.shtml
[3] http://www.blueleafsoftware.com/Products/Dagra/LinearInterpolationExcel.php

Polynomial interpolation

In numerical analysis, **polynomial interpolation** is the interpolation of a given data set by a polynomial: given some points, find a polynomial which goes exactly through these points.

Applications

Polynomials can be used to approximate more complicated curves, for example, the shapes of letters in typography, given a few points. A relevant application is the evaluation of the natural logarithm and trigonometric functions: pick a few known data points, create a lookup table, and interpolate between those data points. This results in significantly faster computations. Polynomial interpolation also forms the basis for algorithms in numerical quadrature and numerical ordinary differential equations.

Polynomial interpolation is also essential to perform sub-quadratic multiplication and squaring such as Karatsuba multiplication and Toom–Cook multiplication, where an interpolation through points on a polynomial which defines the product yields the product itself. For example, given $a = f(x) = a_0x^0 + a_1x^1 + ...$ and $b = g(x) = b_0x^0 + b_1x^1 + ...$ then the product ab is equivalent to $W(x) = f(x)g(x)$. Finding points along $W(x)$ by substituting x for small values in $f(x)$ and $g(x)$ yields points on the curve. Interpolation based on those points will yield the terms of $W(x)$ and subsequently the product ab. In the case of Karatsuba multiplication this technique is substantially faster than quadratic multiplication, even for modest-sized inputs. This is especially true when implemented in parallel hardware.

Definition

Given a set of $n + 1$ data points (x_i,y_i) where no two x_i are the same, one is looking for a polynomial p of degree at most n with the property

$$p(x_i) = y_i,\ i = 0, \ldots, n.$$

The unisolvence theorem states that such a polynomial p exists and is unique, and can be proved by the Vandermonde matrix, as described below.

The theorem states that for n+1 interpolation nodes (x_i), polynomial interpolation defines a linear bijection

$$L_n : \mathbb{K}^{n+1} \to \Pi_n$$

where Π_n is the vector space of polynomials (defined on any interval containing the nodes) of degree at most n.

Constructing the interpolation polynomial

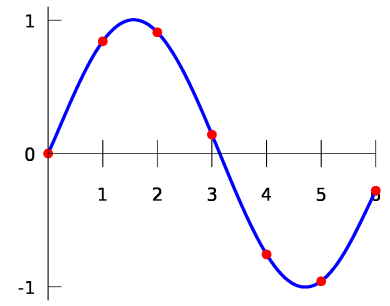
Suppose that the interpolation polynomial is in the form

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0. \qquad (1)$$

The statement that p interpolates the data points means that

$$p(x_i) = y_i \qquad \text{for all } i \in \{0, 1, \ldots, n\}.$$

If we substitute equation (1) in here, we get a system of linear equations in the coefficients a_k. The system in matrix-vector form reads



The red dots denote the data points (x_k, y_k), while the blue curve shows the interpolation polynomial.

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \dots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

We have to solve this system for a_k to construct the interpolant $p(x)$. The matrix on the left is commonly referred to as a Vandermonde matrix.

The condition number of the Vandermonde matrix may be large,[1] causing large errors when computing the coefficients a_i if the system of equations is solved using Gaussian elimination.

Several authors have therefore proposed algorithms which exploit the structure of the Vandermonde matrix to compute numerically stable solutions in $\mathcal{O}(n^2)$ operations instead of the $\mathcal{O}(n^3)$ required by Gaussian elimination.[2][3][4] These methods rely on constructing first a Newton interpolation of the polynomial and then converting it to the monomial form above.

Uniqueness of the interpolating polynomial

Proof 1

Suppose we interpolate through $n + 1$ data points with an at-most n degree polynomial $p(x)$ (we need at least $n + 1$ datapoints or else the polynomial cannot be fully solved for). Suppose also another polynomial exists also of degree at most n that also interpolates the $n + 1$ points; call it $q(x)$.

Consider $r(x) = p(x) - q(x)$. We know,

1. $r(x)$ is a polynomial
2. $r(x)$ has degree at most n, since $p(x)$ and $q(x)$ are no higher than this and we are just subtracting them.
3. At the $n + 1$ data points, $r(x_i) = p(x_i) - q(x_i) = y_i - y_i = 0$. Therefore $r(x)$ has $n + 1$ roots.

But $r(x)$ is an n degree polynomial (or less)! It has one root too many. Formally, if $r(x)$ is any non-zero polynomial, it must be writable as $r(x) = (x - x_0)(x - x_1) \cdots (x - x_n)$. By distributivity the $n + 1$ x's multiply together to make x^{n+1}, i.e. one degree higher than the maximum we set. So the only way $r(x)$ can exist is if $r(x) = 0$.

$$r(x) = 0 = p(x) - q(x) \implies p(x) = q(x)$$

So $q(x)$ (which could be any polynomial, so long as it interpolates the points) is identical with $p(x)$, and $p(x)$ is unique.

Proof 2

Given the Vandermonde matrix used above to construct the interpolant, we can set up the system

$$Va = y$$

We want to prove that V is nonsingular. Given

$$\det(V) = \prod_{i,j=0,i<j}^{n} (x_i - x_j)$$

since the $n + 1$ points are distinct, the determinant can't be zero as $x_i - x_j$ is never zero, therefore V is nonsingular and the system has a unique solution.

Either way this means that no matter what method we use to do our interpolation: direct, spline, lagrange etc., (assuming we can do all our calculations perfectly) we will always get the same polynomial.

Non-Vandermonde solutions

We are trying to construct our unique interpolation polynomial in the vector space $\prod_n$ of polynomials of degree n. When using a monomial basis for $\prod_n$ we have to solve the Vandermonde matrix to construct the coefficients a_k for the interpolation polynomial. This can be a very costly operation (as counted in clock cycles of a computer trying to do the job). By choosing another basis for $\prod_n$ we can simplify the calculation of the coefficients but then we have to do additional calculations when we want to express the interpolation polynomial in terms of a monomial basis.

One method is to write the interpolation polynomial in the Newton form and use the method of divided differences to construct the coefficients, e.g. Neville's algorithm. The cost is O (n^2) operations, while Gaussian elimination costs O (n^3) operations. Furthermore, you only need to do O (n) extra work if an extra point is added to the data set, while for the other methods, you have to redo the whole computation.

Another method is to use the Lagrange form of the interpolation polynomial. The resulting formula immediately shows that the interpolation polynomial exists under the conditions stated in the above theorem. Lagrange formula is to be preferred to Vandermorde formula when we are not interested in computing the coefficients of the polynomial, but in computing the value of $p(x)$ in a given x not in the original data set. In this case, we can reduce complexity to O (n^2).[5]

The Bernstein form was used in a constructive proof of the Weierstrass approximation theorem by Bernstein and has nowadays gained great importance in computer graphics in the form of Bézier curves.

Interpolation error

When interpolating a given function f by a polynomial of degree n at the nodes $x_0,...,x_n$ we get the error

$$f(x) - p_n(x) = f[x_0, \ldots, x_n, x] \prod_{i=0}^{n} (x - x_i)$$

where

$$f[x_0, \ldots, x_n, x]$$

is the notation for divided differences. When f is $n + 1$ times continuously differentiable on the smallest interval I which contains the nodes x_i and x then we can write the error in the Lagrange form as

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

for some ξ in I, according to the Mean value theorem for divided differences. Thus the remainder term in the Lagrange form of the Taylor theorem is a special case of interpolation error when all interpolation nodes x_i are identical.

In the case of equally spaced interpolation nodes $x_i = x_0 + ih$, it follows that the interpolation error is O $\left(h^{n+1}\right)$. However, this does not yield any information on what happens when $n \to \infty$. That question is treated in the section *Convergence properties*.

The above error bound suggests choosing the interpolation points x_i such that the product $|\Pi (x - x_i)|$ is as small as possible. The Chebyshev nodes achieve this.

Lebesgue constants

See the main article: Lebesgue constant.

We fix the interpolation nodes x_0, ..., x_n and an interval $[a, b]$ containing all the interpolation nodes. The process of interpolation maps the function f to a polynomial p. This defines a mapping X from the space $C([a, b])$ of all continuous functions on $[a, b]$ to itself. The map X is linear and it is a projection on the subspace Π_n of polynomials of degree n or less.

The Lebesgue constant L is defined as the operator norm of X. One has (a special case of Lebesgue's lemma):

$$\|f - X(f)\| \leq (L+1)\|f - p^*\|.$$

In other words, the interpolation polynomial is at most a factor $(L + 1)$ worse than the best possible approximation. This suggests that we look for a set of interpolation nodes that L small. In particular, we have for Chebyshev nodes:

$$L \leq \frac{2}{\pi}\log(n+1) + C \quad \text{for some constant } C.$$

We conclude again that Chebyshev nodes are a very good choice for polynomial interpolation, as the growth in n is exponential for equidistant nodes. However, those nodes are not optimal.

Convergence properties

It is natural to ask, for which classes of functions and for which interpolation nodes the sequence of interpolating polynomials converges to the interpolated function as the degree n goes to infinity? Convergence may be understood in different ways, e.g. pointwise, uniform or in some integral norm.

The situation is rather bad for equidistant nodes, in that uniform convergence is not even guaranteed for infinitely differentiable functions. One classical example, due to Carl Runge, is the function $f(x) = 1 / (1 + x^2)$ on the interval $[-5, 5]$. The interpolation error $\|f - p_n\|_\infty$ grows without bound as $n \to \infty$. Another example is the function $f(x) = |x|$ on the interval $[-1, 1]$, for which the interpolating polynomials do not even converge pointwise except at the three points $x = -1$, 0, and 1.[6]

One might think that better convergence properties may be obtained by choosing different interpolation nodes. The following **theorem** seems to be a rather encouraging answer:

> For any function $f(x)$ continuous on an interval $[a,b]$ there exists a table of nodes for which the sequence of interpolating polynomials $p_n(x)$ converges to $f(x)$ uniformly on $[a,b]$.

Proof. It's clear that the sequence of polynomials of best approximation $p_n^*(x)$ converges to $f(x)$ uniformly (due to Weierstrass approximation theorem). Now we have only to show that each $p_n^*(x)$ may be obtained by means of interpolation on certain nodes. But this is true due to a special property of polynomials of best approximation known from the Chebyshev alternation theorem. Specifically, we know that such polynomials should intersect $f(x)$ at least $n+1$ times. Choosing the points of intersection as interpolation nodes we obtain the interpolating polynomial coinciding with the best approximation polynomial.

The defect of this method, however, is that interpolation nodes should be calculated anew for each new function $f(x)$, but the algorithm is hard to be implemented numerically. Does there exist a single table of nodes for which the sequence of interpolating polynomials converge to any continuous function $f(x)$? The answer is unfortunately negative as it is stated by the following **theorem**:

> For any table of nodes there is a continuous function $f(x)$ on an interval $[a,b]$ for which the sequence of interpolating polynomials diverges on $[a,b]$.[7]

The proof essentially uses the lower bound estimation of the Lebesgue constant, which we defined above to be the operator norm of X_n (where X_n is the projection operator on Π_n). Now we seek a table of nodes for which

$$\lim_{n\to\infty} X_n f = f, \text{ for every } f \in C([a,b]).$$

Due to the Banach–Steinhaus theorem, this is only possible when norms of X_n are uniformly bounded, which cannot be true since we know that $\|X_n\| \geq \frac{2}{\pi}\log(n+1) + C.$

For example, if equidistant points are chosen as interpolation nodes, the function from Runge's phenomenon demonstrates divergence of such interpolation. Note that this function is not only continuous but even infinitely times differentiable on $[-1, 1]$. For better Chebyshev nodes, however, such an example is much harder to find because of the **theorem**:

> For every absolutely continuous function on $[-1, 1]$ the sequence of interpolating polynomials constructed on Chebyshev nodes converges to $f(x)$ uniformly.

Related concepts

Runge's phenomenon shows that for high values of n, the interpolation polynomial may oscillate wildly between the data points. This problem is commonly resolved by the use of spline interpolation. Here, the interpolant is not a polynomial but a spline: a chain of several polynomials of a lower degree.

Interpolation of periodic functions by harmonic functions is accomplished by Fourier transform. This can be seen as a form of polynomial interpolation with harmonic base functions, see trigonometric interpolation and trigonometric polynomial.

Hermite interpolation problems are those where not only the values of the polynomial p at the nodes are given, but also all derivatives up to a given order. This turns out to be equivalent to a system of simultaneous polynomial congruences, and may be solved by means of the Chinese remainder theorem for polynomials. Birkhoff interpolation is a further generalization where only derivatives of some orders are prescribed, not necessarily all orders from 0 to a k.

Collocation methods for the solution of differential and integral equations are based on polynomial interpolation.

The technique of rational function modeling is a generalization that considers ratios of polynomial functions.

At last, multivariate interpolation for higher dimensions.

Notes

[1] Gautschi, Walter (1975). "Norm Estimates for Inverses of Vandermonde Matrices". *Numerische Mathematik* **23** (4): 337–347. doi:10.1007/BF01438260.

[2] Higham, N. J. (1988). "Fast Solution of Vandermonde-Like Systems Involving Orthogonal Polynomials". *IMA Journal of Numerical Analysis* **8** (4): 473–486. doi:10.1093/imanum/8.4.473.

[3] Björck, Å; V. Pereyra (1970). "Solution of Vandermonde Systems of Equations". *Mathematics of Computation* (American Mathematical Society) **24** (112): 893–903. doi:10.2307/2004623. JSTOR 2004623.

[4] Calvetti, D and Reichel, L (1993). "Fast Inversion of Vanderomnde-Like Matrices Involving Orthogonal Polynomials". *BIT* **33** (33): 473–484. doi:10.1007/BF01990529.

[5] R.Bevilaqua, D. Bini, M.Capovani and O. Menchi (2003). *Appunti di Calcolo Numerico*. Chapter 5, p. 89. Servizio Editoriale Universitario Pisa - Azienda Regionale Diritto allo Studio Universitario.

[6] Watson (1980, p. 21) attributes the last example to Bernstein (1912).

[7] Watson (1980, p. 21) attributes this theorem to Faber (1914).

References

- Kendell A. Atkinson (1988). *An Introduction to Numerical Analysis* (2nd ed.), Chapter 3. John Wiley and Sons. ISBN 0-471-50023-2.
- Sergei N. Bernstein (1912), Sur l'ordre de la meilleure approximation des fonctions continues par les polynômes de degré donné. *Mem. Acad. Roy. Belg.* **4**, 1–104.
- L. Brutman (1997), Lebesgue functions for polynomial interpolation — a survey, *Ann. Numer. Math.* **4**, 111–127.
- Georg Faber (1912), Über die interpolatorische Darstellung stetiger Funktionen, *Deutsche Math. Jahr.* **23**, 192–210.
- M.J.D. Powell (1981). *Approximation Theory and Methods,* Chapter 4. Cambridge University Press. ISBN 0-521-29514-9.
- Michelle Schatzman (2002). *Numerical Analysis: A Mathematical Introduction,* Chapter 4. Clarendon Press, Oxford. ISBN 0-19-850279-6.
- Endre Süli and David Mayers (2003). *An Introduction to Numerical Analysis,* Chapter 6. Cambridge University Press. ISBN 0-521-00794-1.
- G. Alistair Watson (1980). *Approximation Theory and Numerical Methods*. John Wiley. ISBN 0-471-27706-1.

External links

- ALGLIB (http://www.alglib.net/interpolation/polynomial.php) has an implementations in C++ / C# / VBA / Pascal.
- GSL (http://www.gnu.org/software/gsl/) has a polynomial interpolation code in C
- Interpolating Polynomial (http://demonstrations.wolfram.com/InterpolatingPolynomial/) by Stephen Wolfram, the Wolfram Demonstrations Project.

Muller's method

Muller's method is a root-finding algorithm, a numerical method for solving equations of the form $f(x) = 0$. It is first presented by David E. Muller in 1956.

Muller's method is based on the secant method, which constructs at every iteration a line through two points on the graph of f. Instead, Muller's method uses three points, constructs the parabola through these three points, and takes the intersection of the x-axis with the parabola to be the next approximation.

Recurrence relation

The three initial values needed are denoted as x_k, x_{k-1} and x_{k-2}. The parabola going through the three points $(x_k, f(x_k))$, $(x_{k-1}, f(x_{k-1}))$ and $(x_{k-2}, f(x_{k-2}))$, when written in the Newton form, is

$$y = f(x_k) + (x - x_k)f[x_k, x_{k-1}] + (x - x_k)(x - x_{k-1})f[x_k, x_{k-1}, x_{k-2}],$$

where $f[x_k, x_{k-1}]$ and $f[x_k, x_{k-1}, x_{k-2}]$ denote divided differences. This can be rewritten as

$$y = f(x_k) + w(x - x_k) + f[x_k, x_{k-1}, x_{k-2}]\,(x - x_k)^2$$

where

$$w = f[x_k, x_{k-1}] + f[x_k, x_{k-2}] - f[x_{k-1}, x_{k-2}].$$

The next iterate is now given by the root of the quadratic equation $y = 0$. This yields the recurrence relation

$$x_{k+1} = x_k - \frac{2f(x_k)}{w \pm \sqrt{w^2 - 4f(x_k)f[x_k, x_{k-1}, x_{k-2}]}}.$$

In this formula, the sign should be chosen such that the denominator is as large as possible in magnitude. We do not use the standard formula for solving quadratic equations because that may lead to loss of significance.

Note that x_{k+1} can be complex, even if the previous iterates were all real. This is in contrast with other root-finding algorithms like the secant method or Newton's method, whose iterates will remain real if one starts with real numbers. Having complex iterates can be an advantage (if one is looking for complex roots) or a disadvantage (if it is known that all roots are real), depending on the problem.

Speed of convergence

The order of convergence of Muller's method is approximately 1.84. This can be compared with 1.62 for the secant method and 2 for Newton's method. So, the secant method makes less progress per iteration than Muller's method and Newton's method makes more progress.

More precisely, if ξ denotes a single root of f (so $f(\xi) = 0$ and $f'(\xi) \neq 0$), f is three times continuously differentiable, and the initial guesses x_0, x_1, and x_2 are taken sufficiently close to ξ, then the iterates satisfy

$$\lim_{k\to\infty} \frac{|x - x_k|}{|x - x_{k-1}|^p} = \left|\frac{f'''(\xi)}{6f'(\xi)}\right|^{(p-1)/2},$$

where $p \approx 1.84$ is the positive root of $x^3 - x^2 - x - 1 = 0$.

References

- Muller, David E., "A Method for Solving Algebraic Equations Using an Automatic Computer," *Mathematical Tables and Other Aids to Computation*, 10 (1956), 208-215. JSTOR 2001916
- Atkinson, Kendall E. (1989). *An Introduction to Numerical Analysis*, 2nd edition, Section 2.4. John Wiley & Sons, New York. ISBN 0-471-50023-2.
- Burden, R. L. and Faires, J. D. *Numerical Analysis*, 4th edition, pages 77ff.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 9.5.2. Muller's Method" [1]. *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

External links

- Module for Muller's Method by John H. Mathews [2]

References

[1] http://apps.nrbook.com/empanel/index.html#pg=466

[2] http://math.fullerton.edu/mathews/n2003/MullersMethodMod.html

Inverse quadratic interpolation

In numerical analysis, **inverse quadratic interpolation** is a root-finding algorithm, meaning that it is an algorithm for solving equations of the form $f(x) = 0$. The idea is to use quadratic interpolation to approximate the inverse of f. This algorithm is rarely used on its own, but it is important because it forms part of the popular Brent's method.

The method

The inverse quadratic interpolation algorithm is defined by the recurrence relation

$$x_{n+1} = \frac{f_{n-1}f_n}{(f_{n-2}-f_{n-1})(f_{n-2}-f_n)}x_{n-2} + \frac{f_{n-2}f_n}{(f_{n-1}-f_{n-2})(f_{n-1}-f_n)}x_{n-1} + \frac{f_{n-2}f_{n-1}}{(f_n-f_{n-2})(f_n-f_{n-1})}x_n,$$

where $f_k = f(x_k)$. As can be seen from the recurrence relation, this method requires three initial values, x_0, x_1 and x_2.

Explanation of the method

We use the three preceding iterates, x_{n-2}, x_{n-1} and x_n, with their function values, f_{n-2}, f_{n-1} and f_n. Applying the Lagrange interpolation formula to do quadratic interpolation on the inverse of f yields

$$f^{-1}(y) = \frac{(y-f_{n-1})(y-f_n)}{(f_{n-2}-f_{n-1})(f_{n-2}-f_n)}x_{n-2} + \frac{(y-f_{n-2})(y-f_n)}{(f_{n-1}-f_{n-2})(f_{n-1}-f_n)}x_{n-1} + \frac{(y-f_{n-2})(y-f_{n-1})}{(f_n-f_{n-2})(f_n-f_{n-1})}x_n.$$

We are looking for a root of f, so we substitute $y = f(x) = 0$ in the above equation and this results in the above recursion formula.

Behaviour

The asymptotic behaviour is very good: generally, the iterates x_n converge fast to the root once they get close. However, performance is often quite poor if you do not start very close to the actual root. For instance, if by any chance two of the function values f_{n-2}, f_{n-1} and f_n coincide, the algorithm fails completely. Thus, inverse quadratic interpolation is seldom used as a stand-alone algorithm.

The order of this convergence is approximately 1.8, it can be proved by the Secant Method analysis.

Comparison with other root-finding methods

As noted in the introduction, inverse quadratic interpolation is used in Brent's method.

Inverse quadratic interpolation is also closely related to some other root-finding methods. Using linear interpolation instead of quadratic interpolation gives the secant method. Interpolating f instead of the inverse of f gives Muller's method.

References

- James F. Epperson, An introduction to numerical methods and analysis [1], pages 182-185, Wiley-Interscience, 2007. ISBN 978-0-470-04963-1

References

[1] http://books.google.com/books?id=Mp8-z5mHptcC&lpg=PP1&pg=PA182#v=onepage&q&f=false

Brent's method

In numerical analysis, **Brent's method** is a complicated but popular root-finding algorithm combining the bisection method, the secant method and inverse quadratic interpolation. It has the reliability of bisection but it can be as quick as some of the less reliable methods. The idea is to use the secant method or inverse quadratic interpolation if possible, because they converge faster, but to fall back to the more robust bisection method if necessary. Brent's method is due to Richard Brent (1973) and builds on an earlier algorithm of Theodorus Dekker (1969).

Dekker's method

The idea to combine the bisection method with the secant method goes back to Dekker.

Suppose that we want to solve the equation $f(x) = 0$. As with the bisection method, we need to initialize Dekker's method with two points, say a_0 and b_0, such that $f(a_0)$ and $f(b_0)$ have opposite signs. If f is continuous on $[a_0, b_0]$, the intermediate value theorem guarantees the existence of a solution between a_0 and b_0.

Three points are involved in every iteration:

- b_k is the current iterate, i.e., the current guess for the root of f.
- a_k is the "contrapoint," i.e., a point such that $f(a_k)$ and $f(b_k)$ have opposite signs, so the interval $[a_k, b_k]$ contains the solution. Furthermore, $|f(b_k)|$ should be less than or equal to $|f(a_k)|$, so that b_k is a better guess for the unknown solution than a_k.
- b_{k-1} is the previous iterate (for the first iteration, we set $b_{k-1} = a_0$).

Two provisional values for the next iterate are computed. The first one is given by the secant method:

$$s = b_k - \frac{b_k - b_{k-1}}{f(b_k) - f(b_{k-1})} f(b_k),$$

and the second one is given by the bisection method

$$m = \frac{a_k + b_k}{2}.$$

If the result of the secant method, s, lies between b_k and m, then it becomes the next iterate ($b_{k+1} = s$), otherwise the midpoint is used ($b_{k+1} = m$).

Then, the value of the new contrapoint is chosen such that $f(a_{k+1})$ and $f(b_{k+1})$ have opposite signs. If $f(a_k)$ and $f(b_{k+1})$ have opposite signs, then the contrapoint remains the same: $a_{k+1} = a_k$. Otherwise, $f(b_{k+1})$ and $f(b_k)$ have opposite signs, so the new contrapoint becomes $a_{k+1} = b_k$.

Finally, if $|f(a_{k+1})| < |f(b_{k+1})|$, then a_{k+1} is probably a better guess for the solution than b_{k+1}, and hence the values of a_{k+1} and b_{k+1} are exchanged.

This ends the description of a single iteration of Dekker's method.

Brent's method

Dekker's method performs well if the function *f* is reasonably well-behaved. However, there are circumstances in which every iteration employs the secant method, but the iterates b_k converge very slowly (in particular, $|b_k - b_{k-1}|$ may be arbitrarily small). Dekker's method requires far more iterations than the bisection method in this case.

Brent proposed a small modification to avoid this problem. He inserted an additional test which must be satisfied before the result of the secant method is accepted as the next iterate. Two inequalities must be simultaneously satisfied:

- given a specific numerical tolerance δ, if the previous step used the bisection method, the inequality

  $$|\delta| < |b_k - b_{k-1}|$$

  must hold, otherwise the bisection method is performed and its result used for the next iteration. If the previous step performed interpolation, then the inequality

  $$|\delta| < |b_{k-1} - b_{k-2}|$$

  is used instead.
- Also, if the previous step used the bisection method, the inequality

  $$|s - b_k| < \tfrac{1}{2}|b_k - b_{k-1}|$$

  must hold, otherwise the bisection method is performed and its result used for the next iteration. If the previous step performed interpolation, then the inequality

  $$|s - b_k| < \tfrac{1}{2}|b_{k-1} - b_{k-2}|$$

  is used instead.

This modification ensures that at the k[th] iteration, a bisection step will be performed in at most $2log_2(|b_{k-1} - b_{k-2}|/\delta)$ additional iterations, because the above conditions force consecutive interpolation step sizes to halve every two iterations, and after at most $2log_2(|b_{k-1} - b_{k-2}|/\delta)$ iterations, the step size will be smaller than δ, which invokes a bisection step. Brent proved that his method requires at most N^2 iterations, where *N* denotes the number of iterations for the bisection method. If the function *f* is well-behaved, then Brent's method will usually proceed by either inverse quadratic or linear interpolation, in which case it will converge superlinearly. Furthermore, Brent's method uses inverse quadratic interpolation instead of linear interpolation (as used by the secant method) if $f(b_k)$, $f(a_k)$ and $f(b_{k-1})$ are distinct. This slightly increases the efficiency. As a consequence, the condition for accepting *s* (the value proposed by either linear interpolation or inverse quadratic interpolation) has to be changed: *s* has to lie between $(3a_k + b_k) / 4$ and b_k.

Algorithm

- **input** *a*, *b*, and a pointer to a subroutine for *f*
- calculate *f*(*a*)
- calculate *f*(*b*)
- **if** *f*(*a*) *f*(*b*) >= 0 **then** exit function because the root is not bracketed.
- **if** |*f*(*a*)| < |*f*(*b*)| **then swap** (*a*,*b*) **end if**
- *c* := *a*
- **set** mflag
- **repeat until** *f*(*b or s*) = 0 **or** |*b* − *a*| is small enough *(convergence)*
  - **if** *f*(*a*) ≠ *f*(*c*) **and** *f*(*b*) ≠ *f*(*c*) **then**
    - $s := \frac{af(b)f(c)}{(f(a)-f(b))(f(a)-f(c))} + \frac{bf(a)f(c)}{(f(b)-f(a))(f(b)-f(c))} + \frac{cf(a)f(b)}{(f(c)-f(a))(f(c)-f(b))}$ *(inverse quadratic interpolation)*

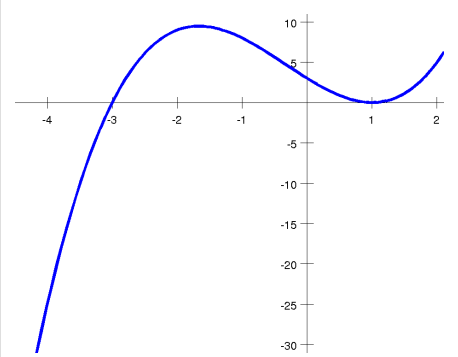
- **else**
  - $s := b - f(b)\dfrac{b-a}{f(b)-f(a)}$ *(secant rule)*
- **end if**
- **if** *(condition 1)* s is not between $(3a + b)/4$ and b
  or *(condition 2)* (mflag is set **and** $|s-b| \geq |b-c| / 2$)
  or *(condition 3)* (mflag is cleared **and** $|s-b| \geq |c-d| / 2$)
  or *(condition 4)* (mflag is set **and** $|b-c| < |\ \delta\ |$)
  or *(condition 5)* (mflag is cleared **and** $|c-d| < |\ \delta\ |$)
  then
  - $s := \dfrac{a+b}{2}$ *(bisection method)*
  - **set** mflag
- **else**
  - **clear** mflag
- **end if**
- calculate $f(s)$
- $d := c$ *(d is assigned for the first time here; it won't be used above on the first iteration because mflag is set)*
- $c := b$
- **if** $f(a)\, f(s) < 0$ **then** $b := s$ **else** $a := s$ **end if**
- **if** $|f(a)| < |f(b)|$ **then swap** (a,b) **end if**

end repeat

output *b or s (return the root)*

Example

Suppose that we are seeking a zero of the function defined by $f(x) = (x + 3)(x - 1)^2$. We take $[a_0, b_0] = [-4, 4/3]$ as our initial interval. We have $f(a_0) = -25$ and $f(b_0) = 0.48148$ (all numbers in this section are rounded), so the conditions $f(a_0)\, f(b_0) < 0$ and $|f(b_0)| \leq |f(a_0)|$ are satisfied.



Graph of $f(x) = (x + 3)(x - 1)^2$

1. In the first iteration, we use linear interpolation between $(b_{-1}, f(b_{-1})) = (a_0, f(a_0)) = (-4, -25)$ and $(b_0, f(b_0)) = (1.33333, 0.48148)$, which yields $s = 1.23256$. This lies between $(3a_0 + b_0) / 4$ and b_0, so this value is accepted. Furthermore, $f(1.23256) = 0.22891$, so we set $a_1 = a_0$ and $b_1 = s = 1.23256$.
2. In the second iteration, we use inverse quadratic interpolation between $(a_1, f(a_1)) = (-4, -25)$ and $(b_0, f(b_0)) = (1.33333, 0.48148)$ and $(b_1, f(b_1)) = (1.23256, 0.22891)$. This yields 1.14205, which lies between $(3a_1 + b_1) / 4$ and b_1. Furthermore, the inequality $|1.14205 - b_1| \leq |b_0 - b_{-1}| / 2$ is satisfied, so this value is accepted. Furthermore, $f(1.14205) = 0.083582$, so we set $a_2 = a_1$ and $b_2 = 1.14205$.
3. In the third iteration, we use inverse quadratic interpolation between $(a_2, f(a_2)) = (-4, -25)$ and $(b_1, f(b_1)) = (1.23256, 0.22891)$ and $(b_2, f(b_2)) = (1.14205, 0.083582)$. This yields 1.09032, which lies between $(3a_2 + b_2) / 4$ and b_2. But here Brent's additional condition kicks in: the inequality $|1.09032 - b_2| \leq |b_1 - b_0| / 2$ is not satisfied, so this value is rejected. Instead, the midpoint $m = -1.42897$ of the interval $[a_2, b_2]$ is computed. We have $f(m) = 9.26891$, so we set $a_3 = a_2$ and $b_3 = -1.42897$.

4. In the fourth iteration, we use inverse quadratic interpolation between $(a_3, f(a_3)) = (-4, -25)$ and $(b_2, f(b_2)) = (1.14205, 0.083582)$ and $(b_3, f(b_3)) = (-1.42897, 9.26891)$. This yields 1.15448, which is not in the interval between $(3a_3 + b_3) / 4$ and b_3). Hence, it is replaced by the midpoint $m = -2.71449$. We have $f(m) = 3.93934$, so we set $a_4 = a_3$ and $b_4 = -2.71449$.
5. In the fifth iteration, inverse quadratic interpolation yields −3.45500, which lies in the required interval. However, the previous iteration was a bisection step, so the inequality $|-3.45500 - b_4| \le |b_4 - b_3| / 2$ need to be satisfied. This inequality is false, so we use the midpoint $m = -3.35724$. We have $f(m) = -6.78239$, so m becomes the new contrapoint ($a_5 = -3.35724$) and the iterate remains the same ($b_5 = b_4$).
6. In the sixth iteration, we cannot use inverse quadratic interpolation because $b_5 = b_4$. Hence, we use linear interpolation between $(a_5, f(a_5)) = (-3.35724, -6.78239)$ and $(b_5, f(b_5)) = (-2.71449, 3.93934)$. The result is $s = -2.95064$, which satisfies all the conditions. But since the iterate did not change in the previous step, we reject this result and fall back to bisection. We update s = -3.03587, and $f(s)$ = -0.58418.
7. In the seventh iteration, we can again use inverse quadratic interpolation. The result is $s = -3.00219$, which satisfies all the conditions. Now, $f(s) = -0.03515$, so we set $a_7 = b_6$ and $b_7 = -3.00219$ (a_7 and b_7 are exchanged so that the condition $|f(b_7)| \le |f(a_7)|$ is satisfied). *(Correct : linear interpolation s = -2.99436, f(s) = 0.089961)*
8. In the eighth iteration, we cannot use inverse quadratic interpolation because $a_7 = b_6$. Linear interpolation yields $s = -2.99994$, which is accepted. *(Correct : s = -2.9999, f(s) = 0.0016)*
9. In the following iterations, the root $x = -3$ is approached rapidly: $b_9 = -3 + 6 \cdot 10^{-8}$ and $b_{10} = -3 - 3 \cdot 10^{-15}$. *(Correct : Iter 9 : f(s) = -1.4E-07, Iter 10 : f(s) = 6.96E-12)*

Example code

The above algorithm can be translated to c-like code as follows:

```
        public static double BrentsMethodSolve(Func<double, double> function, double
lowerLimit, double upperLimit, double errorTol)
        {
            double a = lowerLimit;
            double b = upperLimit;
            double c = 0;
            double d = double.MaxValue;

            double fa = function(a);
            double fb = function(b);

            double fc = 0;
            double s = 0;
            double fs = 0;

            // if f(a) f(b) >= 0 then error-exit
            if (fa * fb >= 0)
            {
                if (fa < fb)
                    return a;
                else
                    return b;
            }
```

```
            // if |f(a)| < |f(b)| then swap (a,b) end if
            if (Math.Abs(fa) < Math.Abs(fb))
            { double tmp = a; a = b; b = tmp; tmp = fa; fa = fb; fb =
tmp; }

            c = a;
            fc = fa;
            bool mflag = true;
            int i = 0;

            while (!(fb==0) && (Math.Abs(a-b) > errorTol))
            {
                if ((fa != fc) && (fb != fc))
                    // Inverse quadratic interpolation
                    s = a * fb * fc / (fa - fb) / (fa - fc) + b * fa *
fc / (fb - fa) / (fb - fc) + c * fa * fb / (fc - fa) / (fc - fb);
                else
                    // Secant Rule
                    s = b - fb * (b - a) / (fb - fa);

                double tmp2 = (3 * a + b) / 4;
                if ((!(((s > tmp2) && (s < b)) || ((s < tmp2) && (s > b)))) || (mflag && (Math.Abs(s
 - b) >= (Math.Abs(b - c) / 2))) || (!mflag && (Math.Abs(s - b) >=
(Math.Abs(c - d) / 2))))
                {
                    s = (a + b) / 2;
                    mflag = true;
                }
                else
                {
                    if ((mflag && (Math.Abs(b - c) < errorTol)) || (!mflag && (Math.Abs(c - d) < errorTol)))
                    {
                        s = (a + b) / 2;
                        mflag = true;
                    }
                    else
                        mflag = false;
                }
                fs = function(s);
                d = c;
                c = b;
                fc = fb;
                if (fa * fs < 0) { b = s; fb = fs; }
                else { a = s; fa = fs; }

                // if |f(a)| < |f(b)| then swap (a,b) end if
                if (Math.Abs(fa) < Math.Abs(fb))
```

```
                { double tmp = a; a = b; b = tmp; tmp = fa; fa = fb; fb
 = tmp; }
                i++;
                if (i > 1000)
                    throw new Exception(String.Format("Error is {0}",
fb));
            }
            return b;
        }
```

Implementations

- Brent (1973) published an Algol 60 implementation.
- Netlib contains a Fortran translation of this implementation with slight modifications.
- The PARI/GP method `solve` implements the method.
- Other implementations of the algorithm (in C++, C, and Fortran) can be found in the Numerical Recipes books.
- The Apache Commons Math library implements the algorithm in Java.
- The Modelica Standard Library implements the algorithm in Modelica.

References

- Atkinson (1989). *An Introduction to Numerical Analysis* (2nd ed.), Section 2.8. John Wiley and Sons. ISBN 0-471-50023-2.
- R.P. Brent (1973). *Algorithms for Minimization without Derivatives,* Chapter 4. Prentice-Hall, Englewood Cliffs, NJ. ISBN 0-13-022335-2.
- T.J. Dekker (1969). "Finding a zero by means of successive linear interpolation." In B. Dejon and P. Henrici (eds), *Constructive Aspects of the Fundamental Theorem of Algebra*, Wiley-Interscience, London. SBN 471-28300-9.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 9.3. Van Wijngaarden–Dekker–Brent Method" (http://apps.nrbook.com/empanel/index.html#pg=454). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

External links

- zeroin.f (http://www.netlib.org/go/zeroin.f) at Netlib.
- Module for Brent's Method (http://math.fullerton.edu/mathews/n2003/BrentMethodMod.html) by John H. Mathews

Horner's method

In mathematics, **Horner's method** (also known as **Horner scheme** in the UK or **Horner's rule** in the U.S.[1][2]) is either of two things: (i) an algorithm for calculating polynomials, which consists in transforming the monomial form into a computationally efficient form;[2] or (ii) a method for approximating the roots of a polynomial.[3] The latter is also known as **Ruffini–Horner's method**.[4]

These methods are named after the British mathematician William George Horner, although they were known before him by Ruffini[5]and even earlier, six hundred years ago by the Chinese mathematician Qin Jiushao[6].

Description of the algorithm

Given the polynomial

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

where $a_0, \ldots, a_n$ are real numbers, we wish to evaluate the polynomial at a specific value of x, say x_0.

To accomplish this, we define a new sequence of constants as follows:

$$\begin{aligned} b_n &:= a_n \\ b_{n-1} &:= a_{n-1} + b_n x_0 \\ &\vdots \\ b_0 &:= a_0 + b_1 x_0. \end{aligned}$$

Then b_0 is the value of $p(x_0)$.

To see why this works, note that the polynomial can be written in the form

$$p(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + a_n x)\cdots)).$$

Thus, by iteratively substituting the b_i into the expression,

$$\begin{aligned} p(x_0) &= a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-1} + b_n x_0)\cdots)) \\ &= a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(b_{n-1})\cdots)) \\ &\vdots \\ &= a_0 + x_0(b_1) \\ &= b_0. \end{aligned}$$

Examples

Evaluate

$$f(x) = 2x^3 - 6x^2 + 2x - 1 \text{ for } x = 3.$$

We use synthetic division as follows:

x_0	x^3	x^2	x^1	x^0
3	2	−6	2	−1
		6	0	6
	2	0	2	5

The entries in the third row are the sum of those in the first two. Each entry in the second row is the product of the x-value (3 in this example) with the third-row entry immediately to the left. The entries in the first row are the coefficients of the polynomial to be evaluated. Then the remainder of $f(x)$ on division by $x - 3$ is 5.

But by the remainder theorem, we know that the remainder is $f(3)$. Thus $f(3) = 5$

In this example, if $a_3 = 2, a_2 = -6, a_1 = 2, a_0 = -1$ we can see that $b_3 = 2, b_2 = 0, b_1 = 2, b_0 = 5$, the entries in the third row. So, synthetic division is based on Horner's method.

As a consequence of the polynomial remainder theorem, the entries in the third row are the coefficients of the second-degree polynomial, the quotient of $f(x)$ on division by $x - 3$. The remainder is 5. This makes Horner's method useful for polynomial long division.

Divide $x^3 - 6x^2 + 11x - 6$ by $x - 2$:

```
2 |   1    -6    11    -6
  |         2    -8     6
  └──────────────────────
      1    -4     3     0
```

The quotient is $x^2 - 4x + 3$.

Let $f_1(x) = 4x^4 - 6x^3 + 3x - 5$ and $f_2(x) = 2x - 1$. Divide $f_1(x)$ by $f_2(x)$ using Horner's method.

```
  2 |  4    -6     0     3   |   -5
----|-------------------------|------
  1 |         2    -2    -1   |    1
    |                         |
    └─────────────────────────|------
       2    -2    -1     1   |   -4
```

The third row is the sum of the first two rows, divided by 2. Each entry in the second row is the product of 1 with the third-row entry to the left. The answer is

$$\frac{f_1(x)}{f_2(x)} = 2x^3 - 2x^2 - x + 1 - \frac{4}{2x - 1}.$$

Floating point multiplication and division

Horner's method is a fast, code-efficient method for multiplication and division of binary numbers on a microcontroller with no hardware multiplier. One of the binary numbers to be multiplied is represented as a trivial polynomial, where, (using the above notation): $a_i = 1$, and x = 2. Then, x (or x to some power) is repeatedly factored out. In this binary numeral system (base 2), x = 2, so powers of 2 are repeatedly factored out.

Example

For example, to find the product of two numbers, (0.15625) and *m*:

$$\begin{aligned}(0.15625)m &= (0.00101_b)m = (2^{-3} + 2^{-5})m = (2^{-3})m + (2^{-5})m \\ &= 2^{-3}(m + (2^{-2})m) = 2^{-3}(m + 2^{-2}(m)).\end{aligned}$$

Method

To find the product of two binary numbers, d and m:

- 1. A register holding the intermediate result is initialized to d.
- 2. Begin with the least significant (rightmost) non-zero bit in m.
  - 2b. Count (to the left) the number of bit positions to the next most significant non-zero bit. If there are no more-significant bits, then take the value of the current bit position.
  - 2c. Using that value, perform a right-shift operation by that number of bits on the register holding the intermediate result
- 3. If all the non-zero bits were counted, then the intermediate result register now holds the final result. Otherwise, add d to the intermediate result, and continue in step #2 with the next most significant bit in m.

Derivation

In general, for a binary number with bit values: ($d_3d_2d_1d_0$) the product is:

$$(d_3 2^3 + d_2 2^2 + d_1 2^1 + d_0 2^0)m = d_3 2^3 m + d_2 2^2 m + d_1 2^1 m + d_0 2^0 m$$

At this stage in the algorithm, it is required that terms with zero-valued coefficients are dropped, so that only binary coefficients equal to one are counted, thus the problem of multiplication or division by zero is not an issue, despite this implication in the factored equation:

$$= d_0(m + 2\frac{d_1}{d_0}(m + 2\frac{d_2}{d_1}(m + 2\frac{d_3}{d_2}(m)))).$$

The denominators all equal one (or the term is absent), so this reduces to:

$$= d_0(m + 2d_1(m + 2d_2(m + 2d_3(m)))),$$

or equivalently (as consistent with the "method" described above):

$$= d_3(m + 2^{-1}d_2(m + 2^{-1}d_1(m + d_0(m)))).$$

In binary (base 2) math, multiplication by a power of 2 is merely a register shift operation. Thus, multiplying by 2 is calculated in base-2 by an arithmetic shift. The factor (2^{-1}) is a right arithmetic shift, a (0) results in no operation (since $2^0 = 1$, is the multiplicative identity element), and a (2^1) results in a left arithmetic shift. The multiplication product can now be quickly calculated using only arithmetic shift operations, addition and subtraction.

The method is particularly fast on processors supporting a single-instruction shift-and-addition-accumulate. Compared to a C floating-point library, Horner's method sacrifices some accuracy, however it is nominally 13 times faster (16 times faster when the "canonical signed digit" (CSD) form is used), and uses only 20% of the code space.[7]

Polynomial root finding

Using Horner's method in combination with Newton's method, it is possible to approximate the real roots of a polynomial. The algorithm works as follows. Given a polynomial $p_n(x)$ of degree n with zeros $z_n < z_{n-1} < \cdots < z_1$, make some initial guess x_0 such that $x_0 > z_1$. Now iterate the following two steps:

1. Using Newton's method, find the largest zero z_1 of $p_n(x)$ using the guess x_0.

2. Using Horner's method, divide out $(x - z_1)$ to obtain p_{n-1}. Return to step 1 but use the polynomial p_{n-1} and the initial guess z_1.

These two steps are repeated until all real zeros are found for the polynomial. If the approximated zeros are not precise enough, the obtained values can be used as initial guesses for Newton's method but using the full polynomial rather than the reduced polynomials.[8]

Example

Consider the polynomial,

$$p_6(x) = (x-3)(x+3)(x+5)(x+8)(x-2)(x-7)$$

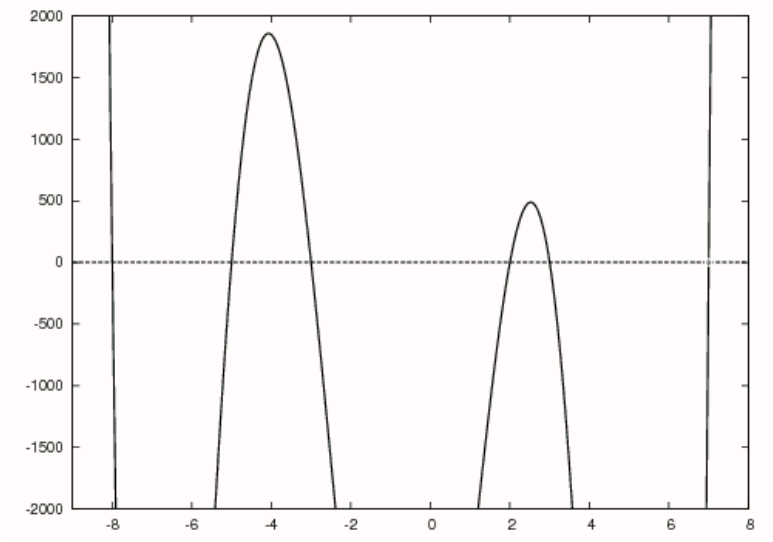
which can be expanded to

$$p_6(x) = x^6 + 4x^5 - 72x^4 - 214x^3 + 1127x^2 + 1602x - 5040.$$

From the above we know that the largest root of this polynomial is 7 so we are able to make an initial guess of 8. Using Newton's method the first zero of 7 is found as shown in black in the figure to the right. Next $p(x)$ is divided by $(x-7)$ to obtain

$$p_5(x) = x^5 + 11x^4 + 5x^3 - 179x^2 - 126x + 720$$

which is drawn in red in the figure to the right. Newton's method is used to find the largest zero of this polynomial with an initial guess of 7. The largest zero of this polynomial which corresponds to the second largest zero of the original polynomial is found at 3 and is circled in red. The degree 5 polynomial is now divided by $(x-3)$ to obtain



Polynomial root finding using Horner's method

$$p_4(x) = x^4 + 14x^3 + 47x^2 - 38x - 240$$

which is shown in yellow. The zero for this polynomial is found at 2 again using Newton's method and is circled in yellow. Horner's method is now used to obtain

$$p_3(x) = x^3 + 16x^2 + 79x + 120$$

which is shown in green and found to have a zero at −3. This polynomial is further reduced to

$$p_2(x) = x^2 + 13x + 40$$

which is shown in blue and yields a zero of −5. The final root of the original polynomial may be found by either using the final zero as an initial guess for Newton's method, or by reducing $p_2(x)$ and solving the linear equation. As can be seen, the expected roots of −8, −5, −3, 2, 3, and 7 were found.

Octave implementation

The following Octave code was used in the example above to implement Horner's method.

```
function [y b] = horner(a,x)
  % Input a is the polynomial coefficient vector, x the value to be
evaluated at.
  % The output y is the evaluated polynomial and b the divided
coefficient vector.
  b(1) = a(1);
  for i = 2:length(a)
    b(i) = a(i)+x*b(i-1);
  end
  y = b(length(a));
  b = b(1:length(b)-1);
end
```

Python implementation

The following Python code implements Horner's method.

```
def horner(x, *args):
    """A function that implements the Horner Scheme for evaluating a
    polynomial of coefficients *args in x."""
    result = 0
    for coefficient in args:
        result = result * x + coefficient
    return result
```

Application

Horner's method can be used to convert between different positional numeral systems – in which case x is the base of the number system, and the a_i coefficients are the digits of the base-x representation of a given number – and can also be used if x is a matrix, in which case the gain in computational efficiency is even greater. In fact, when x is a matrix, further acceleration is possible which exploits the structure of matrix multiplication, and only $\sqrt{n}$ instead of n multiplies are needed (at the expense of requiring more storage) using the 1973 method of Paterson and Stockmeyer.[9]

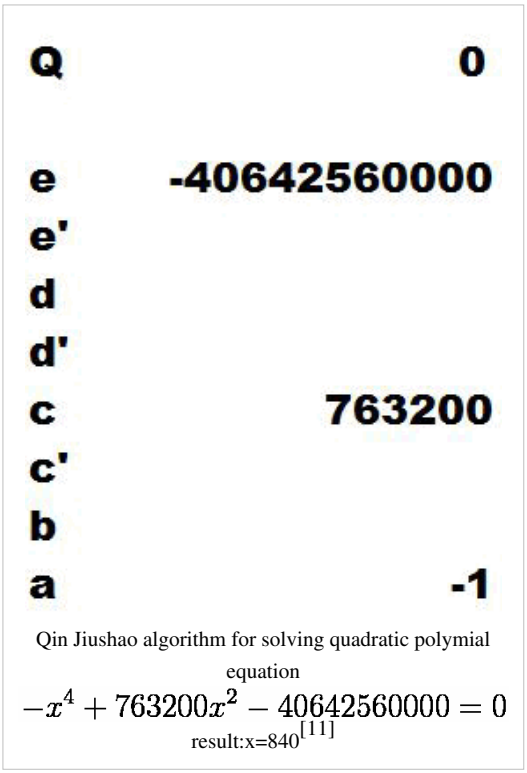
Efficiency

Evaluation using the monomial form of a degree-n polynomial requires at most n additions and $(n^2 + n)/2$ multiplications, if powers are calculated by repeated multiplication and each monomial is evaluated individually. (This can be reduced to n additions and $2n - 1$ multiplications by evaluating the powers of x iteratively.) If numerical data are represented in terms of digits (or bits), then the naive algorithm also entails storing approximately $2n$ times the number of bits of x (the evaluated polynomial has approximate magnitude x^n, and one must also store x^n itself). By contrast, Horner's method requires only n additions and n multiplications, and its storage requirements are only n times the number of bits of x. Alternatively, Horner's method can be computed with n fused multiply–adds. Horner's method can also be extended to evaluate the first k derivatives of the polynomial with kn additions and multiplications.[10]

Horner's method is optimal, in the sense that any algorithm to evaluate an arbitrary polynomial must use at least as many operations. Alexander Ostrowski proved in 1954 that the number of additions required is minimal. Victor Pan proved in 1966 that the number of multiplications is minimal. However, when x is a matrix, Horner's method is not optimal.

This assumes that the polynomial is evaluated in monomial form and no preconditioning of the representation is allowed, which makes sense if the polynomial is evaluated only once. However, if preconditioning is allowed and the polynomial is to be evaluated many times, then faster algorithms are possible. They involve a transformation of the representation of the polynomial. In general, a degree-n polynomial can be evaluated using only $\lfloor n/2 \rfloor + 2$ multiplications and n additions (see Knuth: *The Art of Computer Programming*, Vol.2).

History



Qin Jiushao algorithm for solving quadratic polymial equation

$-x^4 + 763200x^2 - 40642560000 = 0$

result:x=840[11]

Horner's paper in Part II of *Philosophical Transactions of the Royal Society of London* for 1819 was warmly and expansively welcomed by a reviewer in the issue of *The Monthly Review: or, Literary Journal* for April, 1820; in comparison, a technical paper by Charles Babbage is dismissed curtly in this review. However, the reviewer noted that another, similar method had also recently been promoted by the architect and mathematical expositor, Peter Nicholson. This in fact is the method now known as Horner's method which was only subsequently described by Horner in 1830.[12]This theme is developed in a further review of some of Nicholson's books in the issue of *The Monthly Review* for December, 1820, which in turn ends with notice of the appearance of a booklet by Theophilus Holdred, from whom Nicholson acknowledges he obtained the gist of his approach in the first place, although claiming to have improved upon it. The sequence of reviews is concluded in the issue of *The Monthly Review* for September, 1821, with the reviewer reasserting both Horner's priority and the primacy of his method, judiciously observing that had Holdred published forty years earlier, his contribution could more easily be recognized. The reviewer is exceptionally well-informed, even having sighted Horner's preparatory correspondence with Peter Barlow in 1818, seeking work of Budan. As it happened, Henry Atkinson invented a similar technique in 1809. J. R. Young, writing in the mid-1830s, concluded that Holdred's first method replicated Atkinson's while his improved method was only added to his text some months after the appearance of Holdred's booklet. Precocious though Augustus de Morgan was, he was not the reviewer for *The Monthly Review*, while several others wrote Horner firmly into their records.

The Chinese mathematician Qin Jiushao in his *Mathematical Treatise in Nine Sections* in the 1247, published method of solving high order polynomial equation, which was based on earlier works of the 11th century Song dynasty mathematician Jia Xian. Yoshio Mikami in his *Development of Mathematics in China and Japan published in Leipzing in 1912, gave a detailed description of Qin's method, he wrote:"who can deny the fact of Horner's illustrious process being used in China at least nearly six long centuries earlier than in Europe...the lapse of time sufficiently makes it not altogether impossible that the Europeans could have known of the Chinese method in a direct or indirect way."*[13].

Belgium mathematic historian Ulrich Librecht gave a detail description of Qin's method, he concluded:*It is obvious that this procedure is a Chinese invention....the method was not known in India". He said, Fibonacci probably leaned of it from Arabs, who perhaps borrowed from* the Chinese[14].

As for awareness of Ruffini, citations of Ruffini's work by authors in *Philosophical Transactions* speak volumes: there are none - Ruffini's name only appears in 1814, recording a work he donated to the Royal Society.

The feature of Horner's writing that most distinguishes it from his English contemporaries is the way he draws on the Continental literature, notably the work of Arbogast. The advocacy, as well as the detraction, of Horner's Method has this as an unspoken subtext. Quite how he gained that familiarity has not been determined. Horner is known to have made a close reading of John Bonneycastle's book on algebra. Bonneycastle recognizes that Arbogast has the general, combinatorial expression for the reversion of series, a project going back at least to Newton. But Bonneycastle's main purpose in mentioning Arbogast is not to praise him, but to observe that Arbogast's notation is incompatible with the approach he adopts.

According to Augustus De Morgan, Horner made his method known in a paper titled "A new method of solving numerical equations of all orders, by continuous approximation", which was read on July 1, 1819 by Davies Gilbert before the Royal Society of London. This paper was subsequently published in the same year in the "Philosophical Transactions".[15][16]

De Morgan's advocacy of Horner's priority in discovery[5][17] led to "Horner's method" being so called in textbooks. However, this method was known long before. In reverse chronological order, Horner's method was already known to:

- Paolo Ruffini in 1809 (see Ruffini's rule)[5][17]
- Isaac Newton in 1669
- the Chinese mathematician Zhu Shijie in the 14th century[17]
- the Chinese mathematician Qin Jiushao in his *Mathematical Treatise in Nine Sections* in the 13th century
- the Persian mathematician Sharaf al-Dīn al-Tūsī in the 12th century[18]
- the Chinese mathematician Jia Xian in the 11th century (Song Dynasty)
- *The Nine Chapters on the Mathematical Art*, a Chinese work of the Han Dynasty (202 BC – 220 AD) edited by Liu Hui (fl. 3rd century).[19]

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. pp. 41, 900, 990.

[2] "Wolfram MathWorld: Horner's Rule" (http://mathworld.wolfram.com/HornersRule.html). .

[3] "Wolfram MathWorld: Horner's Method" (http://mathworld.wolfram.com/HornersMethod.html). .

[4] "French Wikipedia: Méthode de Ruffini-Horner" (http://fr.wikipedia.org/wiki/MÃ©thode_de_Ruffini-Horner). .

[5] Florian Cajori, Horner's method of approximation anticipated by Ruffini (http://projecteuclid.org/DPubS/Repository/1.0/Disseminate?view=body&id=pdf_1&handle=euclid.bams/1183421253), Bulletin of the American Mathematical Society, Vol. 17, No. 9, pp. 409–414, 1911 (read before the Southwestern Section of the American Mathematical Society on November 26, 1910).

[6] *It is obvious that this procedure is a Chinese invention...." Ulrich Librecht, Chinese Mathematics in the Thirteenth Century, Chapter 13, Numerial Equations of Higher Degree, p178 Dover, ISBN 0-486-44619-0*

[7] Kripasagar, March 2008, "Efficient Micro Mathematics", Circuit Cellar, issue 212, p. 62.

[8] Kress, Rainer, "Numerical Analysis", Springer, 1991, p.112.

[9] Higham, Nicholas. (2002). *Accuracy and Stability of Numerical Algorithms*. Philadelphia: SIAM. ISBN 0-89871-521-0. Section 5.4.

[10] W. Pankiewicz. "Algorithm 337: calculation of a polynomial and its derivative values by Horner scheme" (http://portal.acm.org/citation.cfm?doid=364063.364089). .

[11] Urich Librecht Chinese Mathematics in the Thirteenth Century, p181–191, Dover ISBN 0-486-44619-0

[12] Fuller A. T. :Horner versus Holdred: An Episode in the History of Root Computation, Historia Mathematica 26 (1999), 29–51

[13] Yoshio Mikami, Chinese Mathematics in the Thirteenth Century, Chapter 11, Chin Chiu Shao, p77 Chelsea Publishing Co

[14] Ulrich Librecht, Chinese Mathematics in the Thirteenth Century, Chapter 13, Numerial Equations of Higher Degree, p208 Dover, ISBN 0-486-44619-0

[15] William George Horner (July 1819). "A new method of solving numerical equations of all orders, by continuous approximation". *Philosophical Transactions* (Royal Society of London): pp. 308–335.

[16] *JSTOR archive of the Royal Society of London*. JSTOR 107508.

[17] O'Connor, John J.; Robertson, Edmund F., "Horner's method" (http://www-history.mcs.st-andrews.ac.uk/Biographies/Horner.html), *MacTutor History of Mathematics archive*, University of St Andrews, .

[18] J. L. Berggren (1990). "Innovation and Tradition in Sharaf al-Din al-Tusi's Muadalat", *Journal of the American Oriental Society* **110** (2), p. 304–309.

[19] Temple, Robert. (1986). *The Genius of China: 3,000 Years of Science, Discovery, and Invention*. With a forward by Joseph Needham. New York: Simon and Schuster, Inc. ISBN 0-671-62028-2. Page 142.

Bibliography

- Horner, William George (July 1819). "A new method of solving numerical equations of all orders, by continuous approximation". *Philosophical Transactions* (Royal Society of London): pp. 308–335.
- Spiegel, Murray R. (1956). *Schaum's Outline of Theory and Problems of College Algebra*. McGraw-Hill Book Company.
- Knuth, Donald (1997). *The Art of Computer Programming*. **Vol. 2: Seminumerical Algorithms** (3rd ed.). Addison-Wesley. pp. 486–488 in section 4.6.4. ISBN 0-201-89684-2.
- Kripasagar, Venkat (March 2008). "Efficient Micro Mathematics – Multiplication and Division Techniques for MCUs". *Circuit Cellar magazine* (212): p. 60.
- Yoshio, Mikami (1913). *The Development of Mathematics in China and Japan* (1st ed.). Chelsea Publishing Co reprint. pp. 74-77.
- Ulrich, Librecht (2005). *Chinese Mathematics in the Thirteenth Century* (2rd ed.). Dover. pp. 175-211. ISBN 0-486-44619-0.

External links

- Weisstein, Eric W., " Horner's method (http://mathworld.wolfram.com/HornersMethod.html)" from MathWorld.
- Module for Horner's Method by John H. Mathews (http://math.fullerton.edu/mathews/n2003/HornerMod.html)

Sturm's theorem

In mathematics, the **Sturm's sequence** of a polynomial p is a sequence of polynomials associated to p and its derivative by a variant of Euclid's algorithm for polynomials. **Sturm's theorem** expresses the number of distinct real roots of p located in an interval in terms of the number of changes of signs of the values of the Sturm's sequence at the bounds of the interval. Applied to the interval of all the real numbers, it gives the total number of real roots of p.

Whereas the fundamental theorem of algebra readily yields the overall number of complex roots, counted with multiplicity, Sturm's theorem yields the number of distinct real roots and locates them in intervals. By subdividing the intervals containing some roots, it allows eventually to isolate the roots in arbitrary small intervals each containing exactly one root. This yields a symbolic root finding algorithm, that is available in most computer algebra systems, although some more efficient methods are now usually preferred (see below).

Sturm's sequence and Sturm's theorems are named after Jacques Charles François Sturm.

Sturm chains

A **Sturm chain** or **Sturm sequence** is a finite sequence of polynomials

$$p_0, p_1, \cdots, p_m$$

of decreasing degree with these following properties:

- $p_0 = p$ is square free (no square factors, i.e., no repeated roots);
- if $p(\xi) = 0$, then $\operatorname{sign}(p_1(\xi)) = \operatorname{sign}(p'(\xi))$;
- if $p_i(\xi) = 0$ for $0 < i < m$ then $\operatorname{sign}(p_{i-1}(\xi)) = -\operatorname{sign}(p_{i+1}(\xi))$;
- p_m does not change its sign.

It can be noted that Sturm's sequence is a modification of Fourier's sequence.

To obtain a Sturm chain, Sturm himself proposed to choose the intermediary results when applying Euclid's algorithm to *p* and its derivative:

$$\begin{aligned}
p_0(x) &:= p(x), \\
p_1(x) &:= p'(x), \\
p_2(x) &:= -\operatorname{rem}(p_0, p_1) = p_1(x)q_0(x) - p_0(x), \\
p_3(x) &:= -\operatorname{rem}(p_1, p_2) = p_2(x)q_1(x) - p_1(x), \\
&\vdots \\
0 &= -\operatorname{rem}(p_{m-1}, p_m).
\end{aligned}$$

That is, successively take the remainders with polynomial division and change their signs. Since $\deg(p_{i+1}) < \deg(p_i)$ for $0 \le i < m$, the algorithm terminates. The final polynomial, p_m, is the greatest common divisor of *p* and its derivative. If *p* is square free, it shares no roots with its derivative, hence p_m will be a non-zero constant polynomial. The Sturm chain, called the canonical Sturm chain, then is

$$p_0, p_1, p_2, \ldots, p_m.$$

If *p* is not square-free, this sequence does not formally satisfy the definition of a Sturm chain above, nevertheless it still satisfies the conclusion of Sturm's theorem (see below).

Statement

Let $p_0 = p, p_1, \ldots, p_m$ be a Sturm chain, where *p* is a square-free polynomial, and let σ(ξ) denote the number of sign changes (zeroes are not counted) in the sequence

$$p_0(\xi), p_1(\xi), p_2(\xi), \ldots, p_m(\xi).$$

Sturm's theorem then states that for two real numbers $a < b$, the number of distinct roots of *p* in the half-open interval $(a,b]$ is $\sigma(a) - \sigma(b)$.

The non-square-free case

Let $p_0 = p, p_1, \ldots, p_m$ be the canonical Sturm sequence of a polynomial *p*, not necessarily square-free. Then $\sigma(a) - \sigma(b)$ is the number of distinct roots of *p* in $(a,b]$ whenever $a < b$ are real numbers such that neither *a* nor *b* is a multiple root of *p*.

Proof

Polynomials are continuous functions, and any sign change must occur at a root, so consider the behavior of a Sturm chain around the roots of its constituent polynomials.

First, note that two adjacent polynomials can share a common root only when it is a multiple root of *p* (in which case it is a root of every p_i). Indeed, if p_i and p_{i-1} are both zero at ξ, then p_{i+1} also have to be zero at ξ, since $\operatorname{sign}(p_{i-1}(\xi)) = -\operatorname{sign}(p_{i+1}(\xi))$. The zero then propagates recursively up and down the chain, so that ξ is a root of all the polynomials p_0, ..., p_m.

Next, consider roots of polynomials in the interior (i.e., $0 < i < m$) of the Sturm chain that are not multiple roots of *p*. If $p_i(\xi) = 0$, then from the previous paragraph it is true that $p_{i-1}(\xi) \neq 0$ and $p_{i+1}(\xi) \neq 0$. Furthermore, $\operatorname{sign}(p_{i-1}(\xi)) = -\operatorname{sign}(p_{i+1}(\xi))$, so in the vicinity of ξ there is a single sign change across p_{i-1}, p_i, p_{i+1}. In other words, sign changes in the interior of the Sturm chain do not affect the total number of sign changes across the chain.

So only roots of the original polynomial, at the top of the chain, can affect the total number of sign changes. Consider a root ξ, so $p(\xi) = 0$, and assume first that it is a simple root. Then *p*'s derivative, which is p_1, must be non-zero at ξ, so *p* must be either increasing or decreasing at ξ. If it's increasing, then its sign is changing from

negative to positive when moving from left to right while its derivative (again, p_1) is positive, so the total number of sign changes decreases by one. Conversely, if it's decreasing, then its sign changes from positive to negative while its derivative is negative, so again the total number of sign changes decreases by one.

Finally, let ξ be a multiple root of p, and let p_0, ..., p_m be the canonical Sturm chain. Let $d = \gcd(p,p')$, $q = p/d$, and let q_0, ..., $q_{m'}$ be the canonical Sturm chain of q. Then $m = m'$ and $p_i = q_i d$ for every i. In particular, $\sigma(x)$ is the same for both chains whenever x is not a root of d. Then the number of sign changes (in either chain) around ξ decreases by one, since ξ is a simple root of q.
In summary, only sign changes at roots of the original polynomial affect the total number of sign changes across the chain, and the total number of sign changes always decreases by one as roots are passed. The theorem follows directly.

History section and other related methods

For counting and isolating the real roots, other methods are usually preferred, because they are computationally more efficient; these methods all use Descartes' rule of signs (just like Fourier[1] did back in 1820) and Vincent's theorem. Interestingly, the very first one of those methods[2] was initially called "modified Uspensky's algorithm" by its inventors, but it was later shown that there is no Uspensky's method[3]; afterwards, people started calling it either "Collins-Akritas method"[4] or "Descartes' method" only to be shown that there is no Descartes' method[4] either. Finally, François Boulier, of the University of Lille [5], p. 24, gave it the name "Vincent-Collins-Akritas" (VCA for short) to also give credit to Vincent. VCA is a bisection method; there exists also a continued fractions method based on Vincent's theorem namely, the *Vincent-Akritas-Strzeboński* (VAS) method[6].

VAS is based on Budan's theorem whereas Sturm's method has been inspired by Fourier's theorem. In fact Sturm himself [7], p. 108, acknowledges the great influence Fourier's theorem had on him: « C'est en m'appuyant sur les principes qu'il a posés, et en imitant ses démonstrations, que j'ai trouvé les nouveaux théorèmes que je vais énoncer. » which translates to "It is by relying upon the principles he has laid out and by imitating his proofs that I have found the new theorems which I am about to announce."

Applications

Number of real roots

Sturm's theorem can be used to compute the total number of real roots a polynomial.

This may be done by choosing $-a=b=M$ where M is larger than the absolute value of every root. For example, a bound due to Cauchy says that all real roots of a polynomial with coefficients a_i are in the interval $[-M, M]$, where

$$M = 1 + \frac{\max_{i=0}^{n-1} |a_i|}{|a_n|}.$$

Although theoretically the above approach is the simplest, in practice bounds on the positive (only) roots of polynomials are used and the positive roots are isolated and evaluated first; the negative roots are treated by first substituting x by -x, then compute a new (positive root) bound to isolate and evaluate the negative roots. Sturm's method and VCA need to compute only one bound to isolate the positive roots. By contrast, to isolate the positive roots VAS needs to compute various positive bounds, for the various polynomials that appear in the process. Efficient bounds on the values of the positive roots are described in P. S. Vigklas' Ph.D. Thesis[8] and in [9].

Another method is computationally simpler. One can use the fact that for large x, the sign of

$$p(x) = a_n x^n + \cdots$$

is $\operatorname{sign}(a_n)$, whereas $\operatorname{sign}(p(-x))$ is $(-1)^n a_n$.. In this way, simply counting the sign changes in the leading coefficients in the Sturm chain readily gives the number of distinct real roots of a polynomial.

Sturm's theorem allows also to determine the multiplicity of a given root, say ξ. Indeed, suppose that $a < \xi < b$, with $\sigma(a) - \sigma(b) = 1$. Then, ξ has multiplicity k precisely when ξ is a root with multiplicity $k - 1$ of p_m (since it is the GCD of p and its derivative). Thus the multiplicity of ξ may be computed by recursively applying Sturm's theorem to p_m. However, this method is rarely used, as square-free factorization is computationally more efficient for this purpose.

Quotient

The remainder is needed to compute the chain using Euclid's algorithm. For two polynomials $p(x) = \sum_{k=0}^{i} p_k x^k$ and $\tilde{p}(x) = \sum_{k=0}^{i-1} \tilde{p}_k x^k$ this is accomplished (for non-vanishing $\tilde{p}_{i-1}$) by

$$\operatorname{rem}(p, \tilde{p}) = \tilde{p}_{i-1} \cdot p(x) - p_i \cdot \tilde{p}(x) \left(x + \frac{p_{i-1}}{p_i} - \frac{\tilde{p}_{i-2}}{\tilde{p}_{i-1}} \right),$$

where the quotient is built solely of the first two leading coefficients.

Generalized Sturm chains

Let ξ be in the compact interval $[a,b]$. A **generalized Sturm chain** over $[a,b]$ is a finite sequence of real polynomials $(X_0, X_1, \ldots, X_r)$ such that:

1. $X(a)X(b) \neq 0$
2. $\operatorname{sign}(X_r)$ is constant on $[a,b]$
3. If $X_i(\xi) = 0$ for $1 \leq i \leq r-1$, then $X_{i-1}(\xi)X_{i+1}(\xi) < 0$.

One can check that each Sturm chain is indeed a generalized Sturm chain.

References

[1] Fourier, Jean Baptiste Joseph (1820). "Sur l'usage du théorème de Descartes dans la recherche des limites des racines" (http://ia600309.us.archive.org/22/items/bulletindesscien20soci/bulletindesscien20soci.pdf). *Bulletin des Sciences, par la Société Philomatique de Paris*: 156-165. .

[2] Collins, George E.; Alkiviadis G. Akritas (1976). *Polynomial Real Root Isolation Using Descartes' Rule of Signs* (http://doi.acm.org/10.1145/800205.806346). SYMSAC '76, Proceedings of the third ACM symposium on Symbolic and algebraic computation. Yorktown Heights, NY, USA: ACM. pp. 272-275. .

[3] Akritas, Alkiviadis G. (1986). *There is no "Uspensky's Method"* (http://dl.acm.org/citation.cfm?id=32457). In: Proceedings of the fifth ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '86, Waterloo, Ontario, Canada), pp. 88–90. .

[4] Akritas, Alkiviadis G. (2008). *There is no "Descartes' method"* (http://inf-server.inf.uth.gr/~akritas/articles/71.pdf). In: M.J.Wester and M. Beaudin (Eds), Computer Algebra in Education, AullonaPress, USA, pp. 19-35. .

[5] Boulier, François (2010). *Systèmes polynomiaux : que signifie « résoudre » ?* (http://www.lifl.fr/~boulier/polycopies/resoudre.pdf). Université Lille 1. .

[6] Akritas, Alkiviadis G.; A.W. Strzeboński, P.S. Vigklas (2008). "Improving the performance of the continued fractions method using new bounds of positive roots" (http://www.lana.lt/journal/30/Akritas.pdf). *Nonlinear Analysis: Modelling and Control* **13**: 265–279. .

[7] Hourya, Benis-Sinaceur (1988). "Deux moments dans l'histoire du Théorème d'algèbre de Ch. F. Sturm" (http://www.persee.fr/web/revues/home/prescript/article/rhs_0151-4105_1988_num_41_2_4093). *In: Revue d'histoire des sciences* **41** (2): 99-132. .

[8] Vigklas, Panagiotis, S. (2010). *Upper bounds on the values of the positive roots of polynomials* (http://www.inf.uth.gr/images/PHDTheses/phd_thesis_vigklas.pdf). Ph. D. Thesis, University of Thessaly, Greece. .

[9] Akritas, Alkiviadis, G. (2009). "Linear and Quadratic Complexity Bounds on the Values of the Positive Roots of Polynomials" (http://www.jucs.org/jucs_15_3/linear_and_quadratic_complexity). *Journal of Universal Computer Science* **15** (3): 523-537. .

- Sylvester, J. J. (1853). "On a theory of the syzygetic relations of two rational integral functions, comprising an application to the theory of Sturm's functions, and that of the greatest algebraical common measure". *Phil. Trans. Roy. Soc. London* **143**: 407–548. JSTOR 108572.
- Thomas, Joseph Miller (1941). "Sturm's theorem for multiple roots". *National Mathematics Magazine* **15** (8): 391–394. JSTOR 3028551. MR0005945.

- Heindel, Lee E. (1971), "Integer arithmetic algorithms for polynonial real zero determination", *Proc. SYMSAC '71*: 415, doi:10.1145/800204.806312, MR0300434
- Panton, Don B.; Verdini, William A. (1981). "A fortran program for applying Sturm's theorem in counting internal rates of return". *J. Financ. Quant. Anal.* **16** (3): 381–388. JSTOR 2330245.
- Akritas, Alkiviadis G. (1982). "Reflections on a pair of theorems by Budan and Fourier". *Math. Mag.* **55** (5): 292–298. JSTOR 2690097. MR0678195.
- Petersen, Paul (1991). "Multivariate Sturm theory". *Lecture Notes in Comp. Science* **539**: 318–332. doi:10.1007/3-540-54522-0_120. MR1229329.
- Yap, Chee (2000). *Fundamental Problems in Algorithmic Algebra* (http://www.cs.nyu.edu/yap/book/berlin/). Oxford University Press. ISBN 0-19-512516-9.

External links

- C code (http://tog.acm.org/resources/GraphicsGems/gems/Sturm/) from Graphic Gems by D.G. Hook and P.R. McAree.

Vincent's theorem

This page is part of **WikiProject Mathematics**.

Shortcut: WP:WPMER

In mathematics, **Vincent's theorem**, named after Alexandre Joseph Hidulph Vincent, is a little-known theorem that was (almost) totally forgotten, having been overshadowed by Sturm's theorem. Even though Vincent's theorem is of great interest because it can be used to isolate the real roots of polynomials with rational coefficients, it cannot be found in any of the classical books on *Theory of Equations* (of the 20th century), except for Uspensky's book. Two variants of this theorem are presented along with several (continued fractions as well as bisection) real root isolation methods that are derived from it.

Sign variation

Let $c_0, c_1, c_2, \cdots$ be a finite or infinite sequence of real numbers. Suppose $l < r$ and the following conditions hold:

1. If $r = l + 1$ the numbers c_l and c_r have opposite signs.
2. If $r \geq l + 2$ the numbers $c_{l+1}, \ldots, c_{r-1}$ are all zero and the numbers c_l and c_r have opposite signs.

This is called a *sign variation* or *sign change* between the numbers c_l and c_r.

Vincent's theorem

Two versions of this theorem are presented: the *continued fractions* version due to Vincent,[1][2] and the *bisection* version due to Alesina and Galuzzi.[3][4]

This statement of the *continued fractions* version can be found also in the Wikipedia article Budan's theorem.

Vincent's theorem: Continued fractions version (1834 and 1836)

If in a polynomial equation with rational coefficients and without multiple roots, one makes successive transformations of the form

$$x = a_1 + \frac{1}{x'}, \quad x' = a_2 + \frac{1}{x''}, \quad x'' = a_3 + \frac{1}{x'''}, \ldots$$

where $a_1, a_2, a_3, \ldots$ are any positive numbers greater than or equal to one, then the resulting transformed equation either has zero sign variations or it has a single sign variation. In the first case there is no root, whereas in the second case the equation has a single positive real root represented by the continued fraction:[1][2][5]

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots}}}$$

Vincent's theorem: Bisection version (Alesina and Galuzzi 2000)

Let $p(x)$ be a real polynomial of degree $deg(p)$ which has only simple roots. It is possible to determine a positive quantity δ so that for every pair of positive real numbers a, b with $|b - a| < \delta$, every transformed polynomial of the form

$$f(x) = (1+x)^{deg(p)} p\left(\frac{a+bx}{1+x}\right) \quad \textbf{(1)}$$

has exactly 0 or 1 sign variations. The second case is possible if and only if $p(x)$ has a single root within (a, b).

The Alesina-Galuzzi "a_b roots test"

From equation (**1**) the following criterion is obtained for determining whether a polynomial has any roots in the interval (a, b):

Perform on $p(x)$ the substitution $x \leftarrow \frac{a+bx}{1+x}$ and count the number of sign variations in the sequence of coefficients of the transformed polynomial; this number gives an *upper bound* on the number of real roots $p(x)$ has inside the open interval (a, b). More precisely, the number $\rho_{ab}(p)$ of real roots in the open interval (a, b) — multiplicities counted — of the polynomial $p(x) \in \mathbb{R}[x]$, of degree $deg(p)$, is bounded above by the number of sign variations $var_{ab}(p)$, where

$$var_{ab}(p) = var((1+x)^{deg(p)} p\left(\frac{a+bx}{1+x}\right))$$

and $var_{ab}(p) = var_{ba}(p) \geq \rho_{ab}(p)$. As in the case of Descartes' rule of signs if $var_{ab}(p) = 0$ it follows that $\rho_{ab}(p) = 0$ and if $var_{ab}(p) = 1$ it follows that $\rho_{ab}(p) = 1$.

A special case of the Alesina-Galuzzi **"a_b roots test"** is Budan's **"0_1 roots test"**.

Sketch of a Proof

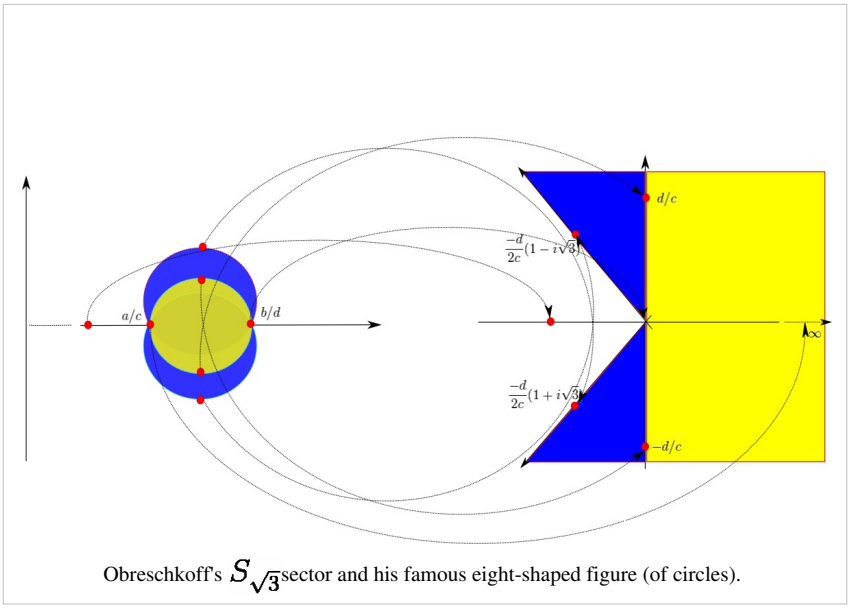
A detailed discussion of Vincent's theorem, its extension, the geometrical interpretation of the transformations involved and three different proofs can be found in the work by Alesina and Galuzzi.[3][4] A fourth proof is due to Ostrowski[6] who rediscovered a special case of a theorem stated by Obreschkoff,[7] p. 81, back in 1920-1923.

To prove (both versions of) Vincent's theorem Alesina and Galuzzi show that after a series of transformations mentioned in the theorem, a polynomial with one positive root will eventually have one sign variation. To show this they use the following corollary to the theorem by Obreschkoff of 1920-1923 mentioned earlier; that is, the following corollary gives the necessary conditions under which a polynomial with one positive root has exactly one sign variation in the sequence of its coefficients; see also the corresponding figure.

Corollary (to Obreschkoff's cone or sector theorem, 1920-1923[7] p. 81): If a real polynomial has one simple root x_0, and all other (possibly multiple) roots lie in the sector

$$S_{\sqrt{3}} = \{x = -\alpha + i\beta \ \mid \ \alpha > 0 \ \text{ and } \ \beta^2 \leq 3\alpha^2\}$$

then the sequence of its coefficients has exactly one sign variation.



Obreschkoff's $S_{\sqrt{3}}$ sector and his famous eight-shaped figure (of circles).

Consider now the Möbius transformation $M(x) = \dfrac{ax+b}{cx+d}$, $a, b, c, d \in \mathbb{Z}_{>0}$ and the three circles shown in the corresponding figure; assume that $\dfrac{a}{c} < \dfrac{b}{d}$.

- The (yellow) circle $\left|x - \dfrac{(a/c + b/d)}{2}\right| = \dfrac{(b/d - a/c)}{2}$ whose diameter lies on the real axis, with endpoints $\dfrac{a}{c}$ and $\dfrac{b}{d}$, is mapped by the inverse Möbius transformation $M^{-1}(x) = \dfrac{dx-b}{-cx+a}$ onto the imaginary axis. For example the point $\dfrac{(a/c + b/d)}{2} + i\dfrac{(b/d - a/c)}{2}$ gets mapped onto the point $-i\dfrac{d}{c}$. The exterior points get mapped onto the half-plane with $Re(x) < 0$.
- The two circles (only their blue crescents are visible) with center $\dfrac{(a/c + b/d)}{2} \pm i\dfrac{(b/d - a/c)}{2\sqrt{3}}$ and radius $\dfrac{(b/d - a/c)}{\sqrt{3}}$ are mapped by the inverse Möbius transformation $M^{-1}(x) = \dfrac{dx-b}{-cx+a}$ onto the lines $Im(x) = \pm\sqrt{3}Re(x)$. For example the point $\dfrac{(a/c + b/d)}{2} - 3i\dfrac{(b/d - a/c)}{2\sqrt{3}}$ gets mapped to the point

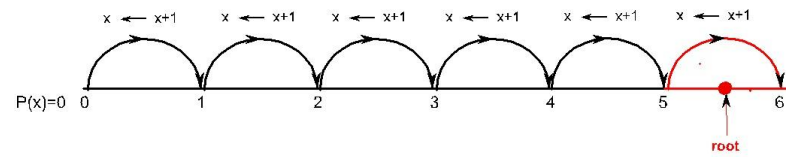
$\frac{-d}{2c}(1-i\sqrt{3})$. The exterior points (those outside the eight-shaped figure) get mapped onto the $S_{\sqrt{3}}$sector.

From the above it becomes obvious that if a polynomial has a single positive root inside the eight-shaped figure and all other roots are outside of it, it will present one sign variation in the sequence of its coefficients. This also guarantees the termination of the process.

Historical background

Early applications of Vincent's theorem

Vincent presented in both of his papers[1][2] several examples showing precisely how his theorem is to be used in order to isolate the real roots of polynomials with continued fractions. However the resulting method had exponential computing time, a fact that must have been realized then, as was also realized by Uspensky[8] p. 136, a century later.



Vincent's search for a root (applying Budan's theorem)

The exponential nature of Vincent's algorithm is due to the way the partial quotients a_i (in Vincent's theorem) are computed. That is, to compute each partial quotient a_i (that is, to locate where the roots lie on the *x*-axis) Vincent uses Budan's theorem as a "no roots test"; in other words, to find the integer part of a root Vincent performs successive substitutions of the form $x \leftarrow x+1$ and stops only when the polynomials $p(x)$ and $p(x+1)$ differ in the number of sign variations in the sequence of their coefficients (i.e when the number of sign variations of $p(x+1)$ is decreased).[1][2]

See the corresponding diagram where the root lies in the interval $(5,6)$. It can be easily inferred that, if the root is far away from the origin, it will take a lot of time to find its integer part this way; hence the exponential nature of Vincent's method. Below there is an explanation of how this drawback is overcome.

Disappearance of Vincent's theorem

Vincent was the last author in the 19th century to use his theorem for the isolation of the real roots of a polynomial.

The reason for that was the appearance of Sturm's theorem in 1827 which solved the real root isolation problem in polynomial time, by defining the precise number of real roots a polynomial has in a real open interval (a, b). The resulting (Sturm's) method for computing the real roots of polynomials has been the only one widely known and used ever since – up to about 1980, when it was replaced (in almost all computer algebra systems) by methods derived from Vincent's theorem, the fastest one being the Vincent–Akritas–Strzeboński (VAS) method.[9]

Serret included in his Algebra[10], pp 363–368, Vincent's theorem along with its proof and directed all interested readers to Vincent's papers for examples on how it is used. Serret was the last author to mention Vincent's theorem in the 19th century.

Comeback of Vincent's theorem

In the 20th century Vincent's theorem cannot be found in any of the theory of equations books; the only exceptions are the books by Uspensky[8] and Obreschkoff,[7] where in the second there is just the statement of the theorem.

It was in Uspensky's book[8] that Akritas found Vincent's theorem and made it the topic of his Ph.D. Thesis "Vincent's Theorem in Algebraic Manipulation", North Carolina State University, USA, 1978. A major achievement at the time was getting hold of Vincent's original paper of 1836, something that had eluded Uspensky — resulting thus in a great misunderstanding. Vincent's original paper of 1836 was made available to Akritas through the commendable efforts (interlibrary loan) of a librarian in the Library of the University of Wisconsin–Madison, USA.

Real root isolation methods derived from Vincent's theorem

Isolation of the real roots of a polynomial is the process of finding open disjoint intervals such that each contains exactly one real root and every real root is contained in some interval. According to the French school of mathematics of the 19th century, this is the first step in computing the real roots, the second being their **approximation** to any degree of accuracy; moreover, the focus is on the **positive** roots, because to isolate the **negative** roots of the polynomial $p(x)$ replace x by $-x$ ($x \leftarrow -x$) and repeat the process.

The continued fractions version of Vincent's theorem can be used to isolate the positive roots of a given polynomial $p(x)$ of degree $deg(p)$. To see this, represent by the Möbius transformation $M(x) = \dfrac{ax+b}{cx+d}$, $a, b, c, d \in \mathbb{N}$ the continued fraction that leads to a transformed polynomial

$$f(x) = (cx+d)^{deg(p)} p\left(\frac{ax+b}{cx+d}\right) \quad (2)$$

with one sign variation in the sequence of its coefficients. Then, the single positive root of $f(x)$ (in the interval $(0, \infty)$) corresponds to *that* positive root of $p(x)$ which is located in the open interval with endpoints $\dfrac{b}{d}$ and $\dfrac{a}{c}$.

These endpoints are *not* ordered and correspond to $M(0)$ and $M(\infty)$ respectively.

Therefore, to isolate the positive roots of a polynomial, all that has to be done is to compute — for *each* root — the variables a, b, c, d of the corresponding Möbius transformation $M(x) = \dfrac{ax+b}{cx+d}$ that leads to a transformed polynomial as in equation (**2**), with one sign variation in the sequence of its coefficients.

Crucial Observation: The variables a, b, c, d of a Möbius transformation $M(x) = \dfrac{ax+b}{cx+d}$ (in Vincent's theorem) leading to a transformed polynomial — as in equation (**2**) — with one sign variation in the sequence of its coefficients can be computed:

- either by *continued fractions*, leading to the *Vincent-Akritas-Strzebonski (VAS)* continued fractions method,[9]
- or by *bisection*, leading to (among others) the *Vincent-Collins-Akritas (VCA)* bisection method.[11]

The "bisection part" of this all important observation appeared as a special theorem in the papers by Alesina and Galuzzi.[3][4]

All methods described below (see the article on Budan's theorem for their historical background) need to compute (once) an upper bound, ub, on the values of the positive roots of the polynomial under consideration. Exception is the VAS method where additionally lower bounds, lb, need to be computed at almost every cycle of the main loop. To compute the lower bound lb of the polynomial $p(x)$ compute the upper bound ub of the polynomial $x^{deg(p)} p\left(\dfrac{1}{x}\right)$ and set $lb = \dfrac{1}{ub}$.

Excellent (upper and lower) bounds on the values of just the positive roots of polynomials have been developed by Akritas, Strzeboński and Vigklas based on previous work by Doru Stefanescu. They are described in P. S. Vigklas'

Ph.D. Thesis[12] and elsewhere.[13] These bounds have already been implemented in the computer algebra systems Mathematica, Sage, SymPy, Xcas etc.

All three methods described below follow the excellent presentation of François Boulier[14], p. 24.

Continued fractions method

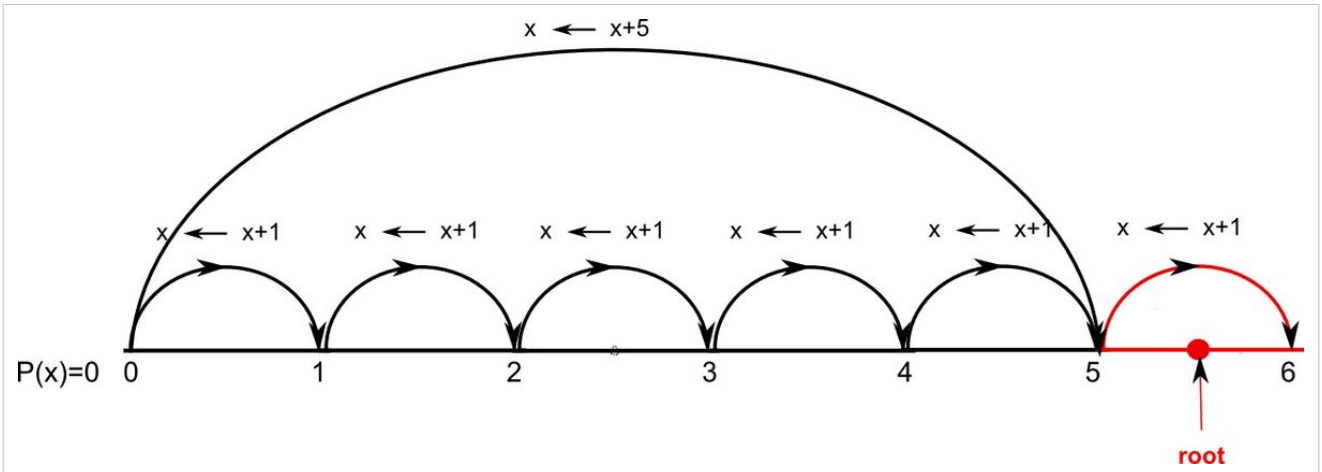
There is only one continued fractions method derived from Vincent's theorem. As has been stated above it started in the 1830s when Vincent presented in both of his papers[1][2] several examples showing precisely how his theorem is to be used in order to isolate the real roots of polynomials with continued fractions. However the resulting method had exponential computing time. Below is an explanation of how this method evolved.

Vincent–Akritas–Strzeboński (VAS, 2005)

This is the second method (after VCA) developed to handle the exponential behavior of Vincent's method.

The VAS continued fractions method is a *direct* implementation of Vincent's theorem. It was originally presented by Vincent in his 1834[1] and 1836[2] papers in an exponential form; namely, Vincent computed each partial quotient a_i by a series of *unit* increments $a_i \leftarrow a_i + 1$, which are equivalent to substitutions of the form $x \leftarrow x + 1$.

Vincent's method was converted into its polynomial complexity form by Akritas, who in his 1978 Ph.D. Thesis ("Vincent's theorem in algebraic manipulation", North Carolina State University, USA) computed each partial quotient a_i as the lower bound, lb, on the values of the positive roots of a polynomial; this is called the *ideal* positive lower root bound which computes the integer part of the smallest positive root (see the corresponding figure). To wit, now set $a_i \leftarrow lb$ or, equivalently, perform the substitution $x \leftarrow x + lb$, which takes about the same time as the substitution $x \leftarrow x + 1$.



VAS searching for a root: The *ideal* lower bound is 5, hence $x \leftarrow x + 5$.

Finally, since the ideal positive lower root bound does not exist, Strzeboński[15] introduced in 2005 the substitution $x \leftarrow lb_{computed} * x$, whenever $lb_{computed} > 16$; in general $lb > lb_{computed}$ and the value 16 was determined experimentally. Moreover, it has been shown[15] that the VAS (continued fractions) method is faster than the fastest implementation of the VCA (bisection) method[16], a fact that was confirmed[17] independently; more precisely, for the Mignotte polynomials of high degree VAS is about 50,000 times faster than the fastest implementation of VCA.

In 2007, Sharma[18] removed the hypothesis of the ideal positive lower bound and proved that VAS is still polynomial in time.

VAS is the default algorithm for root isolation in Mathematica, Sage, SymPy, Xcas.

For a comparison between Sturm's method and VAS use the functions realroot(poly) and time(realroot(poly)) of Xcas. By default, to isolate the real roots of poly realroot uses the VAS method; to use Sturm's method write

realroot(sturm, poly). See also the External links for a pointer to an iPhone/iPod/iPad application that does the same thing.

Here is how $VAS(p, M)$ works, where for simplicity Strzeboński's contribution is not included:

- Let $p(x)$ be a polynomial of degree $deg(p)$ such that $p(0) \neq 0$. To isolate its positive roots, associate with $p(x)$ the Möbius transformation $M(x) = x$ and repeat the following steps while there are pairs $\{p(x), M(x)\}$ to be processed.
- Use Descartes' rule of signs on $p(x)$ to compute (using the number *var* of sign variations in the sequence of its coefficients) the number of its roots inside the interval $(0, \infty)$. If there are no roots return the empty set, $\emptyset$ whereas if there is one root return the interval (a, b), where $a = min(M(0), M(\infty))$, and $b = max(M(0), M(\infty))$; if $b = \infty$ set $b = ub$, where ub is an upper bound on the values of the positive roots of $p(x)$.[12][13]
- If it turns out (from Descartes' rule of signs) that there are two or more roots inside the interval $(0, \infty)$, consider separately the roots of $p(x)$ which lie inside the interval $(0, 1)$ from those which lie inside the interval $(1, \infty)$. A special test has to be made for 1.
  - To guarantee that there will be roots inside the interval $(0, 1)$ the ideal lower bound, lb is used; that is the integer part of the smallest positive root is computed with the help of the lower bound,[12][13] $lb_{computed}$, on the values of the positive roots of $p(x)$. If $lb_{computed} > 1$, the substitution $x \leftarrow x + lb_{computed}$ is performed to $p(x)$ and $M(x)$, whereas if $lb_{computed} \leq 1$ use substitution(s) $x \leftarrow x + 1$ to find the integer part of the root(s).
  - To compute the roots inside the interval $(0, 1)$ perform the substitution $x \leftarrow \frac{1}{1+x}$ to $p(x)$ and $M(x)$ and process the pair $\{(1+x)^{deg(p)} p(\frac{1}{1+x}), M(\frac{1}{1+x})\}$, whereas to compute the roots in the interval $(1, \infty)$ perform the substitution $x \leftarrow 1 + x$ to $p(x)$ and $M(x)$ and process the pair $\{p(1+x), M(1+x)\}$. It may well turn out that 1 is a root of $p(x)$, in which case $M(1)$ is a root of the original polynomial and the isolation interval reduces to a point.

Below is a recursive presentation of $VAS(p, M)$.

VAS(p,M)

Input: A univariate, square-free polynomial $p(x) \in \mathbb{Z}[x], p(0) \neq 0$ and of degree $deg(p)$, and the Möbius transformation $M(x) = \frac{ax+b}{cx+d} = x$, where $a, b, c, d \in \mathbb{N}$.

Output: A list of isolating intervals of the positive roots of $p(x)$.

```
1 var ← the number of sign variations of p(x) // Descartes' rule of signs;
2 if var = 0 then RETURN ∅ ;
3  if var = 1 then RETURN {(a,b)} // a = min (M(0), M(∞)), b = max
(M(0), M(∞)), but if b = ∞ set b = ub, where ub is an upper bound on the
values of the positive roots of p(x);
4 lb ← the ideal lower bound on the positive roots of p(x);
5 if lb ≥ 1 then p ← p(x + lb), M ← M(x + lb);
6 p01 ← (x + 1)^deg(p) p(1/(x+1)), M01 ← M(1/(x+1)) // Look for real roots in (0,1);
7 m ← M(1) // Is 1 a root? ;
8 p1∞ ← p(x + 1), M1∞ ← M(x + 1) // Look for real roots in (1, +∞);
9 if p(1) ≠ 0 then
10 RETURN VAS(p01, M01) ∪ VAS(p1∞, M1∞)
11 else
```

12 **RETURN** $VAS(p_{01}, M_{01}) \cup \{[m, m]\} \cup VAS(p_{1\infty}, M_{1\infty})$

13 **end**

Remarks

- For simplicity Strzeboński's contribution is not included.
- In the above algorithm with each polynomial there is associated a Möbius transformation $M(x)$.
- In line 1 Descartes' rule of signs is applied.
- If lines 4 and 5 are removed from $VAS(p, M)$ the resulting algorithm is Vincent's exponential one.
- Any substitution performed on the polynomial $p(x)$ is also performed on the associated Möbius transformation $M(x)$ (lines 5 6 and 8).
- The isolating intervals are computed from the Möbius transformation in line 3, except for integer roots computed in line 7 (also 12).

Example of VAS(p,M)

Given the polynomial $p(x) = x^3 - 7x + 7$ the arguments of the VAS method are: $p(x) = x^3 - 7x + 7$ and $M(x) = x$.

$VAS(x^3 - 7x + 7, x)$

---

1 *var* $\leftarrow$ 2 // the number of sign variations in the sequence of coefficients of $p(x) = x^3 - 7x + 7$

4 $lb \leftarrow 1$ // the ideal lower bound — found using $lb_{computed}$ and substitution(s) $x \leftarrow x + 1$

5 $p \leftarrow x^3 + 3x^2 - 4x + 1, M \leftarrow x + 1$

6 $p_{01} \leftarrow x^3 - x^2 - 2x + 1, M_{01} \leftarrow \frac{x+2}{x+1}$

7 $m \leftarrow 1$

8 $p_{1\infty} \leftarrow x^3 + 6x^2 + 5x + 1, M_{1\infty} \leftarrow x + 2$

10 **RETURN** $VAS(x^3 - x^2 - 2x + 1, \frac{x+2}{x+1}) \cup VAS(x^3 + 6x^2 + 5x + 1, x + 2)$

List of isolation intervals: $\{\}$. List of pairs $\{poly, Moebius\ transformation\}$ to be processed: $\{\{x^3 - x^2 - 2x + 1, \frac{x+2}{x+1}\}, \{x^3 + 6x^2 + 5x + 1, x + 2\}\}$. Remove the first and process it.

$VAS(x^3 - x^2 - 2x + 1, \frac{x+2}{x+1})$

---

1 *var* $\leftarrow$ 2 // the number of sign variations in the sequence of coefficients of $p(x) = x^3 - x^2 - 2x + 1$

4 $lb \leftarrow 0$ // the ideal lower bound — found using $lb_{computed}$ and substitution(s) $x \leftarrow x + 1$

6 $p_{01} \leftarrow x^3 + x^2 - 2x - 1, M_{01} \leftarrow \frac{2x+3}{x+2}$

7 $m \leftarrow \frac{3}{2}$

8 $p_{1\infty} \leftarrow x^3 + 2x^2 - x - 1, M_{1\infty} \leftarrow \frac{x+3}{x+2}$

10 **RETURN** $VAS(x^3 + x^2 - 2x - 1, \frac{2x+3}{x+2}) \cup VAS(x^3 + 2x^2 - x - 1, \frac{x+3}{x+2})$

List of isolation intervals: $\{\}$. List of pairs $\{poly, Moebius\ transformation\}$ to be processed: $\{\{x^3+x^2-2x-1, \frac{2x+3}{x+2}\}, \{x^3+2x^2-x-1, \frac{x+3}{x+2}\}, \{x^3+6x^2+5x+1, x+2\}\}$. Remove the first and process it.

$VAS(x^3 + x^2 - 2x - 1, \frac{2x+3}{x+2})$

```
1 var ← 1 // the number of sign variations in the sequence of coefficients of
```
$p(x) = x^3 + x^2 - 2x - 1$

3 **RETURN** $\{(\frac{3}{2}, 2)\}$

List of isolation intervals: $\{(\frac{3}{2}, 2)\}$. List of pairs $\{poly, Moebius\ transformation\}$ to be processed: $\{\{x^3 + 2x^2 - x - 1, \frac{x+3}{x+2}\}, \{x^3 + 6x^2 + 5x + 1, x + 2\}\}$. Remove the first and process it.

$VAS(x^3 + 2x^2 - x - 1, \frac{x+3}{x+2})$

```
1 var ← 1 // the number of sign variations in the sequence of coefficients of
```
$p(x) = x^3 + 2x^2 - x - 1$

3 **RETURN** $\{(1, \frac{3}{2})\}$

List of isolation intervals: $\{(1, \frac{3}{2}), (\frac{3}{2}, 2)\}$. List of pairs $\{poly, Moebius\ transformation\}$ to be processed: $\{\{x^3 + 6x^2 + 5x + 1, x + 2\}\}$. Remove the first and process it.

$VAS(x^3 + 6x^2 + 5x + 1, x + 2)$

```
1 var ← 0 // the number of sign variations in the sequence of coefficients of
```
$p(x) = x^3 + 6x^2 + 5x + 1$

2 **RETURN** $\emptyset$

List of isolation intervals: $\{(1, \frac{3}{2}), (\frac{3}{2}, 2)\}$. List of pairs $\{poly, Moebius\ transformation\}$ to be processed: $\emptyset$. Finished.

Therefore, the two positive roots of the polynomial $p(x) = x^3 - 7x + 7$lie inside the isolation intervals $(1, \frac{3}{2})$ and $(\frac{3}{2}, 2)$. Each root can be approximated by (for example) bisecting the isolation interval — inside which it lies — until the difference of the endpoints is smaller than 10^{-6}; following this approach, the roots turn out to be $\rho_1 = 1.3569$and $\rho_2 = 1.69202$.

Bisection methods

There are various bisection methods derived from Vincent's theorem; they are all presented and compared elsewhere.[19] Here the two most important of them are described, namely, the Vincent-Collins-Akritas (VCA) method and the Vincent-Alesina-Galuzzi (VAG) method.

The Vincent-Alesina-Galuzzi (VAG) method is the simplest of all methods derived from Vincent's theorem but has the most time consuming test (in line 1) to determine if a polynomial has roots in the interval of interest; this makes it the slowest of the methods presented in this article.

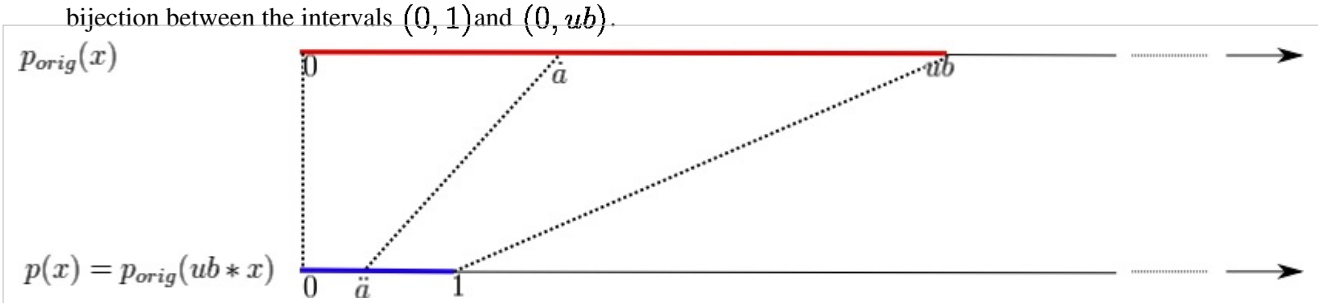
By contrast, the Vincent-Collins-Akritas (VCA) method is more complex but uses a simpler test (in line 1) than VAG. This along with certain improvements[16] have made VCA the fastest bisection method.

Vincent–Collins–Akritas (VCA, 1976)

This was the first method developed to overcome the exponential nature of Vincent's original approach, and has had quite an interesting history as far as its name is concerned. This method, which isolates the real roots, using Descartes' rule of signs and Vincent's theorem, had been originally called *modified Uspensky's algorithm* by its inventors Collins and Akritas[11]. After going through names like "Collins-Akritas method" and "Descartes' method" (too confusing if ones considers Fourier's article[20]), it was finally François Boulier, of Lille University, who gave it the name *Vincent-Collins-Akritas* (VCA) method[14], p. 24, based on the fact that "Uspensky's method" does not exist[21] and neither does "Descartes' method".[22] The best implementation of this method is due to Rouillier and Zimmerman,[16] and to this date, it is the fastest bisection method. It has the same worst case complexity as Sturm's algorithm, but is almost always much faster. It has been implemented in Maple's RootFinding package.

Here is how $VCA(p, (a, b))$ works:

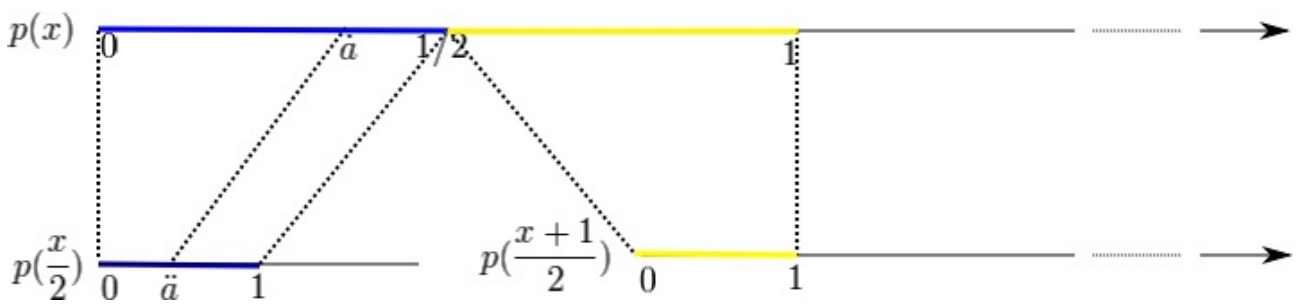
- Given a polynomial $p_{orig}(x)$ of degree $deg(p)$, such that $p_{orig}(0) \neq 0$, whose positive roots need to be isolated, first compute an upper bound,[12][13] ub on the values of these positive roots and set $p(x) = p_{orig}(ub * x)$ and $(a, b) = (0, ub)$. The positive roots of $p(x)$ all lie in the interval $(0, 1)$ and there is a bijection between them and the roots of $p_{orig}(x)$, which all lie in the interval $(a, b) = (0, ub)$ (see the corresponding figure); this bijection is expressed by $\alpha_{(a,b)} = a + \alpha_{(0,1)}(b - a)$. Likewise, there is a bijection between the intervals $(0, 1)$ and $(0, ub)$.



Bijection between the roots of $p_{orig}(x)$ and $p(x)$.

- Repeat the following steps while there are pairs $\{p(x), (a, b)\}$ to be processed.
- Use Budan's "**0_1 roots test**" on $p(x)$ to compute (using the number *var* of sign variations in the sequence of its coefficients) the number of its roots inside the interval $(0, 1)$. If there are no roots return the empty set, $\emptyset$ and if there is one root return the interval (a, b).
- If it turns out (from Budan's "**0_1 roots test**") that there are two or more roots inside the interval $(0, 1)$, cut it in half and consider separately the roots of $p(x)$ which lie inside the interval $(0, \frac{1}{2})$ — and which correspond to the roots of $p_{orig}(x)$ inside the interval $(a, \frac{a+b}{2})$ — from those which lie inside the interval $(\frac{1}{2}, 1)$ — and which correspond to the roots of $p_{orig}(x)$ inside the interval $(\frac{a+b}{2}, b)$; that is, process, respectively, the

pairs $\{2^{deg(p)}p(\frac{x}{2}), (a, \frac{a+b}{2})\}$ and $\{2^{deg(p)}p(\frac{x+1}{2}), (\frac{a+b}{2}, b)\}$ (see the corresponding figure). It may well turn out th isolation interval reduces to a point.



Bijections between the roots of $p(x)$ and those of $p(\frac{x}{2})$ and $p(\frac{x+1}{2})$.

Below is a recursive presentation of the original algorithm $VCA(p, (a, b))$.

VCA(p,(a,b))

Input: A univariate, square-free polynomial $p(ub * x) \in \mathbb{Z}[x], p(0) \neq 0$ of degree $deg(p)$, and the open interval $(a, b) = (0, ub)$, where ub is an upper bound on the values of the positive roots of $p(x)$. (The positive roots of $p(ub * x)$ are all in the open interval $(0, 1)$).

Output: A list of isolating intervals of the positive roots of $p(x)$

1 *var* ← the number of sign variations of $(x+1)^{deg(p)}p(\frac{1}{x+1})$ // Budan's **"0_1 roots test";**

2 **if** *var* = 0 **then RETURN** $\emptyset$;

3 **if** *var* = 1 **then RETURN** {(a, b)};

4 $p_{0\frac{1}{2}} \leftarrow 2^{deg(p)}p(\frac{x}{2})$ // Look for real roots in $(0, \frac{1}{2})$;

5 $m \leftarrow \frac{a+b}{2}$ // Is $\frac{1}{2}$ a root? ;

6 $p_{\frac{1}{2}1} \leftarrow 2^{deg(p)}p(\frac{x+1}{2})$ // Look for real roots in $(\frac{1}{2}, 1)$;

7 **if** $p(\frac{1}{2}) \neq 0$ **then**

8 **RETURN** $VCA(p_{0\frac{1}{2}}, (a, m)) \cup VCA(p_{\frac{1}{2}1}, (m, b))$

9 **else**

10 **RETURN** $VCA(p_{0\frac{1}{2}}, (a, m)) \cup \{[m, m]\} \cup VCA(p_{\frac{1}{2}1}, (m, b))$

11 **end**

Remark

- In the above algorithm with each polynomial there is associated an interval (a, b). As shown elsewhere,[22] p.11, a Möbius transformation can also be associated with each polynomial in which case VCA looks more like VAS.
- In line 1 Budan's "**0_1 roots test**" is applied.

Example of VCA(p,(a,b))

Given the polynomial $p_{orig}(x) = x^3 - 7x + 7$and considering as an upper bound[12][13] on the values of the positive roots $ub = 4$the arguments of the VCA method are: $p(x) = 64x^3 - 28x + 7$and $(a, b) = (0, 4)$.

$VCA(64x^3 - 28x + 7, (0, 4))$

---

1 *var* $\leftarrow$ 2 // the number of sign variations in the sequence of coefficients of $(x+1)^3 p(\frac{1}{x+1}) = 7x^3 - 7x^2 - 35x + 43$

4 $p_{0\frac{1}{2}} \leftarrow 64x^3 - 112x + 56$

5 $m \leftarrow 2$

6 $p_{\frac{1}{2}1} \leftarrow 64x^3 + 192x^2 + 80x + 8$

7 $p(\frac{1}{2}) = 1$

8 **RETURN** $VCA(64x^3 - 112x + 56, (0, 2)) \cup VCA(64x^3 + 192x^2 + 80x + 8, (2, 4))$

List of isolation intervals: $\{\}$. List of pairs $\{poly, interval\}$ to be processed: $\{\{64x^3 - 112x + 56, (0, 2)\}, \{64x^3 + 192x^2 + 80x + 8, (2, 4)\}\}$. Remove the first and process it.

$VCA(64x^3 - 112x + 56, (0, 2))$

---

1 *var* $\leftarrow$ 2 // the number of sign variations in the sequence of coefficients of $(x+1)^3 p(\frac{1}{x+1}) = 56x^3 + 56x^2 - 56x + 8$

4 $p_{0\frac{1}{2}} \leftarrow 64x^3 - 448x + 448$

5 $m \leftarrow 1$

6 $p_{\frac{1}{2}1} \leftarrow 64x^3 + 192x^2 - 256x + 64$

7 $p(\frac{1}{2}) = 8$

8 **RETURN** $VCA(64x^3 - 448x + 448, (0, 1)) \cup VCA(64x^3 + 192x^2 - 256x + 64, (1, 2))$

List of isolation intervals: $\{\}$. List of pairs $\{poly, interval\}$ to be processed: $\{\{64x^3 - 448x + 448, (0, 1)\}, \{64x^3 + 192x^2 - 256x + 64, (1, 2)\}, \{64x^3 + 192x^2 + 80x + 8, (2, 4)\}\}$. Remove the first and process it.

$VCA(64x^3 - 448x + 448, (0, 1))$

---

1 *var* $\leftarrow$ 0 // the number of sign variations in the sequence of coefficients of $(x+1)^3 p(\frac{1}{x+1}) = 448x^3 + 896x^2 + 448x + 64$

2 **RETURN** $\emptyset$

List of isolation intervals: $\{\}$. List of pairs $\{poly, interval\}$ to be processed: $\{\{64x^3+192x^2-256x+64,(1,2)\},\{64x^3+192x^2+80x+8,(2,4)\}\}$. Remove the first and process it.

$VCA(64x^3+192x^2-256x+64,(1,2))$

1 *var* $\leftarrow$ 2 // the number of sign variations in the sequence of coefficients of $(x+1)^3p(\frac{1}{x+1}) = 64x^3-64x^2-128x+64$

4 $p_{0\frac{1}{2}} \leftarrow 64x^3+384x^2-1024x+512$

5 $m \leftarrow \frac{3}{2}$

6 $p_{\frac{1}{2}1} \leftarrow 64x^3+576x^2-64x-64$

7 $p(\frac{1}{2}) = -8$

8 **RETURN** $VCA(64x^3+384x^2-$

$VCA(64x^3+576x^2-64x-64,(\frac{3}{2},2))$

List of isolation intervals: $\{\}$. List of pairs $\{poly, interval\}$ to be processed: $\{\{64x^3+384x^2-1024x+512,(1,\frac{3}{2})\},\{64x^3+576x^2-64x-64,(\frac{3}{2},2)\},\{64x^3+192x^2+80x+8,(2,4)\}\}$. Remove the first and process it.

$VCA(64x^3+384x^2-1024x+512,(1,\frac{3}{2}))$

1 *var* $\leftarrow$ 1 // the number of sign variations in the sequence of coefficients of $(x+1)^3p(\frac{1}{x+1}) = 512x^3+512x^2-128x-64$

3 **RETURN** $\{(1,\frac{3}{2})\}$

List of isolation intervals: $\{(1,\frac{3}{2})\}$. List of pairs $\{poly, interval\}$ to be processed: $\{\{64x^3+576x^2-64x-64,(\frac{3}{2},2)\},\{64x^3+192x^2+80x+8,(2,4)\}\}$. Remove the first and process it.

$VCA(64x^3+576x^2-64x-64,(\frac{3}{2},2))$

1 *var* $\leftarrow$ 1 // the number of sign variations in the sequence of coefficients of $(x+1)^3p(\frac{1}{x+1}) = -64x^3-256x^2+256x+512$

3 **RETURN** $\{(\frac{3}{2},2)\}$

List of isolation intervals: $\{(1, \frac{3}{2}), (\frac{3}{2}, 2)\}$. List of pairs $\{poly, interval\}$ to be processed: $\{\{64x^3 + 192x^2 + 80x + 8, (2, 4)\}\}$. Remove the first and process it.

$VCA(64x^3 + 192x^2 + 80x + 8, (2, 4))$

```
1 var ← 0 // the number of sign variations in the sequence of coefficients of
```
$(x+1)^3 p(\frac{1}{x+1}) = 8x^3 + 104x^2 + 376x + 344$

2 **RETURN** $\emptyset$

List of isolation intervals: $\{(1, \frac{3}{2}), (\frac{3}{2}, 2)\}$. List of pairs $\{poly, interval\}$ to be processed: $\emptyset$. Finished.

Therefore, the two positive roots of the polynomial $p(x) = x^3 - 7x + 7$ lie inside the isolation intervals $(1, \frac{3}{2})$ and $(\frac{3}{2}, 2)$. Each root can be approximated by (for example) bisecting the isolation interval — inside which it lies — until the difference of the endpoints is smaller than 10^{-6}; following this approach, the roots turn out to be $\rho_1 = 1.3569$ and $\rho_2 = 1.69202$.

Vincent–Alesina–Galuzzi (VAG, 2000)

This was developed last and is the simplest real root isolation method derived from Vincent's theorem.

Here is how $VAG(p, (a, b))$ works:

- Given a polynomial $p(x)$ of degree $deg(p)$, such that $p(0) \neq 0$, whose positive roots need to be isolated, first compute an upper bound,[12][13] ub on the values of these positive roots and set $(a, b) = (0, ub)$. The positive roots of $p(x)$ all lie in the interval (a, b).
- Repeat the following steps while there are intervals (a, b) to be processed; in this case the polynomial $p(x)$ stays the same.
- Use the Alesina-Galuzzi "**a_b roots test**" on $p(x)$ to compute (using the number *var* of sign variations in the sequence of its coefficients) the number of its roots inside the interval (a, b). If there are no roots return the empty set, $\emptyset$ and if there is one root return the interval (a, b).
- If it turns out (from the Alesina-Galuzzi "**a_b roots test**") that there are two or more roots inside the interval (a, b), cut it in half and consider separately the roots of $p(x)$ which lie inside the interval $(a, \frac{a+b}{2})$ from those which lie inside the interval $(\frac{a+b}{2}, b)$; that is, process, respectively, the intervals $(a, \frac{a+b}{2})$ and $(\frac{a+b}{2}, b)$. It may well turn out that $\frac{a+b}{2}$ is a root of $p(x)$, in which case the isolation interval reduces to a point.

Below is a recursive presentation of $VAG(p, (a, b))$.

VAG(p,(a,b))

Input: A univariate, square-free polynomial $p(x) \in \mathbb{Z}[x], p(0) \neq 0$ of degree $deg(p)$, and the open interval $(a, b) = (0, ub)$, where ub is an upper bound on the values of the positive roots of $p(x)$.

Output: A list of isolating intervals of the positive roots of $p(x)$.

1 *var* ← the number of sign variations of $(x+1)^{deg(p)} p(\frac{a+bx}{1+x})$ // The Alesina-Galuzzi **"a_b roots test";**

2 **if** *var* = 0 **then RETURN** $\emptyset$;

3 **if** *var* = 1 **then RETURN** {(a, b)};

4 $m \leftarrow \frac{a+b}{2}$ // Subdivide the interval (a,b) in two equal parts;

5 **if** $p(m) \neq 0$ **then**

6 **RETURN** $VAG(p,(a,m)) \cup VAG(p,(m,b))$

7 **else**

8 **RETURN** $VAG(p,(a,m)) \cup \{[m,m]\} \cup VAG(p,(m,b))$

9 **end**

Remarks

- Compared to VCA the above algorithm is extremely simple; by contrast, VAG uses the time consuming **"a_b roots test"** and that makes it much slower than VCA.[19]
- As Alesina and Galuzzi point out,[4] p.189, there is a variant of this algorithm due to Donato Saeli. Saeli suggested that the *mediant* of the endpoints be used instead of their midpoint $\frac{a+b}{2}$. However, it has been shown[19] that using the mediant of the endpoints is in general much slower than the "mid-point" version.

Example of VAG(p,(a,b))

Given the polynomial $p(x) = x^3 - 7x + 7$ and considering as an upper bound[12][13] on the values of the positive roots $ub = 4$ the arguments of VAG are: $p(x) = x^3 - 7x + 7$ and $(a, b) = (0, 4)$.

$VAG(x^3 - 7x + 7, (0, 4))$

---

1 *var* ← 2 // the number of sign variations in the sequence of coefficients of $(1+x)^3 p(\frac{4x}{1+x}) = 43x^3 - 35x^2 - 7x + 7$

4 $m \leftarrow \frac{a+b}{2} = \frac{0+4}{2} = 2$

5 $p(m) = 1$

8 **RETURN** $VAG(x^3 - 7x + 7, (0, 2)) \cup VAG(x^3 - 7x + 7, (2, 4)$

List of isolation intervals: $\{\}$. List of intervals to be processed: $\{(0,2),(2,4)\}$. Remove the first and process it.

$VAG(x^3-7x+7,(0,2))$

1 *var* $\leftarrow$ 2 // the number of sign variations in the sequence of coefficients of $(1+x)^3p(\frac{2x}{1+x})=x^3-7x^2+7x+7$

4 $m \leftarrow \frac{a+b}{2}=\frac{0+2}{2}=1$

5 $p(m)=1$

8 **RETURN** $VAG(x^3-7x+7,(0,1)) \cup VAG(x^3-7x+7,(1,2)$

List of isolation intervals: $\{\}$. List of intervals to be processed: $\{(0,1),(1,2),(2,4)\}$. Remove the first and process it.

$VAG(x^3-7x+7,(0,1))$

1 *var* $\leftarrow$ 0 // the number of sign variations in the sequence of coefficients of $(1+x)^3p(\frac{x}{1+x})=x^3+7x^2+14x+7$

2 **RETURN** $\emptyset$

List of isolation intervals: $\{\}$. List of intervals to be processed: $\{(1,2),(2,4)\}$. Remove the first and process it.

$VAG(x^3-7x+7,(1,2))$

1 *var* $\leftarrow$ 2 // the number of sign variations in the sequence of coefficients of $(1+x)^3p(\frac{1+2x}{1+x})=x^3-2x^2-x+1$

4 $m \leftarrow \frac{a+b}{2}=\frac{1+2}{2}=\frac{3}{2}$

5 $p(m)=-\frac{1}{8}$

8 **RETURN** $VAG(x^3-7x+7,(1,\frac{3}{2})) \cup VAG(x^3-7x+7,(\frac{3}{2},2)$

List of isolation intervals: $\{\}$. List of intervals to be processed: $\{(1,\frac{3}{2}),(\frac{3}{2},2),(2,4)\}$. Remove the first and process it.

$VAG(x^3-7x+7,(1,\frac{3}{2}))$

1 *var* $\leftarrow$ 1 // the number of sign variations in the sequence of coefficients of $2^3(1+x)^3p(\frac{1+\frac{3}{2}x}{1+x})=x^3+2x^2-8x-8$

3 **RETURN** $\{(1,\frac{3}{2})\}$

List of isolation intervals: $\{(1, \frac{3}{2})\}$. List of intervals to be processed: $\{(\frac{3}{2}, 2), (2, 4)\}$. Remove the first and process it.

$VAG(x^3 - 7x + 7, (\frac{3}{2}, 2))$

1 *var* $\leftarrow$ 1 // the number of sign variations in the sequence of coefficients of $2^3(1+x)^3 p(\frac{\frac{3}{2} + 2x}{1+x}) = 8x^3 + 4x^2 - 4x - 1$

3 **RETURN** $\{(\frac{3}{2}, 2)\}$

List of isolation intervals: $\{(1, \frac{3}{2}), (\frac{3}{2}, 2)\}$. List of intervals to be processed: $\{(2, 4)\}$. Remove the first and process it.

$VAG(x^3 - 7x + 7, (2, 4))$

1 *var* $\leftarrow$ 0 // the number of sign variations in the sequence of coefficients of $(1+x)^3 p(\frac{2 + 4x}{1+x}) = 344x^3 + 376x^2 + 104x + 8$

2 **RETURN** $\emptyset$

List of isolation intervals: $\{(1, \frac{3}{2}), (\frac{3}{2}, 2)\}$. List of intervals to be processed: $\emptyset$. Finished.

Therefore, the two positive roots of the polynomial $p(x) = x^3 - 7x + 7$lie inside the isolation intervals $(1, \frac{3}{2})$ and $(\frac{3}{2}, 2)$. Each root can be approximated by (for example) bisecting the isolation interval — inside which it lies — until the difference of the endpoints is smaller than 10^{-6}; following this approach, the roots turn out to be $\rho_1 = 1.3569$and $\rho_2 = 1.69202$.

References

[1] Vincent, Alexandre Joseph Hidulph (1834). "Mémoire sur la résolution des équations numériques" (http://gallica.bnf.fr/ark:/12148/bpt6k57787134/f4.image.r=Agence Rol.langEN). *Mémoires de la Société Royale des Sciences, de l' Agriculture et des Arts, de Lille*: 1–34. .

[2] Vincent, Alexandre Joseph Hidulph (1836). "Sur la résolution des équations numériques" (http://www-mathdoc.ujf-grenoble.fr/JMPA/PDF/JMPA_1836_1_1_A28_0.pdf). *Journal de Mathématiques Pures et Appliquées* **1**: 341–372. .

[3] Alesina, Alberto; Massimo Galuzzi (1998). "A new proof of Vincent's theorem" (http://retro.seals.ch/cntmng?type=pdf&rid=ensmat-001:1998:44::149&subp=hires). *L'Enseignement Mathématique* **44** (3-4): 219–256. .

[4] Alesina, Alberto; Massimo Galuzzi (2000). "Vincent's Theorem from a Modern Point of View" (http://inf-server.inf.uth.gr/~akritas/Alessina_Galuzzi_b.pdf). *Categorical Studies in Italy 2000, Rendiconti del Circolo Matematico di Palermo, Serie II, n. 64*: 179–191. .

[5] Vincent, Alexandre Joseph Hidulph (1838). "Addition à une précédente note relative à la résolution des équations numériques" (http://math-doc.ujf-grenoble.fr/JMPA/PDF/JMPA_1838_1_3_A19_0.pdf). *Journal de Mathématiques Pures et Appliquées* **3**: 235-243. .

[6] Ostrowski, A. M. (1950). "Note on Vincent's theorem" (http://www.jstor.org/stable/10.2307/1969443). *Ann. Math. Second Series* **52** (3): 702-707. .

[7] Obreschkoff, Nikola (1963). *Verteilung und Berechnung der Nullstellen reeller Polynome*. Berlin: VEB Deutscher Verlag der Wissenschaften.

[8] Uspensky, James Victor (1948). *Theory of Equations* (http://www.google.com/search?q=uspensky+theory+of+equations&btnG=Search+Books&tbm=bks&tbo=1). New York: McGraw–Hill Book Company. .

[9] Akritas, Alkiviadis G.; A.W. Strzeboński, P.S. Vigklas (2008). "Improving the performance of the continued fractions method using new bounds of positive roots" (http://www.lana.lt/journal/30/Akritas.pdf). *Nonlinear Analysis: Modelling and Control* **13**: 265–279. .

[10] Serret, Joseph A. (1877). *Cours d'algèbre supérieure. Tome I* (http://archive.org/details/coursdalgbresu01serruoft). Gauthier-Villars. .

[11] Collins, George E.; Alkiviadis G. Akritas (1976). *Polynomial Real Root Isolation Using Descartes' Rule of Signs* (http://doi.acm.org/10.1145/800205.806346). SYMSAC '76, Proceedings of the third ACM symposium on Symbolic and algebraic computation. Yorktown Heights, NY, USA: ACM. pp. 272-275. .

[12] Vigklas, Panagiotis, S. (2010). *Upper bounds on the values of the positive roots of polynomials* (http://www.inf.uth.gr/images/PHDTheses/phd_thesis_vigklas.pdf). Ph. D. Thesis, University of Thessaly, Greece. .

[13] Akritas, Alkiviadis, G. (2009). "Linear and Quadratic Complexity Bounds on the Values of the Positive Roots of Polynomials" (http://www.jucs.org/jucs_15_3/linear_and_quadratic_complexity). *Journal of Universal Computer Science* **15** (3): 523-537. .

[14] Boulier, François (2010). *Systèmes polynomiaux : que signifie « résoudre » ?* (http://www.lifl.fr/~boulier/polycopies/resoudre.pdf). Université Lille 1. .

[15] Akritas, Alkiviadis G.; Adam W. Strzeboński (2005). "A Comparative Study of Two Real Root Isolation Methods" (http://www.lana.lt/journal/19/Akritas.pdf). *Nonlinear Analysis: Modelling and Control* **10** (4): 297-304. .

[16] Rouillier, F.; P. Zimmerman (2004). "Efficient isolation of polynomial's real roots" (http://dl.acm.org/citation.cfm?id=972166). *Journal of Computational and Applied Mathematics* **162**: 33-50. .

[17] Tsigaridas, P.E.; I.Z. Emiris, (2006). "Univariate polynomial real root isolation: Continued fractions revisited" (http://www.springerlink.com/content/c70468755x403481/). *LNCS* **4168**: 817-828. .

[18] Sharma, Vikram (2007). *Complexity Analysis of Algorithms in Algebraic Computation* (http://www.cs.nyu.edu/web/Research/Theses/sharma_vikram.pdf). Ph.D. Thesis, Courant Institute of Mathematical Sciences, New York University,USA. .

[19] Akritas, Alkiviadis G.; Adam W. Strzeboński, Panagiotis S. Vigklas (2008). "On the Various Bisection Methods Derived from Vincent's Theorem" (http://sci-gems.math.bas.bg:8080/jspui/handle/10525/376). *Serdica Journal of Computing* **2** (1): 89–104. .

[20] Fourier, Jean Baptiste Joseph (1820). "Sur l'usage du théorème de Descartes dans la recherche des limites des racines" (http://ia600309.us.archive.org/22/items/bulletindesscien20soci/bulletindesscien20soci.pdf). *Bulletin des Sciences, par la Société Philomatique de Paris*: 156-165. .

[21] Akritas, Alkiviadis G. (1986). *There's no "Uspensky's Method"* (http://dl.acm.org/citation.cfm?id=32457). In: Proceedings of the fifth ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '86, Waterloo, Ontario, Canada), pp. 88–90. .

[22] Akritas, Alkiviadis G. (2008). *There is no "Descartes' method"* (http://books.google.com/books?id=SJR2ybQdZFgC&lpg=PR1&pg=PR1#v=onepage&q&f=false). In: M.J.Wester and M. Beaudin (Eds), Computer Algebra in Education, AullonaPress, USA, pp. 19-35. .

External links

- Encyclopedia of Mathematics http://www.encyclopediaofmath.org/index.php
- Kehagias, Spyros: RealRoots, a free App for iPhone, iPod Touch and iPad to compare Sturm's method and VAS http://itunes.apple.com/gr/app/realroots/id483609988?mt=8

Budan's theorem

This page is part of **WikiProject Mathematics**.

Shortcut: WP:WPMER

In mathematics, **Budan's theorem**, named after F. D. Budan, is a little-known theorem equivalent to the one by Fourier. The latter is widely known and is referred to under various names, including Budan's. Both theorems provide an upper bound on the number of the real roots a polynomial has inside an open interval by counting the number of **sign variations** or **sign changes** in sequences of real numbers.

Sign variation

Let $c_0, c_1, c_2, \cdots$ be a finite or infinite sequence of real numbers. Suppose $l < r$ and the following conditions hold:

1. If $r = l + 1$ the numbers c_l and c_r have opposite signs.
2. If $r \geq l + 2$ the numbers $c_{l+1}, \ldots, c_{r-1}$ are all zero and the numbers c_l and c_r have opposite signs.

This is called a *sign variation* or *sign change* between the numbers c_l and c_r.

Budan's theorem

The following statement of Budan's theorem had disappeared from the literature (for about 150 years) due to its equivalence to the statement of Fourier's theorem.

Statement of Budan's theorem

Given an equation in x, $p(x) = 0$ of degree $n > 0$ it is possible to make two substitutions $x \leftarrow x + l$ and $x \leftarrow x + r$ where l and r are real numbers so that $l < r$. If v_l and v_r are the sign variations in the sequences of the coefficients of $p(x + l)$ and $p(x + r)$ respectively then, provided $p(r) \neq 0$, the following applies:

1. The polynomial $p(x + l)$ cannot have fewer sign variations than those of $p(x + r)$. In short $v_l \geq v_r$
2. The number ρ of the real roots of the equation $p(x) = 0$ located in the open interval (l, r) can never be more than the number of sign variations lost in passing from the transformed polynomial $p(x + l)$ to the transformed polynomial $p(x + r)$. In short, $\rho \leq v_l - v_r$
3. When the number ρ of the real roots of the equation $p(x) = 0$ located in the open interval (l, r) is strictly less than the number of the sign variations lost in passing from the transformed polynomial $p(x + l)$ to the transformed polynomial $p(x + r)$ then the difference is always an even number. In short, $\rho = v_l - v_r - 2\lambda$ where $\lambda \in \mathbb{Z}_+$.

It should also be mentioned that making use of the substitutions $x \leftarrow x + l$ and $x \leftarrow x + r$, the exact number of real roots in the interval (l, r) can be found in only two cases:

1. If there is no sign variation loss, then there are no real roots in the interval (l, r).
2. If there is one sign variation loss, then there is exactly one real root in the interval (l, r). The inverse statement does not hold in this case.

Examples of Budan's theorem

1. Given the polynomial $p(x) = x^3 - 7x + 7$and the open interval $(0, 2)$, the substitutions $x \leftarrow x + 0$and $x \leftarrow x + 2$can be made. The resulting polynomials and the respective sign variations are:

$$p(x+0) = (x+0)^3 - 7(x+0) + 7 \Rightarrow p(x+0) = x^3 - 7x + 7, v_0 = 2$$
$$p(x+2) = (x+2)^3 - 7(x+2) + 7 \Rightarrow p(x+2) = x^3 + 6x + 5x + 1, v_2 = 0$$

Thus, from Budan's theorem $\rho \leq v_0 - v_2 = 2$. Therefore, the polynomial $p(x)$has either two or no real roots in the open interval $(0, 2)$, a case that must be further investigated.

2. Given the same polynomial $p(x) = x^3 - 7x + 7$and the open interval $(0, 1)$the substitutions $x \leftarrow x + 0$ and $x \leftarrow x + 1$can be made. The resulting polynomials and the respective sign variations are:

$$p(x+0) = (x+0)^3 - 7(x+0) + 7 \Rightarrow p(x+0) = x^3 - 7x + 7, v_0 = 2$$
$$p(x+1) = (x+1)^3 - 7(x+1) + 7 \Rightarrow p(x+1) = x^3 + 3x - 4x + 1, v_2 = 2$$

By Budan's theorem $\rho = v_0 - v_2 = 0$, i.e. there are no real roots in the open interval $(0, 1)$.

The last example indicates the main use of Budan's theorem as a "no roots test".

Fourier's theorem

The statement of **Fourier's theorem (for Polynomials)** which also appears as **Fourier–Budan theorem** or as the **Budan–Fourier theorem** or just as **Budan's theorem** can be found in almost all texts and articles on the subject.

Fourier's sequence

Given an equation in x, $p(x) = 0$of degree $n > 0$, the **Fourier sequence** of $p(x)$, $F_{\text{seq}}(x)$, is defined as the sequence of the $n + 1$functions $p(x), p^{(1)}(x), \ldots, p^{(n)}(x)$where $p^{(i)}$is the ith derivative of $p(x)$.Thus, $F_{\text{seq}}(x) = \left\{p(x), p^{(1)}(x), \ldots, p^{(n)}(x)\right\}$

Example of Fourier's sequence

The Fourier sequence of the polynomial $p(x) = x^3 - 7x + 7$is $F_{\text{seq}}(x) = \left\{x^3 - 7x + 7, 3x^2 - 7, 6x, 6\right\}$.

Statement of Fourier's theorem

Given the polynomial equation x, $p(x) = 0$of degree $n > 0$with real coefficients and its corresponding Fourier sequence $F_{\text{seq}}(x) = \left\{p(x), p^{(1)}(x), \ldots, p^{(n)}(x)\right\}$, x can be replaced by any two real numbers l, r $(l < r)$. If the two resulting arithmetic sequences are represented by $F_{\text{seq}}(l)$and $F_{\text{seq}}(r)$respectively, and their corresponding sign variations by v_l, v_r, then, provided $p(r) \neq 0$, the following applies:

1. The sequence $F_{\text{seq}}(l)$cannot present fewer sign variations than the sequence $F_{\text{seq}}(r)$. In short, $v_l \geq v_r$
2. The number $\rho$of the real roots of the equation $p(x) = 0$located in the open interval (l, r)can never be more than the number of sign variations lost in passing from the sequence $F_{\text{seq}}(l)$to the sequence $F_{\text{seq}}(r)$. In short, $\rho \leq v_l - v_r$
3. When the number $\rho$of the real roots of the equation $p(x) = 0$located in the open interval $(l, r)$is strictly less than the number of the sign variations lost in passing from the sequence $F_{\text{seq}}(l)$to the sequence $F_{\text{seq}}(r)$ then the difference is always an even number. In short, $\rho = v_l - v_r - 2\lambda$ where $\lambda \in \mathbb{Z}_+$

Example of Fourier's theorem

Given the previously mentioned polynomial $p(x) = x^3 - 7x + 7$ and the open interval $(0, 2)$, the following finite sequences and the corresponding sign variations can be computed:

$$F_{\text{seq}}(0) = \{7, -7, 0, 6\}, v_0 = 2$$
$$F_{\text{seq}}(2) = \{1, 5, 12, 6\}, v_2 = 0$$

Thus, from Fourier's theorem $\rho \leq v_0 - v_2 = 2$. Therefore, the polynomial $p(x)$ has either two or no real roots in the open interval $(0, 2)$, a case which should be further investigated.

Historical background

In the beginning of the 19th century F. D. Budan and J. B. J. Fourier presented two different (but equivalent) theorems which enable us to determine the maximum possible number of real roots that an equation has within a given interval.

Budan's statement can hardly be found in the bibliography. Instead, it was replaced by Fourier's statement which is usually referred to as Fourier's theorem, or as Fourier–Budan or as Budan–Fourier or even as Budan's theorem. The actual statement of Budan's theorem appeared in 1807 in the memoir "Nouvelle méthode pour la résolution des équations numériques",[1] whereas Fourier's theorem was first published in 1820 in the "Bulletin des Sciences, par la Société Philomatique de Paris".[2] Due to the importance of these two theorems, there was a great controversy regarding priority rights; on this see [3][4] and especially Arago's book[5] p. 383.

Early applications of Budan's theorem

In "Nouvelle méthode pour la résolution des équations numériques",[1] Budan himself used his theorem to compute the roots of any polynomial equation by calculating the decimal digits of the roots. More precisely, Budan used his theorem as a "**no roots test**", which can be stated as follows: if the polynomials $p(x + a)$ and $p(x + a + 1)$ have (in the sequence of their coefficients) the same number of sign variations then it follows (from Budan's theorem) that $p(x)$ has no real roots in the interval $(a, a + 1)$.

Furthermore, in his book,[1] p. 37, Budan presents, independently of his theorem, a "**0_1 roots test**", that is a criterion for determining whether a polynomial has any roots in the interval (0,1). This test can be stated as follows:

Perform on $p(x)$ the substitution $x \longleftarrow \frac{1}{x+1}$ and count the number of sign variations in the sequence of coefficients of the transformed polynomial; this number gives an *upper bound* on the number of real roots $p(x)$ has inside the open interval $(0, 1)$. More precisely, the number $\rho_{01}(p)$ of real roots in the open interval $(0, 1)$ — multiplicities counted — of the polynomial $p(x) \in \mathbb{R}[x]$, of degree $deg(p)$, is bounded above by the number of sign variations $var_{01}(p)$, where

$$var_{01}(p) = var((x+1)^{deg(p)} p(\frac{1}{x+1}))$$

and $var_{01}(p) \geq \rho_{01}(p)$. As in the case of Descartes' rule of signs if $var_{01}(p) = 0$ it follows that $\rho_{01}(p) = 0$ and if $var_{01}(p) = 1$ it follows that $\rho_{01}(p) = 1$.

This test (which is a special case of the more general Alesina-Galuzzi "**a_b roots test**") was subsequently used by Uspensky in the 20th century. [6] (Uspensky,[7] pp. 298–303, was the one who kept Vincent's theorem alive carrying the torch (so to speak) from Serret.[8])

Bourdon,[9] in the last chapter of his 1831 Algebra (6th edition), pp. 717–760, combined Budan's theorem and Lagrange's continued fraction method for approximating real roots of polynomials and, thus, gave a preview of Vincent's method, without actually giving credit to him. As Vincent mentions in the very first sentence of his 1834[10] and 1836[11] papers, Bourdon used (in his book) a joint presentation of theirs.

Disappearance of Budan's theorem

Budan's theorem forms the basis for Vincent's theorem and Vincent's (exponential) method for the isolation of the real roots of polynomials. Therefore, there is no wonder that Vincent in both of his papers of 1834[10] and 1836[11] states Budan's theorem and contrasts it with the one by Fourier. Vincent was the last author in the 19th century to state Budan's theorem in its original form.

Despite the fact that Budan's theorem was of such great importance, the appearance of Sturm's theorem in 1827 gave it (and Vincent's theorem) the death blow. Sturm's theorem solved the real root isolation problem, by defining the precise number of real roots a polynomial has in a real open interval (a, b); moreover, Sturm himself [12], p. 108, acknowledges the great influence Fourier's theorem had on him: « C'est en m'appuyant sur les principes qu'il a posés, et en imitant ses démonstrations, que j'ai trouvé les nouveaux théorèmes que je vais énoncer. » which translates to «It is by relying upon the principles he has laid out and by imitating his proofs that I have found the new theorems which I am about to announce.» . Because of the above, the theorems by Fourier and Sturm appear in almost all the books on the theory of equations and Sturm's method for computing the real roots of polynomials has been the only one widely known and used ever since – up to about 1980, when it was replaced (in almost all computer algebra systems) by methods derived from Vincent's theorem, the fastest one being the Vincent–Akritas–Strzeboński (VAS) method.[13]

Consequently Budan's theorem (but not his name) was pushed into oblivion. Referentially, in Serret's book[8] there is section 121 (p. 266) on Budan's theorem but the statement is the one due to Fourier, because, as the author explains in the footnote of p. 267 , the two theorems are equivalent and Budan had clear priority. To his credit, Serret included in his Algebra[8], pp 363–368, Vincent's theorem along with its proof and directed all interested readers to Vincent's papers for examples on how it is used. Serret was the last author to mention Vincent's theorem in the 19th century.

Comeback of Budan's theorem

Budan's theorem reappeared, after almost 150 years, in Akritas' Ph.D. Thesis "Vincent's Theorem in Algebraic Manipulation", North Carolina State University, USA, 1978, and in several publications that resulted from that dissertation.[3][4] Akritas found the statement of Budan's theorem in Vincent's paper of 1836[11], which was made available to him through the efforts of a librarian in the Library of the University of Wisconsin–Madison, USA.

Equivalence between the theorems by Budan and Fourier

Budan's theorem is equivalent to the one by Fourier. This equivalence is obvious from the fact that, given the polynomial $p(x)$ of degree $n > 0$, the $n + 1$ terms of the Fourier sequence $F_{\text{seq}}(a)$ (obtained by substituting $x \leftarrow a$ in $F_{\text{seq}}(x)$) have the same signs with (and are proportional to) the corresponding coefficients of the polynomial $p(x + a) = \sum_{i=1}^{n} \frac{p^{(i)}(a)}{i!} x^i$, obtained from Taylor's expansion theorem.

As Alesina and Galuzzi point out in Footnote 9, p. 222 of their paper,[14] the controversy over priority rights of Budan or Fourier is rather pointless from a modern point of view. The two authors think that Budan has an "amazingly modern understanding of the relevance of reducing the algorithm (his own word) to translate a polynomial by $x \leftarrow x + p$, where p is an integer, to simple additions".

Despite their equivalence, the two theorems are quite distinct concerning the impact they had on the isolation of the real roots of polynomials with rational coefficients. To wit:

- Fourier's theorem led Sturm to his theorems and method,[12] whereas
- Budan's theorem is the basis of the Vincent–Akritas–Strzeboński (VAS) method.[13]

Xcas is a computer algebra system where Sturm's method and VAS are both implemented and can be compared; to do so use the functions ***realroot(poly)*** *and* ***time(realroot(poly))****. By default, to isolate the real roots of poly* ***realroot***

uses the VAS method; to use Sturm's method write ***realroot(sturm, poly)****. See also the External links for an iPhone/iPod/iPad application that does the same thing.*

The most significant application of Budan's theorem

Vincent's (exponential) method for the isolation of the real roots of polynomials (which is based on Vincent's theorem of 1834 and 1836)[10][11] is the most significant application of Budan's theorem. Moreover, it is the most representative example of the importance of the statement of Budan's theorem. As explained below, knowing the statement of Fourier's theorem did not help Uspensky realize that there are no roots of $p(x)$ in the open interval $(a, a+1)$ if $p(x+a)$ and $p(x+a+1)$ have the same number of sign variations in the sequence of their coefficients (see [7], pp. 127–137).

Vincent's theorem (1834 and 1836)

If in a polynomial equation with rational coefficients and without multiple roots, one makes successive transformations of the form

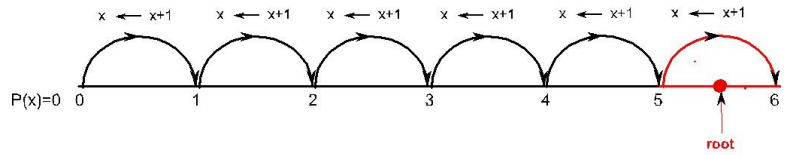
$$x = a + \frac{1}{x'}, \quad x' = b + \frac{1}{x''}, \quad x'' = c + \frac{1}{x'''}, \ldots$$

where *a*, *b*, and *c* are any positive numbers greater than or equal to one, then the resulting transformed equation either has zero sign variations or it has a single sign variation. In the first case there is no root, whereas in the second case the equation has a single positive real root represented by the continued fraction:[10][11][15]

$$a + \cfrac{1}{b + \cfrac{1}{c + \cfrac{1}{\ddots}}}$$

Vincent's implementation of his own theorem

Vincent uses Budan's theorem exclusively as a "no roots test" to locate where the roots lie on the *x*-axis (to compute the quantities $a, b, c, \ldots$ of his theorem); that is, to find the integer part of a root Vincent performs successively substitutions of the form $x \leftarrow x+1$ and stops only when the polynomials $p(x)$ and $p(x+1)$ differ in the number of sign variations in the sequence of their coefficients (i.e when the number of sign variations of $p(x+1)$ is decreased).[10][11]

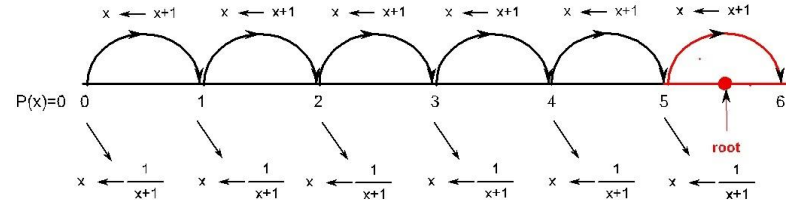


Vincent's search for a root (applying Budan's theorem)

See the corresponding diagram where the root lies in the interval $(5, 6)$. Since in general the location of the root is not known in advance, the root can be found (with the help of Budan's theorem) only by this decrease in the number of sign variations; that is, the polynomial $p(x+6)$ has fewer sign variations than the polynomial $p(x+5)$. Vincent then easily obtains a first continued fraction approximation to this root as $x = 5 + \frac{1}{x'}$ as stated in his theorem. Vincent performs those, and only those, transformations that are described in his theorem.

Uspensky's implementation of Vincent's theorem

According to Alexei Uteshev[16] of St. Petersburg University, Russia, Uspensky came upon the statement (and proof) of Vincent's theorem in the 20th century in Serret's Algebra,[8] pp 363–368, which means that he was not aware of the statement of Budan's theorem (because Serret included in his book Fourier's theorem). Moreover, this means that Uspensky never saw Vincent's papers of 1834[10] and 1836,[11] where Budan's theorem is stated and Vincent's method is explained with several examples (because Serret directed all interested readers to Vincent's papers for examples on how the theorem is used). Therefore, in the preface of his book that came out in 1949,[7] Uspensky erroneously claimed that, based on Vincent's theorem, he had discovered a method for isolating the real roots "much superior in practice to that based on Sturm's Theorem". Uspensky's statement is erroneous because, since he is not using Budan's theorem, he is isolating the real roots doing twice the amount of work done by Vincent (see [7], pp. 127–137). Despite this misunderstanding, kudos to Uspensky for keeping Vincent's theorem alive.



Uspensky's search for a root (not applying Budan's theorem)

Uspensky does not know Budan's theorem and, hence, he cannot use it as a "no roots test". So, for him it does not suffice that $p(x+1)$ has the same number of sign variations as $p(x)$ in order to conclude that $p(x)$ has no roots inside $(0,1)$; to make sure, he also performs the redundant substitution (Budan's "0_1 roots test") $x \leftarrow \frac{1}{1+x}$ in $p(x)$, which unfailingly results in a polynomial with no sign variations and hence no positive roots. Uspensky uses the information obtained from both the needed transformation $x \leftarrow x+1$ and the not needed one $x \leftarrow \frac{1}{1+x}$ to realize that $p(x)$ has no roots in the interval $(0,1)$. In other words, searching for a root, Uspensky advances as illustrated in the corresponding figure.

Uspensky's transformations are not the ones described in Vincent's theorem, and consequently, his transformations take twice as much computation time as the ones needed for Vincent's method. [6][16]

References

[1] Budan, François D. (1807). *Nouvelle méthode pour la résolution des équations numériques* (http://books.google.com/books/about/Nouvelle_méthode_pour_la_résolution_de.html?id=VyMOAAAAQAAJ&redir_esc=y). Paris: Courcier. .

[2] Fourier, Jean Baptiste Joseph (1820). "Sur l'usage du théorème de Descartes dans la recherche des limites des racines" (http://ia600309.us.archive.org/22/items/bulletindesscien20soci/bulletindesscien20soci.pdf). *Bulletin des Sciences, par la Société Philomatique de Paris*: 156–165. .

[3] Akritas, Alkiviadis G. (1981). "On the Budan–Fourier Controversy" (http://dl.acm.org/citation.cfm?id=1089243). *ACM-SIGSAM Bulletin* **15** (1): 8–10. .

[4] Akritas, Alkiviadis G. (1982). "Reflections on a Pair of Theorems by Budan and Fourier" (http://www.jstor.org/stable/2690097). *Mathematics Magazine* **55** (5): 292–298. .

[5] Arago, François (1859), *Biographies of distinguished scientific men* (http://books.google.com/books/about/Biographies_of_distinguished_scientific_men_de.html?id=xGgSAAAAIAAJ&redir_esc=y), Boston: Ticknor and Fields (English Translation),

[6] Akritas, Alkiviadis G. (1986). *There is no "Uspensky's Method"* (http://dl.acm.org/citation.cfm?id=32457). In: Proceedings of the fifth ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '86, Waterloo, Ontario, Canada), pp. 88–90. .

[7] Uspensky, James Victor (1948). *Theory of Equations* (http://www.google.com/search?q=uspensky+theory+of+equations&btnG=Search+Books&tbm=bks&tbo=1). New York: McGraw–Hill Book Company. .

[8] Serret, Joseph A. (1877). *Cours d'algèbre supérieure. Tome I* (http://archive.org/details/coursdalgbresu01serruoft). Gauthier-Villars. .

[9] Bourdon, Louis Pierre Marie (1831). *Éléments d'Algèbre* (http://archive.org/details/elementsalgebra00bourgoog). Paris: Bachelier Père et Fils (6th edition). .

[10] Vincent, Alexandre Joseph Hidulph (1834). "Mémoire sur la résolution des équations numériques" (http://gallica.bnf.fr/ark:/12148/bpt6k57787134/f4.image.r=Agence Rol.langEN). *Mémoires de la Société Royale des Sciences, de l' Agriculture et des Arts, de Lille*: 1–34. .

[11] Vincent, Alexandre Joseph Hidulph (1836). "Sur la résolution des équations numériques" (http://www-mathdoc.ujf-grenoble.fr/JMPA/PDF/JMPA_1836_1_1_A28_0.pdf). *Journal de Mathématiques Pures et Appliquées* **1**: 341–372. .

[12] Hourya, Benis-Sinaceur (1988). "Deux moments dans l'histoire du Théorème d'algèbre de Ch. F. Sturm" (http://www.persee.fr/web/revues/home/prescript/article/rhs_0151-4105_1988_num_41_2_4093). *Revue d'histoire des sciences* **41** (2): 99–132. .

[13] Akritas, Alkiviadis G.; A.W. Strzeboński, P.S. Vigklas (2008). "Improving the performance of the continued fractions method using new bounds of positive roots" (http://www.lana.lt/journal/30/Akritas.pdf). *Nonlinear Analysis: Modelling and Control* **13**: 265–279. .

[14] Alesina, Alberto; Massimo Galuzzi (1998). "A new proof of Vincent's theorem" (http://retro.seals.ch/cntmng?type=pdf&rid=ensmat-001:1998:44::149&subp=hires). *L'Enseignement Mathématique* **44** (3-4): 219–256. .

[15] Vincent, Alexandre Joseph Hidulph (1838). "Addition à une précédente note relative à la résolution des équations numériques" (http://math-doc.ujf-grenoble.fr/JMPA/PDF/JMPA_1838_1_3_A19_0.pdf). *Journal de Mathématiques Pures et Appliquées* **3**: 235-243. .

[16] Akritas, Alkiviadis G. (2008). *There is no "Descartes' method"* (http://books.google.com/books?id=SJR2ybQdZFgC&lpg=PR1&pg=PR1#v=onepage&q&f=false). In: M.J.Wester and M. Beaudin (Eds), Computer Algebra in Education, AullonaPress, USA, pp. 19–35. .

External links

- Budan, F.: extended Biography http://www-history.mcs.st-andrews.ac.uk/Biographies/Budan_de_Boislaurent.html
- Encyclopedia of Mathematics http://www.encyclopediaofmath.org/index.php/Budan-Fourier_theorem
- Kehagias, Spyros: RealRoots, a free App for iPhone, iPod Touch and iPad to compare Sturm's method and VAS http://itunes.apple.com/gr/app/realroots/id483609988?mt=8

Jenkins–Traub algorithm

The **Jenkins–Traub algorithm for polynomial zeros** is a fast globally convergent iterative method published in 1970 by Michael A. Jenkins and Joseph F. Traub. It is "practically a standard in black-box polynomial root-finders".[1]

Given a polynomial P,

$$P(z) = \sum_{i=0}^{n} a_i z^{n-i}, \quad a_0 = 1, \quad a_n \neq 0$$

with complex coefficients compute approximations to the n zeros $\alpha_1, \alpha_2, \ldots, \alpha_n$ of $P(z)$. There is a variation of the Jenkins–Traub algorithm which is faster if the coefficients are real. The Jenkins–Traub algorithm has stimulated considerable research on theory and software for methods of this type.

Overview

The Jenkins–Traub algorithm calculates all of the roots of a polynomial with complex coefficients. The algorithm starts by checking the polynomial for the occurrence of very large or very small roots. If necessary, the coefficients are rescaled by a rescaling of the variable. In the algorithm proper, roots are found one by one and generally in increasing size. After each root is found, the polynomial is deflated by dividing off the corresponding linear factor. Indeed, the factorization of the polynomial into the linear factor and the remaining deflated polynomial is already a result of the root-finding procedure. The root-finding procedure has three stages that correspond to different variants of the inverse power iteration. See Jenkins and Traub.[2] A description can also be found in Ralston and Rabinowitz[3] p. 383. The algorithm is similar in spirit to the two-stage algorithm studied by Traub.[4]

Root-finding procedure

Starting with the current polynomial *P*(*X*) of degree *n*, the smallest root of *P(x)* is computed. To that end, a sequence of so called *H* polynomials is constructed. These polynomials are all of degree *n* − 1 and are supposed to converge to the factor of *P*(*X*) containing all the remaining roots. The sequence of *H* polynomials occurs in two variants, an unnormalized variant that allows easy theoretical insights and a normalized variant of $\bar{H}$ polynomials that keeps the coefficients in a numerically sensible range.

The construction of the *H* polynomials $\left(H^{(\lambda)}(z)\right)_{\lambda=0,1,2,\ldots}$ depends on a sequence of complex numbers $(s_\lambda)_{\lambda=0,1,2,\ldots}$ called shifts. These shifts themselves depend, at least in the third stage, on the previous *H* polynomials. The *H* polynomials are defined as the solution to the implicit recursion

$$H^{(0)}(z) = P'(z) \text{ and } (X - s_\lambda) \cdot H^{(\lambda+1)}(X) \equiv H^{(\lambda)}(X) \pmod{P(X)} .$$

A direct solution to this implicit equation is

$$H^{(\lambda+1)}(X) = \frac{1}{X - s_\lambda} \cdot \left(H^{(\lambda)}(X) - \frac{H^{(\lambda)}(s_\lambda)}{P(s_\lambda)} P(X) \right) ,$$

where the polynomial division is exact.

Algorithmically, one would use for instance the Horner scheme or Ruffini rule to evaluate the polynomials at s_λ and obtain the quotients at the same time. With the resulting quotients *p*(*X*) and *h*(*X*) as intermediate results the next *H* polynomial is obtained as

$$\left.\begin{aligned} P(X) &= p(X) \cdot (X - s_\lambda) + P(s_\lambda) \\ H^{(\lambda)}(X) &= h(X) \cdot (X - s_\lambda) + H^{(\lambda)}(s_\lambda) \end{aligned}\right\} \implies H^{(\lambda+1)}(z) = h(z) - \frac{H^{(\lambda)}(s_\lambda)}{P(s_\lambda)} p(z).$$

Since the highest degree coefficient is obtained from *P(X)*, the leading coefficient of $H^{(\lambda+1)}(X)$ is $-\frac{H^{(\lambda)}(s_\lambda)}{P(s_\lambda)}$. If this is divided out the normalized *H* polynomial is

$$\begin{aligned} \bar{H}^{(\lambda+1)}(X) &= \frac{1}{X - s_\lambda} \cdot \left(P(X) - \frac{P(s_\lambda)}{H^{(\lambda)}(s_\lambda)} H^{(\lambda)}(X) \right) \\ &= \frac{1}{X - s_\lambda} \cdot \left(P(X) - \frac{P(s_\lambda)}{\bar{H}^{(\lambda)}(s_\lambda)} \bar{H}^{(\lambda)}(X) \right) . \end{aligned}$$

Stage one: no-shift process

For $\lambda = 0, 1, \ldots, M - 1$ set $s_\lambda = 0$. Usually *M=5* is chosen for polynomials of moderate degrees up to *n* = 50. This stage is not necessary from theoretical considerations alone, but is useful in practice. It emphasizes in the *H* polynomials the cofactor (of the linear factor) of the smallest root.

Stage two: fixed-shift process

The shift for this stage is determined as some point close to the smallest root of the polynomial. It is quasi-randomly located on the circle with the inner root radius, which in turn is estimated as the positive solution of the equation

$$R^n + |a_{n-1}| R^{n-1} + \ldots + |a_1| R = |a_0| .$$

Since the left side is a convex function and increases monotonically from zero to infinity, this equation is easy to solve, for instance by Newton's method.

Now choose $s = R \cdot \exp(i \phi_{\text{random}})$ on the circle of this radius. The sequence of polynomials $H^{(\lambda+1)}(z)$, $\lambda = M, M + 1, \ldots, L - 1$, is generated with the fixed shift value $s_\lambda = s$. During this iteration, the current approximation for the root

$$t_\lambda = s - \frac{P(s)}{\bar{H}^{(\lambda)}(s)}$$

is traced. The second stage is finished successfully if the conditions

$$|t_{\lambda+1} - t_\lambda| < \tfrac{1}{2}|t_\lambda| \text{ and } |t_\lambda - t_{\lambda-1}| < \tfrac{1}{2}|t_{\lambda-1}|$$

are simultaneously met. If there was no success after some number of iterations, a different random point on the circle is tried. Typically one uses a number of 9 iterations for polynomials of moderate degree, with a doubling strategy for the case of multiple failures.

Stage three: variable-shift process

The $H^{(\lambda+1)}(X)$ are now generated using the variable shifts $s_\lambda, \quad \lambda = L, L+1, \ldots$ which are generated by

$$s_L = t_L = s - \frac{P(s)}{\bar{H}^{(\lambda)}(s)}$$

being the last root estimate of the second stage and

$$s_{\lambda+1} = s_\lambda - \frac{P(s_\lambda)}{\bar{H}^{(\lambda+1)}(s_\lambda)}, \quad \lambda = L, L+1, \ldots,$$

where $\bar{H}^{(\lambda+1)}(z)$ is the normalized *H* polynomial, that is $H^{(\lambda)}(z)$ divided by its leading coefficient.

If the step size in stage three does not fall fast enough to zero, then stage two is restarted using a different random point. If this does not succeed after a small number of restarts, the number of steps in stage two is doubled.

Convergence

It can be shown that, provided *L* is chosen sufficiently large, s_λ always converges to a root of *P*.

The algorithm converges for any distribution of roots, but may fail to find all roots of the polynomial. Furthermore, the convergence is slightly faster than the quadratic convergence of Newton–Raphson iteration, however, it uses at least twice as many operations per step.

What gives the algorithm its power?

Compare with the Newton–Raphson iteration

$$z_{i+1} = z_i - \frac{P(z_i)}{P'(z_i)}.$$

The iteration uses the given *P* and P'. In contrast the third-stage of Jenkins–Traub

$$s_{\lambda+1} = s_\lambda - \frac{P(s_\lambda)}{\bar{H}^{\lambda+1}(s_\lambda)} = s_\lambda - \frac{W^\lambda(s_\lambda)}{(W^\lambda)'(s_\lambda)}$$

is precisely a Newton–Raphson iteration performed on certain rational functions. More precisely, Newton–Raphson is being performed on a sequence of rational functions

$$W^\lambda(z) = \frac{P(z)}{H^\lambda(z)}.$$

For λ sufficiently large,

$$\frac{P(z)}{\bar{H}^\lambda(z)} = W^\lambda(z)\, LC(H^\lambda)$$

is as close as desired to a first degree polynomial

$$z - \alpha_1,$$

where α_1 is one of the zeros of P. Even though Stage 3 is precisely a Newton–Raphson iteration, differentiation is not performed.

Analysis of the *H* polynomials

Let $\alpha_1, \ldots, \alpha_n$ be the roots of *P*(*X*). The so called Lagrange factors of *P(X)* are the cofactors of these roots,

$$P_m(X) = \frac{P(X) - P(\alpha_m)}{X - \alpha_m}.$$

If all roots are different, then the Lagrange factors form a basis of the space of polynomials of degree at most *n* − 1. By analysis of the recursion procedure one finds that the *H* polynomials have the coordinate representation

$$H^{(\lambda)}(X) = \sum_{m=1}^{n} \left[\prod_{\kappa=0}^{\lambda-1}(\alpha_m - s_\kappa)\right]^{-1} P_m(X)\ .$$

Each Lagrange factor has leading coefficient 1, so that the leading coefficient of the H polynomials is the sum of the coefficients. The normalized H polynomials are thus

$$\bar{H}^{(\lambda)}(X) = \frac{\sum_{m=1}^{n}\left[\prod_{\kappa=0}^{\lambda-1}(\alpha_m - s_\kappa)\right]^{-1} P_m(X)}{\sum_{m=1}^{n}\left[\prod_{\kappa=0}^{\lambda-1}(\alpha_m - s_\kappa)\right]^{-1}} = \frac{P_1(X) + \sum_{m=2}^{n}\left[\prod_{\kappa=0}^{\lambda-1}\frac{\alpha_1 - s_\kappa}{\alpha_m - s_\kappa}\right] P_m(X)}{1 + \sum_{m=1}^{n}\left[\prod_{\kappa=0}^{\lambda-1}\frac{\alpha_1 - s_\kappa}{\alpha_m - s_\kappa}\right]}\ .$$

Convergence orders

If the condition $|\alpha_1 - s_\kappa| < \min_{m=2,3,\ldots,n}|\alpha_m - s_\kappa|$ holds for almost all iterates, the normalized H polynomials will converge at least geometrically towards $P_1(X)$.
Under the condition that

$$|\alpha_1| < |\alpha_2| = \min_{m=2,3,\ldots,n}|\alpha_m|$$

one gets the aymptotic estimates for

- stage 1:

$$H^{(\lambda)}(X) = P_1(X) + O\left(\left|\frac{\alpha_1}{\alpha_2}\right|^{\lambda}\right).$$

- for stage 2, if *s* is close enough to α_1:

$$H^{(\lambda)}(X) = P_1(X) + O\left(\left|\frac{\alpha_1}{\alpha_2}\right|^{M} \cdot \left|\frac{\alpha_1 - s}{\alpha_2 - s}\right|^{\lambda - M}\right)$$

  and

$$s - \frac{P(s)}{\bar{H}^{(\lambda)}(s)} = \alpha_1 + O\left(\ldots \cdot |\alpha_1 - s|\right).$$

- and for stage 3:

$$H^{(\lambda)}(X) = P_1(X) + O\left(\prod_{\kappa=0}^{\lambda-1}\left|\frac{\alpha_1 - s_\kappa}{\alpha_2 - s_\kappa}\right|\right)$$

  and

$$s_{\lambda+1} = s_\lambda - \frac{P(s)}{\bar{H}^{(\lambda+1)}(s_\lambda)} = \alpha_1 + O\left(\prod_{\kappa=0}^{\lambda-1}\left|\frac{\alpha_1 - s_\kappa}{\alpha_2 - s_\kappa}\right| \cdot \frac{|\alpha_1 - s_\lambda|^2}{|\alpha_2 - s_\lambda|}\right)$$

  giving rise to a higher than quadratic convergence order of $\phi^2 = 1 + \phi \approx 2.61$, where $\phi = \frac{1}{2}(1 + \sqrt{5})$ is the golden ratio.

Interpretation as inverse power iteration

All stages of the Jenkins–Traub complex algorithm may be represented as the linear algebra problem of determining the eigenvalues of a special matrix. This matrix is the coordinate representation of a linear map in the *n*-dimensional space of polynomials of degree *n* – 1 or less. The principal idea of this map is to interpret the factorization

$$P(X) = (X - \alpha_1) \cdot P_1(X)$$

with a root $\alpha_1 \in \mathbb{C}$ and $P_1(X) = P(X)/(X - \alpha_1)$ the remaining factor of degree *n* – 1 as the eigenvector equation for the multiplication with the variable *X*, followed by remainder computation with divisor *P*(*X*),

$$M_X(H) = (X \cdot H(X)) \bmod P(X)\,.$$

This maps polynomials of degree at most *n* – 1 to polynomials of degree at most *n* – 1. The eigenvalues of this map are the roots of *P*(*X*), since the eigenvector equation reads

$$0 = (M_X - \alpha \cdot id)(H) = ((X - \alpha) \cdot H) \bmod P\,,$$

which implies that $(X - \alpha) \cdot H) = C \cdot P(X)$, that is, $(X - \alpha)$ is a linear factor of *P*(*X*). In the monomial basis the linear map M_X is represented by a companion matrix of the polynomial *P*, as

$$M_X(H) = \sum_{m=1}^{n-1} (H_{m-1} - P_m H_{n-1}) X^m - P_0 H_{n-1}\,,$$

the resulting coefficient matrix is

$$A = \begin{pmatrix} 0 & 0 & \dots & 0 & -P_0 \\ 1 & 0 & \dots & 0 & -P_1 \\ 0 & 1 & \dots & 0 & -P_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -P_{n-1} \end{pmatrix}.$$

To this matrix the inverse power iteration is applied in the three variants of no shift, constant shift and generalized Rayleigh shift in the three stages of the algorithm. It is more efficient to perform the linear algebra operations in polynomial arithmetic and not by matrix operations, however, the properties of the inverse power iteration remain the same.

Real coefficients

The Jenkins–Traub algorithm described earlier works for polynomials with complex coefficients. The same authors also created a three-stage algorithm for polynomials with real coefficients. See Jenkins and Traub A Three-Stage Algorithm for Real Polynomials Using Quadratic Iteration [5] [6] The algorithm finds either a linear or quadratic factor working completely in real arithmetic. If the complex and real algorithms are applied to the same real polynomial, the real algorithm is about four times as fast. The real algorithm always converges and the rate of convergence is greater than second order.

A connection with the shifted QR algorithm

There is a surprising connection with the shifted QR algorithm for computing matrix eigenvalues. See Dekker and Traub The shifted QR algorithm for Hermitian matrices [7].[8] Again the shifts may be viewed as Newton-Raphson iteration on a sequence of rational functions converging to a first degree polynomial.

Software and testing

The software for the Jenkins–Traub algorithm was published as Jenkins and Traub Algorithm 419: Zeros of a Complex Polynomial [9].[10] The software for the real algorithm was published as Jenkins Algorithm 493: Zeros of a Real Polynomial [11].[12]

The methods have been extensively tested by many people. As predicted they enjoy faster than quadratic convergence for all distributions of zeros.

However there are polynomials which can cause loss of precision as illustrated by the following example. The polynomial has all its zeros lying on two half-circles of different radii. Wilkinson recommends that it is desirable for stable deflation that smaller zeros be computed first. The second-stage shifts are chosen so that the zeros on the smaller half circle are found first. After deflation the polynomial with the zeros on the half circle is known to be ill-conditioned if the degree is large; see Wilkinson,[13] p. 64. The original polynomial was of degree 60 and suffered severe deflation instability.

References

[1] Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2007), Numerical Recipes: The Art of Scientific Computing, 3rd ed., Cambridge University Press, page 470.

[2] Jenkins, M. A. and Traub, J. F. (1970), A Three-Stage Variables-Shift Iteration for Polynomial Zeros and Its Relation to Generalized Rayleigh Iteration (http://www.springerlink.com/content/q6w17w30035r2152/?p=ae17d723839045be82d270b45363625f&pi=1), Numer. Math. 14, 252–263.

[3] Ralston, A. and Rabinowitz, P. (1978), A First Course in Numerical Analysis, 2nd ed., McGraw-Hill, New York.

[4] Traub, J. F. (1966), A Class of Globally Convergent Iteration Functions for the Solution of Polynomial Equations (http://links.jstor.org/sici?sici=0025-5718(196601)20:93<113:ACOGCI>2.0.CO;2-3), Math. Comp., 20(93), 113–138.

[5] http://links.jstor.org/sici?sici=0036-1429%28197012%297%3A4%3C545%3AATAFRP%3E2.0.CO%3B2-J

[6] Jenkins, M. A. and Traub, J. F. (1970), A Three-Stage Algorithm for Real Polynomials Using Quadratic Iteration (http://links.jstor.org/sici?sici=0036-1429(197012)7:4<545:ATAFRP>2.0.CO;2-J), SIAM J. Numer. Anal., 7(4), 545–566.

[7] http://linkinghub.elsevier.com/retrieve/pii/0024379571900358

[8] Dekker, T. J. and Traub, J. F. (1971), The shifted QR algorithm for Hermitian matrices (http://linkinghub.elsevier.com/retrieve/pii/0024379571900358), Lin. Algebra Appl., 4(2), 137–154.

[9] http://portal.acm.org/citation.cfm?id=361262&coll=portal&dl=ACM

[10] Jenkins, M. A. and Traub, J. F. (1972), Algorithm 419: Zeros of a Complex Polynomial (http://portal.acm.org/citation.cfm?id=361262&coll=portal&dl=ACM), Comm. ACM, 15, 97–99.

[11] http://portal.acm.org/citation.cfm?id=355643&coll=ACM&dl=ACM

[12] Jenkins, M. A. (1975), Algorithm 493: Zeros of a Real Polynomial (http://portal.acm.org/citation.cfm?id=355643&coll=ACM&dl=ACM), ACM TOMS, 1, 178–189.

[13] Wilkinson, J. H. (1963), Rounding Errors in Algebraic Processes, Prentice Hall, Englewood Cliffs, N.J.

External links

- Additional Bibliography for the Jenkins–Traub Method (http://math.fullerton.edu/mathews/n2003/jenkinstraub/JenkinsTraubBib/Links/JenkinsTraubBib_lnk_2.html)
- Internet Resources for the Jenkins–Traub Method (http://math.fullerton.edu/mathews/n2003/jenkinstraub/JenkinsTraubBib/Links/JenkinsTraubBib_lnk_1.html)
- A free downloadable Windows application using the Jenkins–Traub Method for polynomials with real and complex coefficients (http://www.hvks.com/Numerical/winsolve.html)
- CPOLY (http://www.netlib.org/toms/419) (Algorithm 419 from Comm. ACM) on Netlib. In Fortran.
- RPOLY (http://www.netlib.org/toms/493) Algorithm 493 from ACM TOMS) on Netlib. In Fortran.
- Online Calculator (http://www.novanumeric.com/samples.php?CalcName=Roots) Online Polynomial Calculator using the Jenkins Traub procedure

Laguerre's method

In numerical analysis, **Laguerre's method** is a root-finding algorithm tailored to polynomials. In other words, Laguerre's method can be used to solve numerically the equation

$$p(x) = 0$$

for a given polynomial *p*. One of the most useful properties of this method is that it is, from extensive empirical study, very close to being a "sure-fire" method, meaning that it is almost guaranteed to always converge to *some* root of the polynomial, no matter what initial guess is chosen. This method is named in honour of Edmond Laguerre, a French mathematician.

Derivation

The fundamental theorem of algebra states that every *n*th degree polynomial *p* can be written in the form

$$p(x) = C(x - x_1)(x - x_2)\cdots(x - x_n),$$

where x_k are the roots of the polynomial. If we take the natural logarithm of both sides, we find that

$$\ln|p(x)| = \ln|C| + \ln|x - x_1| + \ln|x - x_2| + \cdots + \ln|x - x_n|.$$

Denote the derivative by

$$G = \frac{d}{dx}\ln|p(x)| = \frac{1}{x - x_1} + \frac{1}{x - x_2} + \cdots + \frac{1}{x - x_n},$$

and the second derivative by

$$H = -\frac{d^2}{dx^2}\ln|p(x)| = \frac{1}{(x - x_1)^2} + \frac{1}{(x - x_2)^2} + \cdots + \frac{1}{(x - x_n)^2}.$$

We then make what Acton calls a 'drastic set of assumptions', that the root we are looking for, say, $x_1$is a certain distance away from our guess x , and all the other roots are clustered together some distance away. If we denote these distances by

$$a = x - x_1$$

and

$$b = x - x_i, \quad i = 2, 3, \ldots, n$$

then our equation for *G* may be written

$$G = \frac{1}{a} + \frac{n - 1}{b}$$

and that for *H* becomes

$$H = \frac{1}{a^2} + \frac{n-1}{b^2}.$$

Solving these equations, we find that

$$a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}},$$

where the square root of a complex number is chosen to produce larger absolute value of the denominator, or equivalently, to satisfy: $\mathrm{Re}\left(\overline{G}\sqrt{(n-1)(nH-G^2)}\right) > 0$, where Re denotes real part of a complex number, and $\overline{G}$ is a complex conjugation of G ; or

$$a = \frac{p(x)}{p'(x)} \cdot \left(\frac{1}{n} + \frac{n-1}{n} \sqrt{1 - \frac{n}{n-1} \frac{p(x)p''(x)}{p'(x)^2}} \right)^{-1},$$

where the square root of a complex number is chosen to have a non-negative real part. For small values of *p(x)* this formula differs from the offset of the third order Halley's method by an error of $O(p(x)^3)$.

Note that, even if the 'drastic set of assumptions' does not work for some particular polynomial P, P can be transformed into a related polynomial Q for which the assumptions are correct, e.g. by adding a suitable complex number to give distinct roots distinct magnitudes if necessary (which it will be if some roots are complex conjugates), and then repeatedly applying the root squaring transformation used in Graeffe's method enough times to make the smaller roots significantly smaller than the largest root (and so, clustered in comparison); the Graeffe's method approximation can be used to start the new iteration for Laguerre's method. An approximate root for P may then be obtained straightforwardly from that for Q.

Definition

The above derivation leads to the following method:

- Choose an initial guess x_0
- For *k* = 0, 1, 2, …
  - Calculate $G = \frac{p'(x_k)}{p(x_k)}$
  - Calculate $H = G^2 - \frac{p''(x_k)}{p(x_k)}$
  - Calculate $a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}}$, where the sign is chosen to give the denominator with the larger absolute value, to avoid loss of significance as iteration proceeds.
  - Set $x_{k+1} = x_k - a$
- Repeat until *a* is small enough or if the maximum number of iterations has been reached.

Properties

If *x* is a simple root of the polynomial *p*, then Laguerre's method converges cubically whenever the initial guess x_0 is close enough to the root *x*. On the other hand, if *x* is a multiple root then the convergence is only linear. This is obtained with the penalty of calculating values for the polynomial and its first and second derivatives at each stage of the iteration.

A major advantage of Laguerre's method is that it is almost guaranteed to converge to *some* root of the polynomial *no matter where the initial approximation is chosen*. This is in contrast to other methods such as the Newton-Raphson method which may fail to converge for poorly chosen initial guesses. It may even converge to a complex root of the polynomial, because of the square root being taken in the calculation of *a* above may be of a negative number. This may be considered an advantage or a liability depending on the application to which the

method is being used. Empirical evidence has shown that convergence failure is extremely rare, making this a good candidate for a general purpose polynomial root finding algorithm. However, given the fairly limited theoretical understanding of the algorithm, many numerical analysts are hesitant to use it as such, and prefer better understood methods such as the Jenkins-Traub method, for which more solid theory has been developed. Nevertheless, the algorithm is fairly simple to use compared to these other "sure-fire" methods, easy enough to be used by hand or with the aid of a pocket calculator when an automatic computer is unavailable. The speed at which the method converges means that one is only very rarely required to compute more than a few iterations to get high accuracy.

References

- Forman S. Acton, *Numerical Methods that Work*, Harper & Row, 1970, ISBN 0-88385-450-3.
- S. Goedecker, Remark on Algorithms to Find Roots of Polynomials, *SIAM J. Sci. Comput.* [1] **15(5)**, 1059–1063 (September 1994).
- Wankere R. Mekwi (2001). Iterative Methods for Roots of Polynomials [2]. Master's thesis, University of Oxford.
- V. Y. Pan, Solving a Polynomial Equation: Some History and Recent Progress, *SIAM Rev.* [3] **39(2)**, 187–220 (June 1997).
- Anthony Ralston and Philip Rabinowitz, *A First Course in Numerical Analysis*, McGraw-Hill, 1978, ISBN 0-07-051158-6.
- Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP (2007). "Section 9.5.3. Laguerre's Method" [1]. *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8.

References

[1] http://epubs.siam.org/sam-bin/dbq/toclist/SISC

[2] http://eprints.maths.ox.ac.uk/archive/00000016/

[3] http://epubs.siam.org/sam-bin/dbq/toclist/SIREV

Bairstow's method

In numerical analysis, **Bairstow's method** is an efficient algorithm for finding the roots of a real polynomial of arbitrary degree. The algorithm first appeared in the appendix of the 1920 book "Applied Aerodynamics" by Leonard Bairstow. The algorithm finds the roots in complex conjugate pairs using only real arithmetic.

See root-finding algorithm for other algorithms.

Description of the method

Bairstow's approach is to use Newton's method to adjust the coefficients *u* and *v* in the quadratic $x^2 + ux + v$ until its roots are also roots of the polynomial being solved. The roots of the quadratic may then be determined, and the polynomial may be divided by the quadratic to eliminate those roots. This process is then iterated until the polynomial becomes quadratic or linear, and all the roots have been determined.

Long division of the polynomial to be solved

$$P(x) = \sum_{i=0}^{n} a_i x^i$$

by $x^2 + ux + v$ yields a quotient $Q(x) = \sum_{i=0}^{n-2} b_i x^i$ and a remainder $cx + d$ such that

$$P(x) = (x^2 + ux + v)\left(\sum_{i=0}^{n-2} b_i x^i\right) + (cx + d).$$

A second division of $Q(x)$ by $x^2 + ux + v$ is performed to yield a quotient $R(x) = \sum_{i=0}^{n-4} f_i x^i$ and remainder $gx + h$ with

$$Q(x) = (x^2 + ux + v)\left(\sum_{i=0}^{n-4} f_i x^i\right) + (gx + h).$$

The variables c, d, g, h, and the $\{b_i\}$, $\{f_i\}$ are functions of u and v. They can be found recursively as follows.

$$\begin{aligned}
b_n &= b_{n-1} = 0, & f_n &= f_{n-1} = 0, \\
b_i &= a_{i+2} - ub_{i+1} - vb_{i+2} & f_i &= b_{i+2} - uf_{i+1} - vf_{i+2} \qquad (i = n-2, \ldots, 0), \\
c &= a_1 - ub_0 - vb_1, & g &= b_1 - uf_0 - vf_1, \\
d &= a_0 - vb_0, & h &= b_0 - vf_0.
\end{aligned}$$

The quadratic evenly divides the polynomial when

$$c(u, v) = d(u, v) = 0.$$

Values of u and v for which this occurs can be discovered by picking starting values and iterating Newton's method in two dimensions

$$\begin{bmatrix} u \\ v \end{bmatrix} := \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} \frac{\partial c}{\partial u} & \frac{\partial c}{\partial v} \\ \frac{\partial d}{\partial u} & \frac{\partial d}{\partial v} \end{bmatrix}^{-1} \begin{bmatrix} c \\ d \end{bmatrix} := \begin{bmatrix} u \\ v \end{bmatrix} - \frac{1}{vg^2 + h(h - ug)} \begin{bmatrix} -h & g \\ -gv & gu - h \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix}$$

until convergence occurs. This method to find the zeroes of polynomials can thus be easily implemented with a programming language or even a spreadsheet.

Example

The task is to determine a pair of roots of the polynomial

$$f(x) = 6\,x^5 + 11\,x^4 - 33\,x^3 - 33\,x^2 + 11\,x + 6.$$

As first quadratic polynomial one may choose the normalized polynomial formed from the leading three coefficients of *f(x)*,

$$u = \frac{a_{n-1}}{a_n} = \frac{11}{6}; \quad v = \frac{a_{n-2}}{a_n} = -\frac{33}{6}.$$

The iteration then produces the table

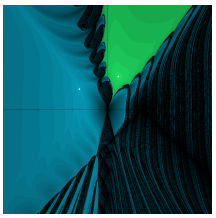
Iteration steps of Bairstow's method

Nr	u	v	step length	roots
0	1.833333333333	-5.500000000000	5.579008780071	-0.916666666667±2.517990821623
1	2.979026068546	-0.039896784438	2.048558558641	-1.489513034273±1.502845921479
2	3.635306053091	1.900693009946	1.799922838287	-1.817653026545±1.184554563945
3	3.064938039761	0.193530875538	1.256481376254	-1.532469019881±1.467968126819
4	3.461834191232	1.385679731101	0.428931413521	-1.730917095616±1.269013105052
5	3.326244386565	0.978742927192	0.022431883898	-1.663122193282±1.336874153612
6	3.333340909351	1.000022701147	0.000023931927	-1.666670454676±1.333329555414
7	3.333333333340	1.000000000020	0.000000000021	-1.666666666670±1.333333333330
8	3.333333333333	1.000000000000	0.000000000000	-1.666666666667±1.333333333333

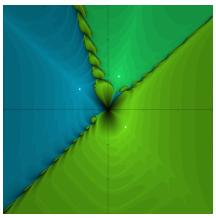
After eight iterations the method produced a quadratic factor that contains the roots -1/3 and -3 within the represented precision. The step length from the fourth iteration on demonstrates the superlinear speed of convergence.

Performance

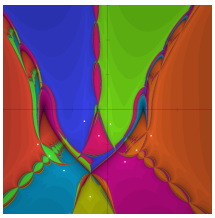
Bairstow's algorithm inherits the local quadratic convergence of Newton's method, except in the case of quadratic factors of multiplicity higher than 1, when convergence to that factor is linear. A particular kind of instability is observed when the polynomial has odd degree and only one real root. Quadratic factors that have a small value at this real root tend to diverge to infinity.



$f(x) = x^5 - 1$



$f(x) = x^6 - x$



$f(x) = 6x^5 + 11x^4 - 33x^3 - 33x^2 + 11x + 6$

The images represent pairs $(s, t) \in [-3, 3]^2$. Points in the upper half plane *t>0* correspond to a linear factor with roots $s \pm it$, that is $x^2 + ux + v = (x - s)^2 + t^2$. Points in the lower half plane *t<0* correspond to quadratic factors with roots $s \pm t$, that is, $x^2 + ux + v = (x - s)^2 - t^2$, so in general $(u,\, v) = (-2s,\, s^2 + t\,|t|)$. Points are colored according to the final point of the Bairstow iteration, black points indicate divergent behavior.

The first image is a demonstration of the single real root case. The second indicates that one can remedy the divergent behavior by introducing an additional real root, at the cost of slowing down the speed of convergence. The third image corresponds to the example above.

External links

- Bairstow's Algorithm on Mathworld [1]
- Numerical Recipes in Fortran 77 Online [2]
- Example polynomial root solver (deg(P)<=10) using Bairstow's Method [3]
- LinBairstowSolve, an open-source C++ implementation of the Lin-Bairstow method available as a method of the VTK library [4]
- Online root finding of a polynomial-Bairstow's method [5] by Farhad Mazlumi

References

[1] http://mathworld.wolfram.com/BairstowsMethod.html
[2] http://library.lanl.gov/numerical/bookfpdf.html
[3] http://www.polarhome.com:793/~amate2/php/polysolver_en.php
[4] http://www.vtk.org/doc/nightly/html/classvtkMath.html
[5] http://catc.ac.ir/mazlumi/jscodes/bairstow.php

Durand–Kerner method

In numerical analysis, the **Durand–Kerner method** established 1960–66 and named after E. Durand and Immo Kerner, also called the **method of Weierstrass**, established 1859–91 and named after Karl Weierstrass, is a root-finding algorithm for solving polynomial equations. In other words, the method can be used to solve numerically the equation

$f(x) = 0$

where f is a given polynomial, which can be taken to be scaled so that the leading coefficient is 1.

Explanation

The explanation is for equations of degree four. It is easily generalized to other degrees.

Let the polynomial f be defined by

$f(x) = x^4 + ax^3 + bx^2 + cx + d$

for all x.

The known numbers a, b, c, d are the coefficients.

Let the (complex) numbers P,Q,R,S be the roots of this polynomial f.

Then

$f(x) = (x - P)(x - Q)(x - R)(x - S)$

for all x. One can isolate the value P from this equation,

$$P = x - \frac{f(x)}{(x - Q)(x - R)(x - S)}.$$

The substitution

$$x := x - \frac{f(x)}{(x - Q)(x - R)(x - S)}$$

is a strongly stable fixed point iteration in that every initial point $x \neq Q,R,S$ delivers after one iteration the root P.

If one replaces the zeros Q, R and S by approximations $q \approx Q$, $r \approx R$, $s \approx S$, such that q,r,s are not equal to P, then P is still a fixed point of the perturbed fixed point iteration since

$$P - \frac{f(P)}{(P-q)(P-r)(P-s)} = P - 0 = P.$$

Note that the denominator is still different from zero. This fixed point iteration is a contraction mapping for x around P.

The clue to the method now is to combine the fixed point iteration for P with similar iterations for Q,R,S into a simultaneous iteration for all roots.

Initialize p, q, r, s:

$p_0 := (0.4 + 0.9\ \mathrm{i})^0$;

$q_0 := (0.4 + 0.9\ \mathrm{i})^1$;

$r_0 := (0.4 + 0.9\ \mathrm{i})^2$;

$s_0 := (0.4 + 0.9\ \mathrm{i})^3$;

There is nothing special about choosing 0.4 + 0.9 i except that it is neither a real number nor a root of unity.

Make the substitutions for $n = 1,2,3,\cdots$

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{(p_{n-1}-q_{n-1})(p_{n-1}-r_{n-1})(p_{n-1}-s_{n-1})};$$

$$q_n = q_{n-1} - \frac{f(q_{n-1})}{(q_{n-1}-p_n)(q_{n-1}-r_{n-1})(q_{n-1}-s_{n-1})};$$

$$r_n = r_{n-1} - \frac{f(r_{n-1})}{(r_{n-1}-p_n)(r_{n-1}-q_n)(r_{n-1}-s_{n-1})};$$

$$s_n = s_{n-1} - \frac{f(s_{n-1})}{(s_{n-1}-p_n)(s_{n-1}-q_n)(s_{n-1}-r_n)}.$$

Re-iterate until the numbers p, q, r, s stop essentially changing. Then they have the values P, Q, R, S in some order and in the chosen precision. So the problem is solved.

Note that you must use complex number arithmetic, and that the roots are found simultaneously rather than one at a time.

Variations

This iteration procedure, like the Gauss–Seidel method for linear equations, computes one number at a time based on the already computed numbers. A variant of this procedure, like the Jacobi method, computes a vector of root approximations at a time. Both variant are effective root-finding algorithms.

One could also choose the initial values for p,q,r,s by some other procedure, even randomly, but in a way that

- they are inside some not too large circle containing also the roots of $f(x)$, e.g. the circle around the origin with radius $1 + \max(|a|, |b|, |c|, |d|)$, (where $1,a,b,c,d$ are the coefficients of $f(x)$) and that
- they are not too close to each other, which may increasingly become a concern as the degree of the polynomial increases.

Example

This example is from the reference 1992. The equation solved is $x^3 - 3x^2 + 3x - 5 = 0$. The first 4 iterations move *p, q, r* seemingly chaotically, but then the roots are located to 1 decimal. After iteration number 5 we have 4 correct decimals, and the subsequent iteration number 6 confirms that the computed roots are fixed. This general behaviour is characteristic for the method.

it.-no.	p	q	r
0	+1.0000+0.0000i	+0.4000+0.9000i	−0.6500+0.7200i
1	+1.3608+2.0222i	−0.3658+2.4838i	−2.3858−0.0284i
2	+2.6597+2.7137i	+0.5977+0.8225i	−0.6320−1.6716i
3	+2.2704+0.3880i	+0.1312+1.3128i	+0.2821−1.5015i
4	+2.5428−0.0153i	+0.2044+1.3716i	+0.2056−1.3721i
5	+2.5874+0.0000i	+0.2063+1.3747i	+0.2063−1.3747i
6	+2.5874+0.0000i	+0.2063+1.3747i	+0.2063−1.3747i

Note that the equation has one real root and one pair of complex conjugate roots, and that the sum of the roots is 3.

Derivation of the method via Newton's method

For every *n*-tuple of complex numbers, there is exactly one monic polynomial of degree *n* that has them as its zeros (keeping multiplicities). This polynomial is given by multiplying all the corresponding linear factors, that is

$$g_{\vec{z}}(X) = (X - z_1)\cdots(X - z_n).$$

This polynomial has coefficients that depend on the prescribed zeros,

$$g_{\vec{z}}(X) = X^n + g_{n-1}(\vec{z})X^{n-1} + \cdots + g_0(\vec{z}).$$

Those coefficients are, up to a sign, the elementary symmetric polynomials $\alpha_1(\vec{z}), \ldots, \alpha_n(\vec{z})$ of degrees *1,...,n*.

To find all the roots of a given polynomial $f(X) = X^n + c_{n-1}X^{n-1} + \cdots + c_0$ with coefficient vector $(c_{n-1}, \ldots, c_0)$ simultaneously is now the same as to find a solution vector to the system

$$\begin{array}{rcccccc} c_0 & = & g_0(\vec{z}) & = & (-1)^n\alpha_n(\vec{z}) & = & (-1)^n z_1\cdots z_n \\ c_1 & = & g_1(\vec{z}) & = & (-1)^{n-1}\alpha_{n-1}(\vec{z}) & & \\ & & \vdots & & & & \\ c_{n-1} & = & g_{n-1}(\vec{z}) & = & -\alpha_1(\vec{z}) & = & -(z_1 + z_2 + \cdots + z_n). \end{array}$$

The Durand–Kerner method is obtained as the multidimensional Newton's method applied to this system. It is algebraically more comfortable to treat those identities of coefficients as the identity of the corresponding polynomials, $g_{\vec{z}}(X) = f(X)$. In the Newton's method one looks, given some initial vector $\vec{z}$, for an increment vector $\vec{w}$ such that $g_{\vec{z}+\vec{w}}(X) = f(X)$ is satisfied up to second and higher order terms in the increment. For this one solves the identity

$$f(X) - g_{\vec{z}}(X) = \sum_{k=1}^{n} \frac{\partial g_{\vec{z}}(X)}{\partial z_k} w_k = -\sum_{k=1}^{n} w_k \prod_{j\neq k}(X - z_j).$$

If the numbers $z_1, \ldots, z_n$ are pairwise different, then the polynomials in the terms of the right hand side form a basis of the *n*-dimensional space $\mathbb{C}[X]_{n-1}$ of polynomials with maximal degree *n* – 1. Thus a solution $\vec{w}$ to the increment equation exists in this case. The coordinates of the increment $\vec{w}$ are simply obtained by evaluating the increment equation

$$-\sum_{k=1}^{n} w_k \prod_{j\neq k}(X - z_j) = f(X) - \prod_{j=1}^{n}(X - z_j)$$

at the points $X = z_k$, which results in

$-w_k \prod_{j \neq k} (z_k - z_j) = -w_k g'_{\vec{z}}(z_k) = f(z_k)$, that is $w_k = -\frac{f(z_k)}{\prod_{j \neq k}(z_k - z_j)}$.

Root inclusion via Gerschgorin's circles

In the quotient ring (algebra) of residue classes modulo f(*X*), the multiplication by *X* defines an endomorphism that has the zeros of f(*X*) as eigenvalues with the corresponding multiplicities. Choosing a basis, the multiplication operator is represented by its coefficient matrix *A*, the companion matrix of f(*X*) for this basis.

Since every polynomial can be reduced modulo f(*X*) to a polynomial of degree *n* – 1 or lower, the space of residue classes can be identified with the space of polynomials of degree bounded by *n* – 1. A problem specific basis can be taken from Lagrange interpolation as the set of *n* polynomials

$$b_k(X) = \prod_{1 \leq j \leq n,\ j \neq k} (X - z_j), \quad k = 1, \ldots, n,$$

where $z_1, \ldots, z_n \in \mathbb{C}$ are pairwise different complex numbers. Note that the kernel functions for the Lagrange interpolation are $L_k(X) = \frac{b_k(X)}{b_k(z_k)}$.

For the multiplication operator applied to the basis polynomials one obtains from the Lagrange interpolation

$$\begin{aligned} X \cdot b_k(X) \mod f(X) = X \cdot b_k(X) - f(X) \ &= \sum_{j=1}^{n} \left(z_j \cdot b_k(z_j) - f(z_j) \right) \cdot \frac{b_j(X)}{b_j(z_j)} \\ &= z_k \cdot b_k(X) + \sum_{j=1}^{n} w_j \cdot b_j(X), \end{aligned}$$

where $w_j = -\frac{f(z_j)}{b_j(z_j)}$ are again the Weierstrass updates.

The companion matrix of f(*X*) is therefore

$$A = \operatorname{diag}(z_1, \ldots, z_n) + \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \cdot (w_1, \ldots, w_n).$$

From the transposed matrix case of the Gershgorin circle theorem it follows that all eigenvalues of *A*, that is, all roots of f(*X*), are contained in the union of the disks $D(a_{k,k}, r_k)$ with a radius $r_k = \sum_{j \neq k} |a_{j,k}|$.

Here one has $a_{k,k} = z_k + w_k$, so the centers are the next iterates of the Weierstrass iteration, and radii $r_k = (n-1)\,|w_k|$ that are multiples of the Weierstrass updates. If the roots of f(*X*) are all well isolated (relative to the computational precision) and the points $z_1, \ldots, z_n \in \mathbb{C}$ are sufficiently close approximations to these roots, then all the disks will become disjoint, so each one contains exactly one zero. The midpoints of the circles will be better approximations of the zeros.

Every conjugate matrix TAT^{-1} of *A* is as well a companion matrix of f(*X*). Choosing *T* as diagonal matrix leaves the structure of *A* invariant. The root close to z_k is contained in any isolated circle with center z_k regardless of *T*. Choosing the optimal diagonal matrix *T* for every index results in better estimates (see ref. Petkovic et al. 1995).

Convergence results

The connection between the Taylor series expansion and Newton's method suggests that the distance from $z_k + w_k$ to the corresponding root is of the order $O(|w_k|^2)$, if the root is well isolated from nearby roots and the approximation is sufficiently close to the root. So after the approximation is close, Newton's method converges *quadratically*; that is: the error is squared with every step (which will greatly reduce the error once it is less than 1). In the case of the Durand–Kerner method, convergence is quadratic if the vector $\vec{z} = (z_1, \ldots, z_n)$ is close to some permutation of the vector of the roots of f.

For the conclusion of linear convergence there is a more specific result (see ref. Petkovic et al. 1995). If the initial vector $\vec{z}$ and its vector of Weierstrass updates $\vec{w} = (w_1, \ldots, w_n)$ satisfies the inequality

$$\max_{1 \le k \le n} |w_k| \le \frac{1}{5n} \min_{1 \le j < k \le n} |z_k - z_j|,$$

then this inequality also holds for all iterates, all inclusion disks $D\left(z_k + w_k, (n-1)|w_k|\right)$ are disjoint and linear convergence with a contraction factor of *1/2* holds. Further, the inclusion disks can in this case be chosen as

$$D\left(z_k + w_k, \tfrac{1}{4}|w_k|\right) \qquad k = 1, \ldots, n,$$

each containing exactly one zero of f.

References

- Weierstraß, Karl (1891). "Neuer Beweis des Satzes, dass jede ganze rationale Function einer Veränderlichen dargestellt werden kann als ein Product aus linearen Functionen derselben Veränderlichen". *Sitzungsberichte der königlich preussischen Akademie der Wissenschaften zu Berlin*.
- Durand, E. (1960). "Equations du type *F*(*x*) = 0: Racines d'un polynome". In Masson et al. *Solutions Numériques des Equations Algébriques, vol. 1*.
- Kerner, Immo O. (1966). "Ein Gesamtschrittverfahren zur Berechnung der Nullstellen von Polynomen" [1]. *Numerische Mathematik* **8**: 290–294. doi:10.1007/BF02162564.
- Prešić, Marica (1980). "A convergence theorem for a method for simultaneous determination of all zeros of a polynomial". *Publications de l'institut mathematique (Beograd) (N.S.)* **28** (42): 158–168.
- Petkovic, M.S., Carstensen, C. and Trajkovic, M. (1995). "Weierstrass formula and zero-finding methods" [2]. *Numerische Mathematik* **69**: 353–372. doi:10.1007/s002110050097.
- Bo Jacoby, *Nulpunkter for polynomier*, CAE-nyt (a periodical for Dansk CAE Gruppe [Danish CAE Group]), 1988.
- Agnethe Knudsen, *Numeriske Metoder* (lecture notes), Københavns Teknikum.
- Bo Jacoby, *Numerisk løsning af ligninger*, Bygningsstatiske meddelelser (Published by Danish Society for Structural Science and Engineering) volume 63 no. 3-4, 1992, pp. 83–105.
- Gourdon, Xavier (1996). *Combinatoire, Algorithmique et Geometrie des Polynomes* [3]. Paris: Ecole Polytechnique.
- Victor Pan (May 2002): *Univariate Polynomial Root-Finding with Lower Computational Precision and Higher Convergence Rates* [4]. Tech-Report, City University of New York
- Neumaier, Arnold (2003). "Enclosing clusters of zeros of polynomials" [5]. *Journal of Computational and Applied Mathematics* **156**: 389. doi:10.1016/S0377-0427(03)00380-7.
- Jan Verschelde, *The method of Weierstrass (also known as the Durand-Kerner method)* [6], 2003.

External links

- *Ada Generic_Roots using the Durand-Kerner Method* [7] — an open-source implementation in Ada
- *Polynomial Roots* [8] — an open-source implementation in Java
- *Roots Extraction from Polynomials : The Durand-Kerner Method* [9] — contains a Java applet demonstration

References

[1] http://www.springerlink.com/content/q5p055l61pm63206
[2] http://www.springerlink.com/content/x467nejrv3c8hq8j
[3] http://algo.inria.fr/gourdon/thesis.html
[4] http://www.cs.gc.cuny.edu/tr/techreport.php?id=26
[5] http://www.mat.univie.ac.at/~neum/papers.html#polzer
[6] http://www2.math.uic.edu/~jan/mcs471f03/Project_Two/proj2/node2.html
[7] http://home.roadrunner.com/~jbmatthews/misc/groots.html
[8] http://sites.google.com/site/drjohnbmatthews/polyroots
[9] http://www.cpc.wmin.ac.uk/~spiesf/Solve/solve.html

Aberth method

The **Aberth method**, or **Aberth–Ehrlich method**, named after Oliver Aberth[1] and Louis W. Ehrlich,[2] is a root-finding algorithm for simultaneous approximation of all the roots of a univariate polynomial.

The fundamental theorem of algebra states that for each polynomial with complex coefficients there are as many roots as the degree of the polynomial. More specifically, each polynomial of degree *n* is identical to a product of *n* linear polynomials. The fundamental theorem in itself does not offer an efficient numerical procedure to compute those roots. Numerical algorithms that approximate all roots at once are the Weierstrass–(Durand–Kerner) method and the Aberth–Ehrlich method.

Description

Let $p(x) = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_1 x + p_0$ be a univariate polynomial of degree *n* with real or complex coefficients. Then there exist complex numbers $z_1^*, z_2^*, \ldots, z_n^*$, the roots of *p(x)*, that give the factorisation:

$$p(x) = p_n \cdot (x - z_1^*) \cdot (x - z_2^*) \cdots (x - z_n^*).$$

Although those numbers are unknown, upper and lower bounds of their absolute values are computable from the coefficients of the polynomial. Now one can pick *n* distinct numbers in the complex plane—randomly or evenly distributed—such that their absolute values obey the same bounds. The set of those numbers is called the initial approximation of the set of roots of *p(x)*.This approximation is iteratively improved by the following procedure.

Let $z_1, \ldots, z_n \in \mathbb{C}$ be the current approximations of the zeros of *p(x)*. Then offset numbers $w_1, \ldots, w_n \in \mathbb{C}$ are computed as

$$w_k = -\frac{\frac{p(z_k)}{p'(z_k)}}{1 - \frac{p(z_k)}{p'(z_k)} \cdot \sum_{j \neq k} \frac{1}{z_k - z_j}},$$

where *p'(z)* is the polynomial derivative of *p* evaluated in the point *z*.

The next set of approximations of roots of *p(x)* is then $z_1 + w_1, \ldots, z_n + w_n$. One can measure the quality of the current approximation by the values of the polynomial or by the size of the offsets.

Inside the formula of the Aberth method one can find elements of Newton's method and the Weierstrass–(Durand–Kerner) method. Details for an efficient implementation, esp. on the choice of good initial approximations, can be found in Bini (1996).[3]

Derivation from Newton's method

The iteration formula is the univariate Newton iteration for the function

$$F(x) = \frac{p(x)}{\prod_{j=0;\, j\neq k}^{n}(x - z_j)}$$

If the values $z_1, \ldots, z_n$ are already close to the roots of $p(x)$, then the rational function $F(x)$ is almost linear with poles at $z_1, \ldots, z_{k-1}, z_{k+1}, \ldots, z_n$ that direct the Newton iteration away from the roots of $p(x)$ that are close to them. That is, the corresponding bassins of attraction get rather small.

The Newton step in the univariate case is the reciprocal value to the logarithmic derivative

$$\begin{aligned}\frac{F'(x)}{F(x)} &= \frac{d}{dx}\ln|F(x)| \\ &= \frac{d}{dx}\Big(\ln|p(x)| - \sum_{j=0;\, j\neq k}^{n}\ln|x - z_j|\Big) \\ &= \frac{p'(x)}{p(x)} - \sum_{j=0;\, j\neq k}^{n}\frac{1}{x - z_j}\end{aligned}$$

Thus,

$$z_k' = z_k - \frac{F(x)}{F'(x)} = z_k - \frac{1}{\frac{p'(z_k)}{p(z_k)} - \sum_{j=0;\, j\neq k}^{n}\frac{1}{z_k - z_j}}$$

results in the Aberth–Ehrlich iteration.

The iteration may be executed in a simultaneous Jacobi-like iteration where first all new approximations are computed from the old approximations or in a sequential Gauss–Seidel-like iteration that uses each new approximation from the time it is computed.

Literature

[1] Aberth, Oliver (1973). "Iteration methods for finding all zeros of a polynomial simultaneously". *Math. Comp.* (Mathematics of Computation, Vol. 27, No. 122) **27** (122): 339–344. doi:10.2307/2005621. JSTOR 2005621.

[2] Ehrlich, Louis W. (1967). "A modified Newton method for polynomials". *Comm. ACM* **10** (2): 107–108. doi:10.1145/363067.363115.

[3] Bini, Dario Andrea (1996). "Numerical computation of polynomial zeros by means of Aberth's method" (http://www.springerlink.com/content/b35647833p354348). *Numerical Algorithms* **13** (2): 179–200. doi:10.1007/BF02207694. .

Splitting circle method

In mathematics, the **splitting circle method** is a numerical algorithm for the numerical factorization of a polynomial and, ultimately, for finding its complex roots. It was introduced by Arnold Schönhage in his 1982 paper *The fundamental theorem of algebra in terms of computational complexity* (Technical report, Mathematisches Institut der Universität Tübingen). A revised algorithm was presented by Victor Pan in 1998. An implementation was provided by Xavier Gourdon in 1996 for the Magma and PARI/GP computer algebra systems.

General description

The fundamental idea of the **splitting circle method** is to use methods of complex analysis, more precisely the residue theorem, to construct factors of polynomials. With those methods it is possible to construct a factor of a given polynomial $p(x) = x^n + p_{n-1}x^{n-1} + \cdots + p_0$ for any region of the complex plane with a piecewise smooth boundary. Most of those factors will be trivial, that is constant polynomials. Only regions that contain roots of *p(x)* result in nontrivial factors that have exactly those roots of *p(x)* as their own roots, preserving multiplicity.
In the numerical realization of this method one uses disks *D*(*c*,*r*) (center *c*, radius *r*) in the complex plane as regions. The boundary circle of a disk splits the set of roots of *p*(*x*) in two parts, hence the name of the method. To a given disk one computes approximate factors following the analytical theory and refines them using Newton's method. To avoid numerical instability one has to demand that all roots are well separated from the boundary circle of the disk. So to obtain a good splitting circle it should be embedded in a root free annulus *A*(*c*,*r*,*R*) (center *c*, inner radius *r*, outer radius *R*) with a large relative width *R/r*.

Repeating this process for the factors found, one finally arrives at an approximative factorization of the polynomial at a required precision. The factors are either linear polynomials representing well isolated zeros or higher order polynomials representing clusters of zeros.

Details of the analytical construction

Newton's identities are a bijective relation between the elementary symmetric polynomials of a tuple of complex numbers and its sums of powers. Therefore, it is possible to compute the coefficients of a polynomial

$$p(x) = x^n + p_{n-1}x^{n-1} + \cdots + p_0 = (x - z_1)\cdots(x - z_n)$$

(or of a factor of it) from the sums of powers of its zeros

$$t_m = z_1^m + \cdots + z_n^m\,,\ m = 0, 1, \ldots, n$$

by solving the triangular system that is obtained by comparing the powers of *u* in the following identity of formal power series

$$\begin{aligned} &a_{n-1} + 2\,a_{n-2}\,u + \cdots + (n-1)\,a_1\,u^{n-2} + n\,a_0\,u^{n-1} \\ &= -(1 + a_{n-1}\,u + \cdots + a_1\,u^{n-1} + a_0\,u^n)\cdot(t_1 + t_2\,u + t_3\,u^2 + \ldots + t_n\,u^{n-1} + \cdots). \end{aligned}$$

If $G \subset \mathbb{C}$ is a domain with piecewise smooth boundary *C* and if the zeros of *p*(*x*) are pairwise distinct and not on the boundary *C*, then from the residue theorem of residual calculus one gets

$$\frac{1}{2\pi\,i}\oint_C \frac{p'(z)}{p(z)} z^m\,dz = \sum_{z\in G:\,p(z)=0} \frac{p'(z)z^m}{p'(z)} = \sum_{z\in G:\,p(z)=0} z^m.$$

The identity of the left to the right side of this equation also holds for zeros with multiplicities. By using the Newton identities one is able to compute from those sums of powers the factor

$$f(x) := \prod_{z\in G:\,p(z)=0} (x - z)$$

of $p(x)$ corresponding to the zeros of $p(x)$ inside G. By polynomial division one also obtains the second factor $g(x)$ in $p(x) = f(x)g(x)$.

The commonly used regions are circles in the complex plane. Each circle gives raise to a split of the polynomial $p(x)$ in factors $f(x)$ and $g(x)$. Repeating this procedure on the factors using different circles yields finer and finer factorizations. This recursion stops after a finite number of proper splits with all factors being nontrivial powers of linear polynomials.

The challenge now consists in the conversion of this analytical procedure into a numerical algorithm with good running time. The integration is approximated by a finite sum of a numerical integration method, making use of the fast Fourier transform for the evaluation of the polynomials $p(x)$ and $p'(x)$. The polynomial $f(x)$ that results will only be an approximate factor. To ensure that its zeros are close to the zeros of p inside G and only to those, one must demand that all zeros of p are far away from the boundary C of the region G.

Basic numerical observation

(Schönhage 1982) Let $p \in \mathbb{C}[X]$ be a polynomial of degree n has k zeros inside the circle of radius *1/2* and the remaining n-k zeros outside the circle of radius *2*. With $N=O(k)$ large enough, the approximation of the contour integrals using N points results in an approximation f_0 of the factor f with error

$$\|f - f_0\| \leq 2^{2k-N}\, nk\, 100/98,$$

where the norm of a polynomial is the sum of the moduli of its coefficients.

Since the zeros of a polynomial are continuous in its coefficients, one can make the zeros of f_0 as close as wanted to the zeros of f by choosing N large enough. However, one can improve this approximation faster using a Newton method. Division of p with remainder yields an approximation g_0 of the remaining factor g. Now

$$p - f_0 g_0 = (f - f_0)g_0 + (g - g_0)f_0 + (f - f_0)(g - g_0),$$

so discarding the last second order term one has to solve $p - f_0 g_0 = f_0 \Delta g + g_0 \Delta f$ using any variant of the extended euclidian algorithm to obtain the incremented approximations $f_1 = f_0 + \Delta f$ and $g_1 = g_0 + \Delta g$. This is repeated until the increments are zero relative to the chosen precision.

Graeffe iteration

The crucial step in this method is to find an annulus of relative width *4* in the complex plane that contains no zeros of p and contains approximately as many zeros of p inside as outside of it. Any annulus of this characteristic can be transformed, by translation and scaling of the polynomial, into the annulus between the radii 1/2 and 2 around the origin. But, not every polynomial admits such a splitting annulus.

To remedy this situation, the Graeffe iteration is applied. It computes a sequence of polynomials

$$p_0 = p, \qquad p_{j+1}(x) = (-1)^{\deg p} p(\sqrt{x})\, p(-\sqrt{x}),$$

where the roots of $p_j(x)$ are the 2^j-th dyadic powers of the roots of the initial polynomial p. By splitting $p_j(x) = e(x) + x\, o(x)$ into even and odd parts, the succeeding polynomial is obtained by purely arithmetic operations as $p_{j+1}(x) = (-1)^{\deg p}(e(x)^2 - x\, o(x)^2)$. The ratios of the absolute moduli of the roots increase by the same power 2^j and thus tend to infinity. Choosing j large enough one finally finds a splitting annulus of relative width 4 around the origin

The approximate factorization of $p_j(x) \approx f_j(x)\, g_j(x)$ is now to be lifted back to the original polynomial. To this end an alternation of Newton steps and Pade approximations is used. It is easy to check that

$$\frac{p_{j-1}(x)}{g_j(x^2)} \approx \frac{f_{j-1}(x)}{g_{j-1}(-x)}$$

holds. The polynomials on the left side are known in step *j*, the polynomials on the right side can be obtained as Padé approximants of the corresponding degrees for the power series expansion of the fraction on the left side.

Finding a good circle

Making use of the Graeffe iteration and any known estimate for the absolute value of the largest root one can find estimates *R* of this absolute value of any precision. Now one computes estimates for the largest and smallest distances $R_j > r_j > 0$ of any root of *p*(*x*) to any of the five center points 0, 2*R*, −2*R*, 2*Ri*, −2*Ri* and selects the one with the largest ratio R_j/r_j between the two. By this construction it can be guaranteed that $R_j/r_j > e^{0.3} \approx 1.35$ for at least one center. For such a center there has to be a root-free annulus of relative width $e^{0.3/n} \approx 1 + \frac{0.3}{n}$. After $3 + \log_2(n)$ Graeffe iterations, the corresponding annulus of the iterated polynomial has a relative width greater than 11 > 4, as required for the initial splitting described above (see Schönhage (1982)). After $4 + \log_2(n) + \log_2(2 + \log_2(n))$ Graeffe iterations, the corresponding annulus has a relative width greater than $2^{13.8} \cdot n^{6.9} > (64 \cdot n^3)^2$, allowing a much simplified initial splitting (see Malajovich/Zubelli (1997)).

To locate the best root-free annulus one uses a consequence of the Rouché theorem: For *k* = 1, ..., *n* − 1 the polynomial equation

$$0 = \sum_{j \neq k} |p_j| u^j - |p_k| u^k,$$

u > 0, has, by Descartes' rule of signs zero or two positive roots $u_k < v_k$. In the latter case, there are exactly *k* roots inside the (closed) disk $D(0, u_k)$ and $A(0, u_k, v_k)$ is a root-free (open) annulus.

References

- Schönhage, Arnold (1982): *The fundamental theorem of algebra in terms of computational complexity.* [1] Preliminary Report, Math. Inst. Univ. Tübingen (1982), 49 pages. (ps.gz)
- Gourdon, Xavier (1996). *Combinatoire, Algorithmique et Geometrie des Polynomes* [3]. Paris: Ecole Polytechnique.
- V. Y. Pan (1996). "Optimal and nearly optimal algorithms for approximating polynomial zeros". *Comput. Math. Appl.* **31** (12): 97–138. doi:10.1016/0898-1221(96)00080-6.
- V. Y. Pan (1997). "Solving a polynomial equation: Some history and recent progresses". *SIAM Review* **39** (2): 187–220. doi:10.1137/S0036144595288554.
- Gregorio Malajovich and Jorge P. Zubelli (1997). "A fast and stable algorithm for splitting polynomials" [2]. *Computers & Mathematics with Applications*. 33 **No 3** (2): 1–23. doi:10.1016/S0898-1221(96)00233-7.
- Pan, Victor (1998). *Algorithm for Approximating Complex Polynomial Zeros* [3]
- Pan, Victor (2002). *Univariate Polynomials: Nearly Optimal Algorithms for Numerical Factorization and Root-finding* [4]
- Magma documentation. Real and Complex Fields: Element Operations [5].

References

[1] http://www.informatik.uni-bonn.de/~schoe/fdthmrep.ps.gz
[2] http://www.labma.ufrj.br/~gregorio/papers.php#splittin
[3] http://algo.inria.fr/seminars/sem97-98/pan.html
[4] http://comet.lehman.cuny.edu/vpan/pdf/JSCOptimal.pdf
[5] http://magma.maths.usyd.edu.au/magma/handbook/text/223#2021

Graeffe's method

In mathematics, **Graeffe's method** or **Dandelin–Graeffe method** is an algorithm for finding all of the roots of a polynomial. It was developed independently by Germinal Pierre Dandelin in 1826 and Karl Heinrich Gräffe in 1837. Lobachevsky in 1834 also discovered the principal idea of the method.[1] The method separates the roots of a polynomial by squaring them repeatedly. This squaring of the roots is done implicitly, that is, only working on the coefficients of the polynomial. Finally, Viète's formulas are used in order to approximate the roots.

Dandelin–Graeffe iteration

Let $p(x)$ be an nth degree polynomial.

$$p(x) = (x - x_1)(x - x_2) \cdots (x - x_n)$$

Then

$$p(-x) = (-1)^n (x + x_1)(x + x_2) \cdots (x + x_n).$$

Let $q(x)$ be the polynomial which has the squares $x_1^2,\ x_2^2, \ldots,\ x_n^2$ as its roots,

$$q(x) = (x - x_1^2)(x - x_2^2) \cdots (x - x_n^2).$$

Hence by the binomial identity $x^2 - x_k^2 = (x - x_k)(x + x_k)$

$$q(x^2) = (x^2 - x_1^2)(x^2 - x_2^2) \cdots (x^2 - x_n^2) = (-1)^n p(x)p(-x).$$

The polynomial $q(x)$ can now be computed by algebraic operations on the coefficients of the polynomial $p(x)$ alone. Write

$$p(x) = x^n + a_1 x^{n-1} + \cdots + a_{n-1}x + a_n$$

and

$$q(x) = x^n + b_1 x^{n-1} + \cdots + b_{n-1}x + b_n,$$

then the coefficients are related by

$$b_k = (-1)^k a_k^2 + 2\sum_{j=0}^{k-1} (-1)^j a_j a_{2k-j},\ \text{with } a_0 = b_0 = 1.$$

Graeffe observed that one obtains a simplified algebraic expression for $q(x)$ when separating $p(x)$ into its odd and even parts,

$$p(x) = p_e(x^2) + x\, p_o(x^2) \implies q(x) = (-1)^n (p_e(x)^2 - x\, p_o(x)^2).$$

This expression involves the squaring of two polynomials of only half the degree, and is therefore used in most implementations of the method.

Iterating this procedure several times separates the roots with respect to their magnitudes. Repeating k times gives a polynomial

$$q^k(y) = y^n + a^k{}_1\, y^{n-1} + \cdots + a^k{}_{n-1}\, y + a^k{}_n$$

of degree n with roots $y_1 = x_1^{2^k},\ y_2 = x_2^{2^k},\ \ldots,\ y_n = x_n^{2^k}$. If the magnitudes of the roots of the original polynomial were separated by some factor $\rho > 1$, that is, $|x_k| \geq \rho\, |x_{k+1}|$, then the roots of the k-th iterate are

separated by a fast growing factor $\rho^{2^k} \geq 1 + 2^k(\rho - 1)$.

Classical Graeffe's method

Next the Vieta relations are used

$$\begin{aligned} a_1^k &= -(y_1 + y_2 + \cdots + y_n) \\ a_2^k &= y_1y_2 + y_1y_3 + \cdots + y_{n-1}y_n \\ &\vdots \\ a_n^k &= (-1)^n(y_1y_2 \cdots y_n). \end{aligned}$$

If the roots $x_1, \ldots, x_n$ are sufficiently separated, say by a factor $\rho > 1$, $|x_m| \geq \rho|x_{m+1}|$, then the iterated powers $y_1, y_2, \ldots, y_n$ of the roots are separated by the factor ρ^{2^k}, which quickly becomes very big.

The coefficients of the iterated polynomial can then be approximated by their leading term,

$$a_1^k \approx -y_1$$
$$a_2^k \approx y_1y_2$$ and so on,

implying

$$y_1 \approx -a_1^k,\ y_2 \approx -a_2^k/a_1^k,\ \ldots\ y_n \approx -a_n^k/a_{n-1}^k.$$

Finally, logarithms are used in order to find the absolute values of the roots of the original polynomial. These magnitudes alone are already useful to generate meaningful starting points for other root-finding methods.

To also obtain the angle of these roots, a multitude of methods has been proposed, the most simple one being to successively compute the square root of a (possibly complex) root of $q^m(y)$, *m* ranging from *k* to 1, and testing which of the two sign variants is a root of $q^{m-1}(x)$. Before continuing to the roots of $q^{m-2}(x)$, it might be necessary to numerically improve the accuracy of the root approximations for $q^{m-1}(x)$, for instance by Newton's method.

Graeffe's method works best for polynomials with simple real roots, though it can be adapted for polynomials with complex roots and coefficients, and roots with higher multiplicity. For instance, it has been observed[2] that for a root $x_{\ell+1} = x_{\ell+2} = \ldots = x_{\ell+d}$ with multiplicity *d*, the fractions

$$\left|\frac{(a_{\ell+i}^{m-1})^2}{a_{\ell+i}^m}\right| \text{ tend to } \binom{d}{i}$$

for $i = 0, 1, \ldots, d$. This allows to estimate the multiplicity structure of the set of roots.

From a numerical point of view, this method is problematic since the coefficients of the iterated polynomials span very quickly many orders of magnitude, which implies serious numerical errors. One second, but minor concern is that many different polynomials lead to the same Graeffe iterates.

Tangential Graeffe method

This method replaces the numbers by truncated power series of degree 1. Symbolically, this is achieved by introducing an "algebraic infinitesimal" ε with the defining property $\varepsilon^2 = 0$. Then the polynomial $p(x + \varepsilon) = p(x) + \varepsilon\, p'(x)$ has roots $x_m - \varepsilon$, with powers

$$(x_m - \varepsilon)^{2^k} = x_m^{2^k} - \varepsilon\, 2^k\, x_m^{2^k-1} = y_m + \varepsilon\, \dot{y}_m.$$

Thus the value of x_m is easily obtained as fraction $x_m = -\frac{2^k\, y_m}{\dot{y}_m}$.

This kind of computation with infinitesimals is easy to implement analogous to the computation with complex numbers. If one assumes complex coordinates or an initial shift by some randomly chosen complex number, then all roots of the polynomial will be distinct and consequently recoverable with the iteration.

Renormalization

Every polynomial can be scaled in domain and range such that in the resulting polynomial the first and the last coefficient have size one. If the size of the inner coefficients is bounded by *M*, then the size of the inner coefficients after one stage of the Graeffe iteration is bounded by nM^2. After *k* stages one gets the bound $n^{2^k-1}M^{2^k}$ for the inner coefficients.

To overcome the limit posed by the growth of the powers, Malajovich–Zubelli propose to represent coefficients and intermediate results in the *k*th stage of the algorithm by a scaled polar form

$$c = \alpha\, e^{-2^k r}$$

where $\alpha = \dfrac{c}{|c|}$ is a complex number of unit length and $r = -2^{-k}\log|c|$ is a positive real. Splitting off the power 2^k in the exponent reduces the absolute value of *c* to the corresponding dyadic root. Since this preserves the magnitude of the (representation of the) initial coefficients, this process was named renormalization.

Multiplication of two numbers of this type is straightforward, whereas addition is performed following the factorization $c_3 = c_1 + c_2 = |c_1| \cdot \left(\alpha_1 + \alpha_2 \frac{|c_2|}{|c_1|}\right)$, where c_1 is chosen as the larger of both numbers, that is, $r_1 < r_2$. Thus

$$\alpha_3 = \tfrac{s}{|s|} \text{ and } r_3 = r_1 + 2^{-k}\log|s| \text{ with } s = \alpha_1 + \alpha_2\, e^{2^k(r_1 - r_2)}.$$

The coefficients $a_0, a_1, \ldots, a_n$ of the final stage *k* of the Graeffe iteration, for some reasonably large value of *k*, are represented by pairs (α_m, r_m), $m = 0, \ldots, n$. By identifying the corners of the convex envelope of the point set $\{(m, r_m) : m = 0, \ldots, n\}$ one can determine the multiplicities of the roots of the polynomial. Combining this renormalization with the tangent iteration one can extract directly from the coefficients at the corners of the envelope the roots of the original polynomial.

References

[1] Alston Scott Householder: *Dandelin, Lobačevskiĭ, or Graeffe?*, Amer. Math. Monthly, 66 (1959), pp. 464–466 (on JSTOR) (http://www.jstor.org/stable/2310626)

[2] G. C. Best: *Notes on the Graeffe method of root squaring*, Amer. Math. Monthly, 56, (1949) (on JSTOR) (http://www.jstor.org/stable/2306166)

- Weisstein, Eric W., " Graeffe's Method (http://mathworld.wolfram.com/GraeffesMethod.html)" from MathWorld.
- G. Malajovich, J. P. Zubelli: "Tangent Graeffe Iteration". Scientific Commons (http://en.scientificcommons.org/221845), Numerische Mathematik 89, No.4, 749-782 (2001). ISSN 0029-599X; ISSN 0945-3245
- Module for Graeffe's Method by John H. Mathews (http://math.fullerton.edu/mathews/n2003/GraeffeMethodMod.html)

Greatest common divisor

In mathematics, the **greatest common divisor** (**gcd**), also known as the **greatest common factor** (**gcf**), or **highest common factor** (**hcf**), of two or more non-zero integers, is the largest positive integer that divides the numbers without a remainder. For example, the GCD of 8 and 12 is 4.

This notion can be extended to polynomials, see greatest common divisor of two polynomials.

Overview

Notation

In this article we will denote the greatest common divisor of two integers *a* and *b* as gcd(*a*,*b*). Some older textbooks use (*a*,*b*).[1][2]

Example

The number 54 can be expressed as a product of two other integers in several different ways:

$$54 \times 1 = 27 \times 2 = 18 \times 3 = 9 \times 6.$$

Thus the **divisors of 54** are:

$$1, 2, 3, 6, 9, 18, 27, 54.$$

Similarly **the divisors of 24** are:

$$1, 2, 3, 4, 6, 8, 12, 24.$$

The numbers that these two lists share in common are the **common divisors** of 54 and 24:

$$1, 2, 3, 6.$$

The greatest of these is 6. That is the **greatest common divisor** of 54 and 24. One writes:

$$\gcd(54, 24) = 6.$$

Reducing fractions

The greatest common divisor is useful for reducing fractions to be in lowest terms. For example, gcd(42, 56) = 14, therefore,

$$\frac{42}{56} = \frac{3 \cdot 14}{4 \cdot 14} = \frac{3}{4}.$$

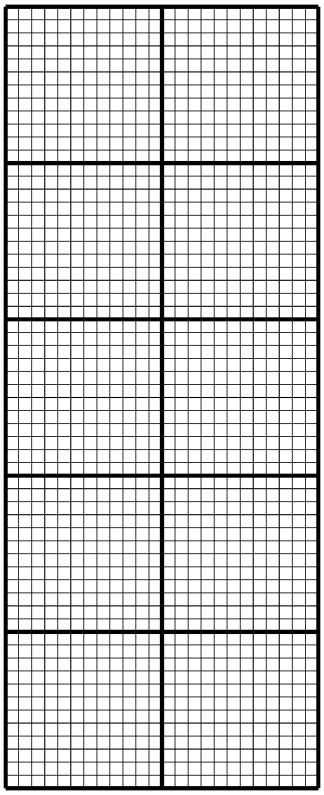
Coprime numbers

Two numbers are called *relatively prime*, or *coprime* if their greatest common divisor equals 1. For example, 9 and 28 are relatively prime.

A geometric view

For example, a 24-by-60 rectangular area can be divided into a grid of: 1-by-1 squares, 2-by-2 squares, 3-by-3 squares, 4-by-4 squares, 6-by-6 squares or 12-by-12 squares. Therefore, 12 is the greatest common divisor of 24 and 60. A 24-by-60 rectangular area can be divided into a grid of 12-by-12 squares, with two squares along one edge (24/12 = 2) and five squares along the other (60/12 = 5).

Calculation



A 24-by-60 rectangle is covered with ten 12-by-12 square tiles, where 12 is the GCD of 24 and 60. More generally, an *a*-by-*b* rectangle can be covered with square tiles of side-length *c* only if *c* is a common divisor of *a* and *b*.

Using prime factorizations

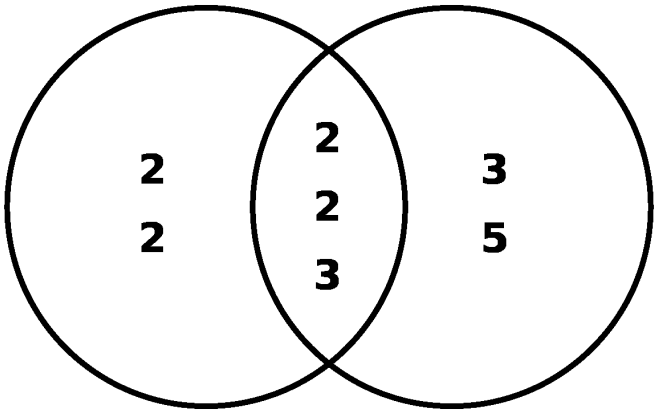
Greatest common divisors can in principle be computed by determining the prime factorizations of the two numbers and comparing factors, as in the following example: to compute gcd(18, 84), we find the prime factorizations $18 = 2 \cdot 3^2$ and $84 = 2^2 \cdot 3 \cdot 7$ and notice that the "overlap" of the two expressions is $2 \cdot 3$; so gcd(18, 84) = 6. In practice, this method is only feasible for small numbers; computing prime factorizations in general takes far too long.

Here is another concrete example, illustrated by a Venn diagram. Suppose it is desired to find the greatest common divisor of 48 and 180. First, find the prime factorizations of the two numbers:

$48 = 2 \times 2 \times 2 \times 2 \times 3,$

$180 = 2 \times 2 \times 3 \times 3 \times 5.$

What they share in common is two "2"s and a "3":



Least common multiple = $2 \times 2 \times 2 \times 2 \times 3 \times 3 \times 5 = 720$

Greatest common divisor = $2 \times 2 \times 3 = 12$.

Using Euclid's algorithm

A much more efficient method is the Euclidean algorithm, which uses the division algorithm in combination with the observation that the gcd of two numbers also divides their difference. To compute gcd(48,18), divide 48 by 18 to get a quotient of 2 and a remainder of 12. Then divide 18 by 12 to get a quotient of 1 and a remainder of 6. Then divide 12 by 6 to get a remainder of 0, which means that 6 is the gcd. Note that we ignored the quotient in each step except to notice when the remainder reached 0, signalling that we had arrived at the answer. Formally the algorithm can be described as:

$$\gcd(a, 0) = a$$

$$\gcd(a,b) = \gcd(b, a - b\left\lfloor\frac{a}{b}\right\rfloor).$$

Which also could be written as

$$\gcd(a,0) = a$$
$$\gcd(a,b) = \gcd(b, a \bmod b).$$

If the arguments are both greater than zero then the algorithm can be written in more elementary terms as follows:

$$\gcd(a,a) = a$$
$$\gcd(a,b) = \gcd(a-b,b) \quad \text{, if } b<a$$
$$\gcd(a,b) = \gcd(a,b-a) \quad \text{, if } a<b$$

Other methods

If a and b are not both zero, the greatest common divisor of a and b can be computed by using least common multiple (lcm) of a and b:

$$\gcd(a,b) = \frac{a \cdot b}{\operatorname{lcm}(a,b)}.$$

Keith Slavin has shown that for odd $a \geq 1$:

$$\gcd(a,b) = \log_2 \prod_{k=0}^{a-1}(1 + e^{-2i\pi kb/a})$$

which is a function that can be evaluated for complex b.[3] Wolfgang Schramm has shown that

$$\gcd(a,b) = \sum_{k=1}^{a} \exp(2\pi ikb/a) \cdot \sum_{d|a} \frac{c_d(k)}{d}$$

is an entire function in the variable b for all positive integers a where $c_d(k)$ is Ramanujan's sum.[4] Marcelo Polezzi has shown that:

$$\gcd(a,b) = (2\sum_{k=1}^{a-1}\lfloor kb/a \rfloor) + (a + b - ab)$$

for positive integers a and b.[5] Donald Knuth proved the following reduction:

$$\gcd(2^a - 1, 2^b - 1) = 2^{\gcd(a,b)} - 1$$

for non-negative integers a and b, where a and b are not both zero.[6] More generally

$$\gcd(n^a - 1, n^b - 1) = n^{\gcd(a,b)} - 1$$

which can easily be proven by considering the Euclidean algorithm in base n.

Complexity

The existence of the Euclidean algorithm places (the decision problem version of) the greatest common divisor problem in P, the class of problems solvable in polynomial time. The GCD problem is not known to be in NC, and so there is no known way to parallelize its computation across many processors; nor is it known to be P-complete, which would imply that it is unlikely to be possible to parallelize GCD computation. In this sense the GCD problem is analogous to e.g. the integer factorization problem, which has no known polynomial-time algorithm, but is not known to be NP-complete. Shallcross et al. showed that a related problem (EUGCD, determining the remainder sequence arising during the Euclidean algorithm) is NC-equivalent to the problem of integer linear programming with two variables; if either problem is in **NC** or is **P-complete**, the other is as well.[7] Since **NC** contains NL, it is also unknown whether a space-efficient algorithm for computing the GCD exists, even for nondeterministic Turing machines.

Although the problem is not known to be in **NC**, parallel algorithms with time superior to the Euclidean algorithm exist; the best known deterministic algorithm is by Chor and Goldreich, which (in the CRCW-PRAM model) can solve the problem in $O(n/\log n)$ time with $n^{1+\varepsilon}$ processors.[8] Randomized algorithms can solve the problem in $O((\log n)^2)$ time on $\exp\left[O\left(\sqrt{n \log n}\right)\right]$ processors (note this is superpolynomial).[9]

Properties

- Every common divisor of a and b is a divisor of $\gcd(a, b)$.
- $\gcd(a, b)$, where a and b are not both zero, may be defined alternatively and equivalently as the smallest positive integer d which can be written in the form $d = a \cdot p + b \cdot q$ where p and q are integers. This expression is called Bézout's identity. Numbers p and q like this can be computed with the extended Euclidean algorithm.
- $\gcd(a, 0) = |a|$, for $a \neq 0$, since any number is a divisor of 0, and the greatest divisor of a is $|a|$. This is usually used as the base case in the Euclidean algorithm.
- If a divides the product $b \cdot c$, and $\gcd(a, b) = d$, then a/d divides c.
- If m is a non-negative integer, then $\gcd(m \cdot a, m \cdot b) = m \cdot \gcd(a, b)$.
- If m is any integer, then $\gcd(a + m \cdot b, b) = \gcd(a, b)$.
- If m is a nonzero common divisor of a and b, then $\gcd(a/m, b/m) = \gcd(a, b)/m$.
- The gcd is a multiplicative function in the following sense: if a_1 and a_2 are relatively prime, then $\gcd(a_1 \cdot a_2, b) = \gcd(a_1, b) \cdot \gcd(a_2, b)$.
- The gcd is a commutative function: $\gcd(a, b) = \gcd(b, a)$.
- The gcd is an associative function: $\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c)$.
- The gcd of three numbers can be computed as $\gcd(a, b, c) = \gcd(\gcd(a, b), c)$, or in some different way by applying commutativity and associativity. This can be extended to any number of numbers.
- $\gcd(a, b)$ is closely related to the least common multiple $\operatorname{lcm}(a, b)$: we have

  $$\gcd(a, b) \cdot \operatorname{lcm}(a, b) = a \cdot b.$$

  This formula is often used to compute least common multiples: one first computes the gcd with Euclid's algorithm and then divides the product of the given numbers by their gcd.
- The following versions of distributivity hold true:

  $$\gcd(a, \operatorname{lcm}(b, c)) = \operatorname{lcm}(\gcd(a, b), \gcd(a, c))$$

  $$\operatorname{lcm}(a, \gcd(b, c)) = \gcd(\operatorname{lcm}(a, b), \operatorname{lcm}(a, c)).$$
- It is useful to define $\gcd(0, 0) = 0$ and $\operatorname{lcm}(0, 0) = 0$ because then the natural numbers become a complete distributive lattice with gcd as meet and lcm as join operation. This extension of the definition is also compatible with the generalization for commutative rings given below.
- In a Cartesian coordinate system, $\gcd(a, b)$ can be interpreted as the number of points with integral coordinates on the straight line joining the points (0, 0) and (a, b), excluding (0, 0).

Probabilities and expected value

In 1972, James E. Nymann showed that the probability that k independently chosen integers are coprime is $1/\zeta(k)$.[10] (See coprime for a derivation.) This result was extended in 1987 to show that the probability that k random integers has greatest common divisor d is $d^{-k}/\zeta(k)$.[11]

Using this information, the expected value of the greatest common divisor function can be seen (informally) to not exist when $k = 2$. In this case the probability that the gcd equals d is $d^{-2}/\zeta(2)$, and since $\zeta(2) = \pi^2/6$ we have

$$\mathrm{E}(2) = \sum_{d=1}^{\infty} d \frac{6}{\pi^2 d^2} = \frac{6}{\pi^2} \sum_{d=1}^{\infty} \frac{1}{d}.$$

This last summation is the harmonic series, which diverges. However, when $k \geq 3$, the expected value is well-defined, and by the above argument, it is

$$\mathrm{E}(k) = \sum_{d=1}^{\infty} d^{1-k}\zeta(k)^{-1} = \frac{\zeta(k-1)}{\zeta(k)}.$$

For $k = 3$, this is approximately equal to 1.3684. For $k = 4$, it is approximately 1.1106.

The gcd in commutative rings

The notion of greatest common divisor can more generally be defined for elements of an arbitrary commutative ring, although in general there need not exist one for every pair of elements.

If R is a commutative ring, and a and b are in R, then an element d of R is called a *common divisor* of a and b if it divides both a and b (that is, if there are elements x and y in R such that $d \cdot x = a$ and $d \cdot y = b$). If d is a common divisor of a and b, and every common divisor of a and b divides d, then d is called a *greatest common divisor* of a and b.

Note that with this definition, two elements a and b may very well have several greatest common divisors, or none at all. If R is an integral domain then any two gcd's of a and b must be associate elements, since by definition either one must divide the other; indeed if a gcd exists, any one of its associates is a gcd as well. Existence of a gcd is not assured in arbitrary integral domains. However if R is a unique factorization domain, then any two elements have a gcd, and more generally this is true in gcd domains. If R is a Euclidean domain in which euclidean division is given algorithmically (as is the case for instance when $R = F[X]$ where F is a field, or when R is the ring of Gaussian integers), then greatest common divisors can be computed using a form of the Euclidean algorithm based on the division procedure.

The following is an example of an integral domain with two elements that do not have a gcd:

$$R = \mathbb{Z}\left[\sqrt{-3}\,\right], \quad a = 4 = 2 \cdot 2 = \left(1 + \sqrt{-3}\,\right)\left(1 - \sqrt{-3}\,\right), \quad b = \left(1 + \sqrt{-3}\,\right) \cdot 2.$$

The elements 2 and $1 + \sqrt{(-3)}$ are two "maximal common divisors" (i.e. any common divisor which is a multiple of 2 is associated to 2, the same holds for $1 + \sqrt{(-3)}$), but they are not associated, so there is no greatest common divisor of a and b.

Corresponding to the Bezout property we may, in any commutative ring, consider the collection of elements of the form $pa + qb$, where p and q range over the ring. This is the ideal generated by a and b, and is denoted simply (a, b). In a ring all of whose ideals are principal (a principal ideal domain or PID), this ideal will be identical with the set of multiples of some ring element d; then this d is a greatest common divisor of a and b. But the ideal (a, b) can be useful even when there is no greatest common divisor of a and b. (Indeed, Ernst Kummer used this ideal as a replacement for a gcd in his treatment of Fermat's Last Theorem, although he envisioned it as the set of multiples of some hypothetical, or *ideal*, ring element d, whence the ring-theoretic term.)

Notes

[1] Long (1972, p. 33)

[2] Pettofrezzo & Byrkit (1970, p. 34)

[3] Slavin, Keith R. (2008). "Q-Binomials and the Greatest Common Divisor" (http://www.integers-ejcnt.org/vol8.html). *Integers Electronic Journal of Combinatorial Number Theory* (University of West Georgia, Charles University in Prague) **8**: A5. . Retrieved 2008-05-26.

[4] Schramm, Wolfgang (2008). "The Fourier transform of functions of the greatest common divisor" (http://www.integers-ejcnt.org/vol8.html). *Integers Electronic Journal of Combinatorial Number Theory* (University of West Georgia, Charles University in Prague) **8**: A50. . Retrieved 2008-11-25.

[5] Polezzi, Marcelo (1997). "A Geometrical Method for Finding an Explicit Formula for the Greatest Common Divisor". *Amer. Math. Monthly* (Mathematical Association of America) **104** (5): 445–446. doi:10.2307/2974739. JSTOR 2974739.

[6] Knuth, Donald E.; R. L. Graham, O. Patashnik (March 1994). *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley. ISBN 0-201-55802-5.

[7] Shallcross, D.; Pan, V.; Lin-Kriz, Y. (1993). "The NC equivalence of planar integer linear programming and Euclidean GCD" (http://www.icsi.berkeley.edu/pubs/techreports/tr-92-041.pdf). *34th IEEE Symp. Foundations of Computer Science*. pp. 557–564. .

[8] Chor, B.; Goldreich, O. (1990). "An improved parallel algorithm for integer GCD". *Algorithmica* **5** (1–4): 1–10. doi:10.1007/BF01840374.

[9] Adleman, L. M.; Kompella, K. (1988). "Using smoothness to achieve parallelism". *20th Annual ACM Symposium on Theory of Computing*. New York. pp. 528–538. doi:10.1145/62212.62264. ISBN 0-89791-264-0.

[10] Nymann, J. E. (1972). "On the probability that *k* positive integers are relatively prime". *Journal of Number Theory* **4** (5): 469–473. doi:10.1016/0022-314X(72)90038-8.

[11] Chidambaraswamy, J.; Sitarmachandrarao, R. (1987). "On the probability that the values of *m* polynomials have a given g.c.d.". *Journal of Number Theory* **26** (3): 237–245. doi:10.1016/0022-314X(87)90081-3.

References

- Long, Calvin T. (1972), *Elementary Introduction to Number Theory* (2nd ed.), Lexington: D. C. Heath and Company
- Pettofrezzo, Anthony J.; Byrkit, Donald R. (1970), *Elements of Number Theory*, Englewood Cliffs: Prentice Hall

Further reading

- Donald Knuth. *The Art of Computer Programming*, Volume 2: *Seminumerical Algorithms*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89684-2. Section 4.5.2: The Greatest Common Divisor, pp. 333–356.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 31.2: Greatest common divisor, pp. 856–862.
- Saunders MacLane and Garrett Birkhoff. *A Survey of Modern Algebra*, Fourth Edition. MacMillan Publishing Co., 1977. ISBN 0-02-310070-2. 1–7: "The Euclidean Algorithm."

External links

- greatest common divisor at Everything2.com (http://everything2.com/?node_id=482506)
- Greatest Common Measure: The Last 2500 Years (http://www.stepanovpapers.com/gcd.pdf), by Alexander Stepanov

Article Sources and Contributors

Root-finding algorithm *Source*: http://en.wikipedia.org/w/index.php?oldid=491551654 *Contributors*: Access Denied, Adam majewski, Ahmadabdolkader, Akritas2, Anonymous Dissident, Army1987, Berland, Bo Jacoby, CRGreathouse, CaritasUbi, Cat2020, Catslash, Charles Matthews, D.Lazard, David Johnson, Decrease789, Denitsa The Truth Seeker, Diberri, Elwikipedista, EmreDuran, Favonian, Fredrik, Frencheigh, Gco, Giftlite, Happyrabbit, Hermitian, Hike395, Hvestermark, Immunize, JRSpriggs, Jaded-view, Jaredwf, Jitse Niesen, JonMcLoone, Jrvz, Jujutacular, KDesk, KSmrq, Kakoui, KnightRider, Leonard G., Linas, Lingwitt, Lostella, LutzL, Magmait, Marshcmb, Matthewmoncek, Michael Hardy, Niceguyedc, Oleg Alexandrov, Oliphaunt, Oyd11, Prohlep, QueenCake, Qwfp, Raymondwinn, Rhubbarb, Rjwilmsi, Romanski, Ronhjones, Rschwieb, Stevenj, The Anome, Thetwentyone, Wonderstruck, 71 anonymous edits

Multiplicity (mathematics) *Source*: http://en.wikipedia.org/w/index.php?oldid=489867277 *Contributors*: 16@r, Akriasas, Aleph4, Alpha Beta Epsilon, Andreas Rejbrand, Anthony Appleyard, Arcette, Bob.v.R, CBM, Charles Matthews, Dhollm, DionysosProteus, Fgb, Giftlite, Gmelli, GreenReaper, He Who Is, JCSantos, JRSpriggs, Jab843, Jitse Niesen, Jujutacular, Kintetsubuffalo, Lapaz, Linas, Magmait, Maksim-e, Marc van Leeuwen, Mattbuck, Melchoir, Michael Hardy, Monsterman222, Nickalh50, Oleg Alexandrov, Patrick, Qutezuce, Salix alba, Siddhant, Smalljim, Tanner Swett, Tiny green, Toon05, Wang ty87916, Yoshigev, 29 anonymous edits

Descartes' rule of signs *Source*: http://en.wikipedia.org/w/index.php?oldid=491019207 *Contributors*: 478jjjz, 777sms, A. Pichler, Adam Field, Adavis444, Alansohn, Alethiareg, Bender235, Bento00, Charles Matthews, Chenxlee, Chuunen Baka, DavidMcKenzie, Duoduoduo, Dzordzm, Estudiarme, FHGJ, GNB, Gandalf61, Gazpacho, Geometry guy, Giftlite, Haihe, Jeekc, Jeepday, JimVC3, Lechatjaune, Lou Crazy, Magic Window, Mets501, Mglg, Michael Hardy, Mild Bill Hiccup, Nishantsah, PV=nRT, Palladinus, Pdebart, Reywas92, Salix alba, Silly rabbit, Sinblox, Spacepotato, Tide rolls, Tosha, Vroo, 61 anonymous edits

Derivative *Source*: http://en.wikipedia.org/w/index.php?oldid=495938901 *Contributors*: 10p12a0195, 123Mike456Winston789, 127, 24.44.206.xxx, A. Parrot, A876, ABCD, AbcXyz, Acannas, Ace Frahm, Ahmad87, Akshaysrinivasan, Alex Bakharev, Algebraist, AllTheThings, AllyUnion, Aly89, Andre Engels, Andrejj, Andres, Anonymous Dissident, Anwar saadat, ArglebargleIV, Art LaPella, Arthur543, Ash4Math, AstroHurricane001, Audiovideo, AugPi, AvicAWB, Aw.rootbeer1, AxelBoldt, AySz88, Az1568, AzaToth, Azuredu, BJjake, Baccyak4H, Baprameya, Barak Sh, Bcrowell, Bdesham, Bdmy, Ben b, BenFrantzDale, Bethnim, Bethpage89, Bhxho, BiT, Binary TSO, Blaze2010, Bo Jacoby, Bobianite, Bobo192, Bomac, Boothy443, Brian Tvedt, Bryan Derksen, Bsadowski1, Bsmntbombdood, Buckyboy314, Burk, Buttseckks, CBM, CSTAR, Caesura, Calabe1992, CanadianLinuxUser, Canthusus, Cataclysm12, Catgut, Celestianpower, Cenarium, Ceyockey, ChP94, Charles Matthews, Charvest, Cheeser1, Chenyu, ChrKrabbe, Chrishmt0423, Chrisportelli, Chuck SMITH, Cj005257, Cj67, Cje, Cobaltcigs, Complex (de), Conversion script, Cooolsduudes, Courcelles, Crisófilax, Cronholm144, Cyp, DGaw, DHN, Daniel.Cardenas, Darrel francis, DarrylNester, Den fjättrade ankan, Denisarona, Deor, DerHexer, Dhaluza, Dialecticas, Dino, Dmcq, Dmharvey, Doctormatt, Dojarca, Dolphin51, Dotancohen, DrBob, Drmies, DroEsperanto, Dtm142, DuKot, Duoduoduo, Dysprosia, ENIAC, EconoPhysicist, Elektron, Eman2129, Enigmaman, Enviroboy, Epbr123, Eric119, Erik Sandberg, Error792, Essjay, Estudiarme, Evan2718281828, Everyking, Evil saltine, Evolauxia, Extransit, Ezh, Fazan1983, Fcelikler, Fintor, Foobarnix, Foxandpotatoes, Fredrik, Fresheneesz, Frickylick, Fsiler, GTBacchus, Gabbe, Gabn1, Gagueci, Garrooney, Gary King, Geometry guy, Georgia guy, Gesslein, Giftlite, Gilliam, Ginkgo100, Glacialfox, Glenn, Gmaxwell, Graham87, Greg von greg, Guardian of Light, Gurch, Haham hanuka, Hashar, Headbomb, Heimstern, HenningThielemann, Hephaestos, Heron, Hoolycheeese, Hteen, Iamthedeus, Icairns, Ideyal, If I Am Blocked I Will Cry, InkKznr, Innerproduct, IstvanWolf, Iulianu, Izno, IznoRepeat, JAn Dudík, JB82, JRSpriggs, Jacj, Jackzhp, Jacobolus, JamesBWatson, Janus Shadowsong, Jarble, JeLuF, Jeff G., Jim.belk, Jimothytrotter, Jitse Niesen, Jixzad123, Jketola, Joey-das-WBF, John254, JohnOwens, Johnuniq, Jon Harald Søby, Jonathan48, Josh dos, Jrtayloriv, Jshadias, Jsjsjs1111, KSmrq, Kaimbridge, Karol Langner, Katzmik, Kbolino, Keilana, Keithcc, Kerdek, Keryk, Kevin Saff, King Bee, KleinKlio, KnowledgeOfSelf, Knutux, Kocher2006, Konradek, Korovyev, Kourtney88, Kozuch, Kpennington82, Kukini, Kumioko (renamed), Kurykh, Kwantus, Lethe, Lewallen, Lightmouse, Livius3, Loisel, Loodog, Lorrybizzle314159, Lotje, Ltmboy, Lynxmb, MC10, MStankie, Mac, Mac Davis, Madhero88, Maelin, Magister Mathematicae, MagnaMopus, Mandavi, Mani1, Manulinho72, MarSch, Marc Venot, Marek69, Markjoseph125, Marquez, Masterofpsi, Materialscientist, MatrixFrog, Maurice Carbonaro, Mboverload, Mcorazao, Melchoir, Melongrower, Metacomet, Michael Hardy, Mike2vil, Mindmatrix, MisterSheik, Mo0, Mormegil, Mountain, Mouse20080706, Mpatel, Mphilip1, Mr Ape, MrOllie, Ms2ger, Nahum Reduta, Najoj, Nandesuka, Nayuki, Nbarth, NeonMerlin, Netrapt, Newone, NickBush24, Nickrds09, No Guru, Nobaddude, Noel Bush, Nonagonal Spider, ObserverFromAbove, OlEnglish, Oleg Alexandrov, Omegatron, Onebravemonkey, Orderud, Oscabat, Ozob, Palfrey, Paolo.dL, Pascal.Tesson, Patrick, Paul August, PaulSmith641, Pbroks13, Pcb21, PeteX, Peterwhy, Phgao, Phillip J, Pinkadelica, Pizza Puzzle, Pmanderson, Polx, Populus, Potatoswatter, Poulter7777777, Powowatushtoosha, Programmar, Pseudolus42, Qef, Quadrescence, Quondum, R3m0t, RDBury, RMFan1, Ralian, Ramcdpai, Rdsmith4, Realmestup, RedWolf, Revolver, Rhyshuw1, Ricardo sandoval, Rick Norwood, Ritchy, Rjwilmsi, Robbjedi, Rosasco, Rossami, Rowland606, Rrenner, Ruud Koot, Rx5674, Ryulong, SJP, SQGibbon, Salix alba, Sanu.soysa, Saotomi5102, Saretakis, Scientific29, Scifitechie360, Sergiobgaspar, Shellgirl, Shellreef, Sigmalmtd, Silly rabbit, Simetrical, Simfish, Skizzik, Sky Attacker, SkyMachine, Slakr, Sligocki, SlipperyHippo, Slysplace, Smithpith, Snaxe920, Sonett72, Spartan S58, Srleffler, Starr sam, Stevertigo, Stomme, StradivariusTV, Stuart07, Stw, Sukolsak, SunCreator, Sushant gupta, Svick, Sławomir Biała, Sławomir Biały, THEN WHO WAS PHONE?, TStein, TakuyaMurata, Tammojan, Tcncv, Tcnuk, Template namespace initialisation script, Terminaterjohn, Tewy, Thalesfernandes, Thariath1, The Anome, The Font, The Wordsmith, TheNightFly, TheRanger, TheTito, Theda, Theunixgeek, ThirdParty, Tholme, Thomasmeeks, Tiddly Tom, Timothy Clemans, Timwi, Titoxd, Tkuvho, Tobias Bergemann, Tomaxer, Tosha, Trombe29, Turenar, Turgidson, Ufnoise, Ukexpat, Ukikonno, Unint, Urdutext, User A1, Usuwiki, VKokielov, Vangos, Vanished User 0001, VectorPosse, Vevek, WatermelonPotion, Wavelength, Wik, Wikid77, Wikifixi, Wikithesource, Willking1979, Wisdom89, Wolfmankurd, WriterHound, X!, XP1, Xantharius, Yacht, Yazaq, Yk Yk Yk, Youssefsan, Zack wadghiri, Zchenyu, Zoicon5, Zundark, Περίεργος, 780 anonymous edits

Rate of convergence *Source*: http://en.wikipedia.org/w/index.php?oldid=492085780 *Contributors*: A Real Kaiser, Ahoerstemeier, Alexander Chervov, Asadi1978, Bender235, Brad7777, CRGreathouse, Cellomathen, Charles Matthews, DannyAsher, Dethron, Ezander, Foobaz, Gadykozma, Giftlite, Hkhk59333, J04n, Jdh8, Jean-Charles.Gilbert, Jitse Niesen, Keithh, Linas, LutzL, MFH, Marcisoo, Michael Hardy, Mr.woozie, Oleg Alexandrov, Olegalexandrov, Patrick, Paul August, Salleyhyatt, SunCreator, Ynhockey, 62 anonymous edits

Bisection method *Source*: http://en.wikipedia.org/w/index.php?oldid=494005166 *Contributors*: Alejo2083, Arithmonic, AzaToth, Beetstra, Berland, Betacommand, BigJohnHenry, CBoeckle, CRGreathouse, Calle, CanadianLinuxUser, Cobi, Cybercobra, Danger, Decrease789, Diberri, Email4mobile, Emerson7, Exp3.14, Frap, Giftlite, Glenn L, Glrx, Gombang, Googl, InverseHypercube, J04n, JJL, JabberWok, Jaredwf, Jitse Niesen, Jujutacular, Kangaroopower, Kawautar, Kenny TM~, Kpym, Laurentius, Lhf, LiDaobing, Lozeldafan, Luc4, Lukaskoz, Majromax, MarkSweep, Marshall.Mills, Materialscientist, McKay, Melsaran, Mohan.chanpur, Mthwiki, NeoUrfahraner, Obradovic Goran, Oleg Alexandrov, Olegalexandrov, Patrick, Paul August, RDBury, Rursus, Sango123, Sc920505, Simone.rossi, Sparkl3y, Sturkyn, Xieyihui, Xutu1116, Zhurov, Zundark, 109 anonymous edits

Newton's method *Source*: http://en.wikipedia.org/w/index.php?oldid=492991196 *Contributors*: 123Mike456Winston789, 1exec1, Aaron Rotenberg, AaronSw, Adam majewski, Aero-Plex, AgadaUrbanit, Alias Flood, Annne furanku, Anonymous Dissident, Arithmonic, Aude, Avicennasis, AxelBoldt, Baccyak4H, Bdiscoe, Ben pcc, BenFrantzDale, Berland, BigJohnHenry, Billtubbs, Birge, Blotwell, Bomazi, Borisblue, CBoeckle, CPMartin, CRGreathouse, Capricorn42, Casey Abell, Chiswick Chap, Chris the speller, Chris857, Chrisgolden, Christian75, Ckamas, ClarkSims, CnkALTDS, ConMan, Conversion script, Coredesat, Cyfal, Cyp, DNA Games, David Cooke, Dawn Bard, Dcljr, Dcoetzee, Decrease789, Dekisugi, Dlazesz, Dominus, Draconx, Dreamofthedolphin, Drift chambers, Dulcamara, Duoduoduo, Earlh, EconoPhysicist, Ejrh, Elvek, EmadIV, Epbr123, Eric Le Bigot, Estudiarme, Evil saltine, Eweinber, Exir Kamalabadi, Eyrryds, Eyu100, Fintor, Fluffernutter, Formulax, Frau Holle, Fredrik, Galoisgroupie, Gco, Gex999, Giftlite, Glenn L, Goingstuckey, Gombang, GregorB, GuidoGer, H.ehsaan, Haham hanuka, Headbomb, Henning Makholm, Hike395, Hirzel, Holycow958, Homo logos, Howard nyc, Iamunknown, Illegal604, InverseHypercube, JRSpriggs, JabberWok, Jackzhp, Jagged 85, JamesBWatson, Jaredwf, Jasondet, Jeltz, Jfmantis, Jim.belk, Jimbryho, Jitse Niesen, JohnOwens, Jon Awbrey, Joriki, Jpginn, K.menin, KHamsun, KMic, KSmrq, Katzmik, Kawautar, Kbk, Kiefer.Wolfowitz, KlappCK, Kutulu, Kyle.drerup, LOL, Laug, Laurentius, Lee Daniel Crocker, Loisel, Looxix, LutzL, Magmait, MarkSweep, Marquez, Martin Hedegaard, Metaprimer, Michael Hardy, Miguel, Mild Bill Hiccup, Minesweeper, MinorContributor, Nedunuri, Neilc, Nk, Nonagonal Spider, Nyirenda, Oleg Alexandrov, Olegalexandrov, PDH, Patrick, Paul August, Pbroks13, Philip Trueman, Pichai Asokan, Pit, Pitel, Pizza Puzzle, PlantTrees, Point-set topologist, Poor Yorick, Protonk, Psymun747, Pt, Quibik, RDBury, RMFan1, Redmarkviolinist, Roadrunner, Robert K S, Rrburke, Rs2, Sam Hocevar, Saravask, Scuchina, Shadowjams, Shreevatsa, Shuipzv3, Shultzc, Skizzik, Slaniel, Smarchesini, Some jerk on the Internet, Suffusion of Yellow, Swerty, Tbsmith, Tesspub, TheJesterLaugh, TheObtuseAngleOfDoom, Tide rolls, Titolatif, Tkuvho, Tom Lougheed, Tomisti, Torokun, User A1, Vrenator, White Shadows, Wikipedia is Nazism, Wysprgr2005, Xieyihui, Xooll, Zundark, Пика Пика, 306 anonymous edits

Householder's method *Source*: http://en.wikipedia.org/w/index.php?oldid=497298740 *Contributors*: Creatorlarryli, Gandalf61, Gauge00, Giftlite, Johnuniq, Kaimbridge, Krishnavedala, Lateef Kareem Adewale, Loisel, LutzL, Mild Bill Hiccup, Paul Carpenter, R.e.b., RDBury, Savonneux, Slawekb, Thumperward, 6 anonymous edits

Halley's method *Source*: http://en.wikipedia.org/w/index.php?oldid=456216384 *Contributors*: BenFrantzDale, BigJohnHenry, CRGreathouse, Fintor, Giftlite, Hair Commodore, Isheden, JRSpriggs, Jitse Niesen, Lachaume, LutzL, Michael Hardy, Mild Bill Hiccup, Mmeijeri, Pjacklam, Rich Farmbrough, Ronhoogwater, Stephen B Streater, Tgmachina, 10 anonymous edits

Secant method *Source*: http://en.wikipedia.org/w/index.php?oldid=497406842 *Contributors*: Adriaan Joubert, Alone Coder, Beetstra, BenFrantzDale, Berland, BigJohnHenry, Decrease789, Decrypt3, Don Quixote de la Mancha, Epbr123, Fintor, Gco, Giftlite, Gombang, Helios Entity 2, Hongooi, Isheden, Jaredwf, Jitse Niesen, Joriki, Jujutacular, Kawautar, Kenny TM~, Kuszi, Laurentius, LutzL, Magioladitis, Marino-slo, Michael Hardy, Monguin61, Obradovic Goran, Oleg Alexandrov, PV=nRT, PeterBFZ, PhysicsPat, Plasticup, Polluxonis, Qef, Qwfp, Rror, Scuchina, Slaniel, Smarchesini, Stlrams22, Visor, Vividendsends, Wdvorak, X7q, 43 anonymous edits

Secant line *Source*: http://en.wikipedia.org/w/index.php?oldid=497369294 *Contributors*: A. Parrot, Abdullah795, Almit39, Angus Lepper, Anonymous Dissident, Atomicthumbs, Candleknight, Charles Matthews, Chewings72, Corvus coronoides, Crazy Boris with a red beard, Dantheox, Dravecky, Equendil, Evil saltine, FlamingSilmaril, Gareth Griffith-Jones, Giftlite, Glenn, Gmadey, Hajatvrc, Hellbus, KDesk, Kaimbridge, Kiensvay, Kinu, La goutte de pluie, Lantonov, Letsbefiends, Luiskillers, Lzur, MIT Trekkie, Mandarax, MarSch, Michael Hardy, Mikiemike, Minesweeper, OhanaUnited, Ojs, Oliver Pereira, PV=nRT, Penubag, PhattyFatt, Pizza Puzzle, Qetuth, Rejnal, Rjwilmsi, Rogper, Rojomoke, S3ndal3, Sayginer, Scientific29, Srbauer, Stevertigo, Sverdrup, Tom harrison, Yasirnaeem, 56 anonymous edits

Broyden's method *Source*: http://en.wikipedia.org/w/index.php?oldid=493891108 *Contributors*: Andrewhayes, BigJohnHenry, Calleman21, Cobi, Domination989, Dtrebbien, Haseldon, Imjustmatthew, Jitse Niesen, Lavaka, NathanHagen, Neuralwarp, Oleg Alexandrov, Paulnwatts, Rjwilmsi, Smarchesini, Timena dava, Tmonzenet, Wikielwikingo, 30 anonymous edits

False position method *Source*: http://en.wikipedia.org/w/index.php?oldid=496788830 *Contributors*: A. Pichler, Alhcnbzx, Army1987, BigJohnHenry, Bubba73, David Eppstein, Decrease789, Deeptrivia, Ethan Mitchell, Gombang, JRSpriggs, Jagged 85, Jaredwf, Jeroendv, Jitse Niesen, Johnuniq, Marino-slo, Matt Cook, Mike Jones, MikeLynch, Milogardner, Munnijandu, Oleg Alexandrov, Olegalexandrov, Petter Per, Qwfp, Reallynca, Rks22, Roylee, Rursus, Shadowjams, The Anome, Thetwentyone, Tom Lougheed, Vishahu, Wiqi55, Wizzy, 54 anonymous edits

Ridders' method *Source*: http://en.wikipedia.org/w/index.php?oldid=491302456 *Contributors*: Coppertwig, Jitse Niesen, Jujutacular, Nick Number, Qwfp, Recon Etc, 6 anonymous edits

Linear interpolation *Source*: http://en.wikipedia.org/w/index.php?oldid=488890305 *Contributors*: AlanUS, Ap, Berland, Blotwell, ChrisGualtieri, Cshimmin, Dcoetzee, Elwikipedista, Fgnievinski, Fredrik, Giftlite, Gxvmjz, Hmains, Jagged 85, Jitse Niesen, Jokestress, Jynus, MER-C, Maurog, Michael Hardy, Mike s, Neffk, Nixdorf, Nunoplopes, Party, Paul Martinsen, PenguiN42, Pjvpjv, Poor Yorick, Rjwilmsi, Sterrys, The Anome, Wykypydya, 72 anonymous edits

Polynomial interpolation *Source*: http://en.wikipedia.org/w/index.php?oldid=477496869 *Contributors*: Alai, Andris, Arrizaba, Astrodogdog, Autarch, Balabiot, Berland, Btyner, CatherineMunro, Charles Matthews, DavidCary, Dmcq, Doccolinni, Donarreiskoffer, Dues Ex Machina, Elwikipedista, Fgnievinski, Giftlite, Glrx, Idont Havaname, Igor Kuchmienko, Jitse Niesen, Jni, JohnBlackburne, Julien Tuerlinckx, KlappCK, Lambiam, Lockeownzj00, Marco4math, Mariana Briggs, MathFacts, MathMartin, Mazemaster, Michael Hardy, Mks004, Myasuda, Nbarth, Nixdorf, Nomatter01, Oleg Alexandrov, PMajer, Pedrito, PedroPVZ, Pentti71, Pleasantville, Qiqi.wang, RDBury, Rhys, Robin S, Snaily, Snaxe920, Sterrys, StuRat, SunCreator, Svick, Sławomir Biały, Tbackstr, The Anome, Tlroche, Tobias Bergemann, WojciechSwiderski, Yugsdrawkcabeht, Zoicon5, Zundark, 62 anonymous edits

Muller's method *Source*: http://en.wikipedia.org/w/index.php?oldid=479833889 *Contributors*: 127, BigJohnHenry, Charles Matthews, Decrease789, Favonian, Frencheigh, Gene Nygaard, JRSpriggs, Jacob grace, Jitse Niesen, Oleg Alexandrov, Qwfp, 17 anonymous edits

Inverse quadratic interpolation *Source*: http://en.wikipedia.org/w/index.php?oldid=492093533 *Contributors*: Btyner, Decrease789, Favonian, Jitse Niesen, Ludovic89, Michael Hardy, Oleg Alexandrov, Pegasusayanami, Thetwentyone, Trialinone, 5 anonymous edits

Brent's method *Source*: http://en.wikipedia.org/w/index.php?oldid=491995315 *Contributors*: Agro r, BigJohnHenry, CRGreathouse, Coffee2theorems, Decrease789, Dhatfield, Dmoggles, DoriSmith, DougAJ4, EdH, Erwan.lemonnier, JRSpriggs, JesseGarrett, Jitse Niesen, Jrvz, Laug, Luis Sanchez, Meungkim, Mild Bill Hiccup, Nrbelex, Oleg Alexandrov, Pashevus, Pcheah, Pma jones, Qwfp, Rjwilmsi, Simoneau, The Anome, Treutwein, Vpapanik, Withlifecomesgoals, Yangli2, Zappa711, 30 anonymous edits

Horner's method *Source*: http://en.wikipedia.org/w/index.php?oldid=498393181 *Contributors*: AdjustShift, AlanUS, Alexandre Vassalotti, Ali Esfandiari, Alink, AxelBoldt, Badpazzword, Balabiot, BigJohnHenry, Bo Jacoby, CESSMASTER, CRGreathouse, Caholman, Charles Matthews, Conversion script, Cyan, DVdm, DavidCBryant, Dcljr, Dcoetzee, Dekart, Derek farn, Dicklyon, Dirtyliberal, Dlrohrer2003, Dmcq, Dominus, Email4mobile, Fredrik, Giftlite, Gisling, Headbomb, Hephaestos, JFB80, Jagged 85, JakeVortex, Jitse Niesen, Jp.martin-flatin, LOL, Lambiam, Laser brain, Lavaka, Liftarn, Lockeownzj00, Logic master, Loisel, LutzL, MathMartin, Michael Hardy, Miguel, Mikiemike, Misantropo, Mrrmv, NathanHurst, Numbo3, Oleg Alexandrov, Patrickriendeau, Pegua, PericlesofAthens, Peter Karlsen, Philten, Pitel, Quadrescence, Ryan Reich, Sam Hocevar, Sebastianvattamattam, Shreevatsa, Snoopy67, Tbhotch, Tdoune, Tom harrison, Tommy2010, Wdrev, WojciechSwiderski, Xjwiki, Xrjunque, 64 anonymous edits

Sturm's theorem *Source*: http://en.wikipedia.org/w/index.php?oldid=495803801 *Contributors*: A. Pichler, Actam, Akritas2, Akruppa, Asmeurer, Baccala@freesoft.org, Bduke, Bender235, BeteNoir, Brad7777, CRGreathouse, Catslash, Charles Matthews, Chris the speller, CàlculIntegral, D.Lazard, DemiHD, Eliko, EmilJ, Giftlite, Glacialfox, Hurkyl, Icairns, JadeNB, JakeVortex, JoshuaZ, Jujutacular, Julien Tuerlinckx, KSmrq, Kidnamedlox, Lourakis, Luther Tychonievich, Macrakis, MathMartin, Maxal, Michael Hardy, NathanHurst, Oleg Alexandrov, PV=nRT, Phe, Prohlep, R. J. Mathar, RDBury, Rschwieb, Salix alba, Securiger, Slawekb, Tabletop, Tbackstr, Tbsmith, Vroo, 22 anonymous edits

Vincent's theorem *Source*: http://en.wikipedia.org/w/index.php?oldid=494436203 *Contributors*: Akritas2, Drbreznjev, Michael Hardy, Surferbuffer, Xro7, 15 anonymous edits

Budan's theorem *Source*: http://en.wikipedia.org/w/index.php?oldid=494178729 *Contributors*: A412, Akritas2, Bearcat, Catslash, Doctree, Drbreznjev, EricEnfermero, Malcolma, Michael Hardy, Rschwieb, Stausifr, Surferbuffer, Woohookitty, Xro7, 46 anonymous edits

Jenkins–Traub algorithm *Source*: http://en.wikipedia.org/w/index.php?oldid=466240722 *Contributors*: Apapageorgiou, Cardamon, Charles Matthews, David Eppstein, EoGuy, Gareth McCaughan, Giftlite, Hvestermark, Ideogram, LilHelpa, LutzL, Macrakis, Michael Hardy, Mild Bill Hiccup, RobertM52, 14 anonymous edits

Laguerre's method *Source*: http://en.wikipedia.org/w/index.php?oldid=484628775 *Contributors*: 81120906713, Andri Egilsson, Balabam, Charles Matthews, DavidCreswick, Decrease789, Destynova, EoGuy, Ezh, Giftlite, Greatdiwei, JamesBWatson, Jeltz, Jitse Niesen, LutzL, Oleg Alexandrov, Olegalexandrov, Oli Filth, PMLawrence, Rainwarrior, Samuel Huang, Stormwyrm, Wikielwikingo, Ylai, 8 anonymous edits

Bairstow's method *Source*: http://en.wikipedia.org/w/index.php?oldid=493851264 *Contributors*: Archangelo, Chuunen Baka, Decrease789, Giftlite, Hermitian, Ironholds, Jitse Niesen, Lim Wei Quan, LutzL, Mathman10, Oleg Alexandrov, Rjstott, 17 anonymous edits

Durand–Kerner method *Source*: http://en.wikipedia.org/w/index.php?oldid=482634639 *Contributors*: Berland, Bo Jacoby, ChrisCork, Decrease789, Fredrik, Giftlite, JMatthews, Jitse Niesen, Jmath666, Kelly Martin, LutzL, Michael Hardy, Myasuda, OdedSchramm, Oleg Alexandrov, Rich Farmbrough, Rjwilmsi, Tom Lougheed, Xrjunque, 4 anonymous edits

Aberth method *Source*: http://en.wikipedia.org/w/index.php?oldid=473888545 *Contributors*: Adam majewski, Constructive editor, Decrease789, Giftlite, JRSpriggs, LutzL, Magioladitis, Michael Hardy, Oleg Alexandrov, Rjwilmsi, 5 anonymous edits

Splitting circle method *Source*: http://en.wikipedia.org/w/index.php?oldid=487802344 *Contributors*: CRGreathouse, ChrisCork, Chuunen Baka, Decrease789, Fredrik, Jmath666, LutzL, Michael Hardy, Oleg Alexandrov, Rjwilmsi, 5 anonymous edits

Graeffe's method *Source*: http://en.wikipedia.org/w/index.php?oldid=496267222 *Contributors*: BigJohnHenry, Bullzeye, Burn, CRGreathouse, Decrease789, Giftlite, JRSpriggs, Linas, LutzL, Michael Hardy, Mild Bill Hiccup, Mmoore2, NathanHurst, Oleg Alexandrov, 8 anonymous edits

Greatest common divisor *Source*: http://en.wikipedia.org/w/index.php?oldid=497883904 *Contributors*: 8.128, Aconcagua, Allens, AmigoNico, Anita5192, Armend, Arthena, Arthur Rubin, AvicAWB, AxelBoldt, Azuredu, Bemoeial, Bender235, Blankfaze, Bryan Derksen, Bryan H Bell, Captain panda, Carey Evans, Charles Matthews, Closenplay, ComputScientist, Conversion script, Crazy Boris with a red beard, Cwinstanley, Cybercobra, Dantheox, Dav!dB, Dcoetzee, Decltype, DixonD, Dysprosia, Ero lupin 3, Espetkov, EyeRmonkey, Fredrik, GeneralHooHa, Genghiskhanviet, Geregen2, Gesslein, Giftlite, Glenn L, Gogobera, Guno1618, Haham hanuka, Hannes Hirzel, Henrygb, Herbee, Howard nyc, Hu12, HupHollandHup, Hyad, II MusLiM HyBRiD II, Ichudov, Ideyal, Indwatch, JCipriani, JamesBWatson, Jbergquist, Jiddisch, Jitse Niesen, Jleedev, Jmr, Joaoferreira, JorgeGG, Josh Parris, Karl Dickman, Kenyon, Kevin Ryde, Knakts, KneeLess, KnowledgeOfSelf, Krslavin, LOL, Lambiam, Linas, Little Mountain 5, LutzL, MC-CPO, MacMed, Marc van Leeuwen, Matiasholte, Michael Hardy, Michael M Clarke, Mifter, Mild Bill Hiccup, Mohammadrdeh, Moreschi, Mormegil, Mrjoerizkallah, Natalie Erin, Nayuki, NeMewSys, Nehuse, Nikai, Normanzhang, Obradovic Goran, Oleg Alexandrov, Orion11M87, Palnatoke, Patstuart, Paul August, Pesit, Planetscape, Pokipsy76, Poor Yorick, Qwertyus, Revolver, Ridhi bookworm, Rjwilmsi, Ruud Koot, SLi, Salgueiro, Salvatore Ingala, Shanefb, Shirik, Shoeofdeath, Silver hr, SirJective, Sniper 95 jonas, Sopoforic, Spnashville, Spoon!, Staecker, TakuyaMurata, TangentCube, Tarquin, Taw, Tmvphil, Tom harrison, TomyDuby, Tosha, Turgidson, VKokielov, Ves123, Vladkornea, Vssun, Wabbit98, WikiTome, XJamRastafire, Zundark, Περίεργος, రహీంఠదోర్, 277 anonymous edits

Image Sources, Licenses and Contributors

File:Tangent to a curve.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Tangent_to_a_curve.svg *License*: Public Domain *Contributors*: Original uploader was Jacj at en.wikipedia Later versions were uploaded by Oleg Alexandrov at en.wikipedia.

File:Graph of sliding derivative line.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:Graph_of_sliding_derivative_line.gif *License*: Public Domain *Contributors*: en:User:Dino

Image:Tangent-calculus.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Tangent-calculus.svg *License*: GNU Free Documentation License *Contributors*: derivative work: Pbroks13 (talk) Tangent-calculus.png: Rhythm

Image:Secant-calculus.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Secant-calculus.svg *License*: GNU Free Documentation License *Contributors*: derivative work: Pbroks13 (talk) Secant-calculus.png: Shizhao

Image:Lim-secant.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Lim-secant.svg *License*: GNU Free Documentation License *Contributors*: derivative work: Pbroks13 (talk) Lim-secant.png: Original uploader was CSTAR at en.wikipedia

File:Right-continuous.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Right-continuous.svg *License*: Public Domain *Contributors*: Jacj

File:Absolute value.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Absolute_value.svg *License*: GNU Free Documentation License *Contributors*: This hand-written SVG version by Qef Original bitmap version Image:Absolute_value.png by Ævar Arnfjörð Bjarmason

Image:ConvergencePlots.png *Source*: http://en.wikipedia.org/w/index.php?title=File:ConvergencePlots.png *License*: Creative Commons Attribution-Share Alike *Contributors*: LutzL, Mr.woozie

Image:Bisection method.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Bisection_method.svg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Bisection_method.svg: Tokuchan derivative work: Tokuchan (talk)

Image:NewtonIteration Ani.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:NewtonIteration_Ani.gif *License*: GNU Free Documentation License *Contributors*: Ralf Pfeifer

Image:NewtonsMethodConvergenceFailure.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:NewtonsMethodConvergenceFailure.svg *License*: Public Domain *Contributors*: Aaron Rotenberg

image:newtroot 1 0 0 0 0 m1.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Newtroot_1_0_0_0_0_m1.png *License*: GNU General Public License *Contributors*: D-Kuru, Dbc334, LutzL, Maksim, Norro

Image:Secant method.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Secant_method.svg *License*: Public Domain *Contributors*: User:Jitse Niesen

Image:Secant method example code result.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Secant_method_example_code_result.svg *License*: Public Domain *Contributors*: Qef

Image:CIRCLE LINES.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:CIRCLE_LINES.svg *License*: GNU Free Documentation License *Contributors*: User:Jleedev

Image:False position method.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:False_position_method.svg *License*: Public Domain *Contributors*: User:Jitse Niesen

Image:LinearInterpolation.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:LinearInterpolation.svg *License*: Public Domain *Contributors*: Berland. Based on png-version Image:Linear_interpolation.png.

Image:Interpolation example linear.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Interpolation_example_linear.svg *License*: Public Domain *Contributors*: Berland

Image:Bilininterp.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Bilininterp.png *License*: Public Domain *Contributors*: Berland, WikipediaMaster

Image:Piecewise linear function2D.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Piecewise_linear_function2D.svg *License*: Public Domain *Contributors*: Oleg Alexandrov

Image:Interpolation example polynomial.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Interpolation_example_polynomial.svg *License*: Public Domain *Contributors*: Berland

Image:Brent method example.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Brent_method_example.png *License*: Public Domain *Contributors*: User:Jitse Niesen

Image:HornerScheme.gif *Source*: http://en.wikipedia.org/w/index.php?title=File:HornerScheme.gif *License*: Public Domain *Contributors*: Philten

File:Qingjiushaoquad1.GIF *Source*: http://en.wikipedia.org/w/index.php?title=File:Qingjiushaoquad1.GIF *License*: Creative Commons Attribution-Sharealike 3.0,2.5,2.0,1.0 *Contributors*: Gisling

Image:Nuvola apps edu mathematics.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Nuvola_apps_edu_mathematics.png *License*: unknown *Contributors*: Alno, Alphax, Naconkantari, Rfc1394, Rocket000, Zundark, 1 anonymous edits

File:Sketch_of_proof.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Sketch_of_proof.jpg *License*: GNU Free Documentation License *Contributors*: Surferbuffer

File:Vincent_method.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Vincent_method.jpg *License*: GNU Free Documentation License *Contributors*: Surferbuffer

File:VAS_method_example.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:VAS_method_example.jpg *License*: GNU Free Documentation License *Contributors*: Surferbuffer

File:VCA_Algorithm.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:VCA_Algorithm.jpg *License*: GNU Free Documentation License *Contributors*: Surferbuffer

File:VCA_Example.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:VCA_Example.jpg *License*: GNU Free Documentation License *Contributors*: Surferbuffer

File:Uspensky_attempt.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Uspensky_attempt.jpg *License*: GNU Free Documentation License *Contributors*: Surferbuffer

File:Bairstow-fractal_1_0_0_0_0_m1_scale_03.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Bairstow-fractal_1_0_0_0_0_m1_scale_03.png *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: LutzL

File:Bairstow-fractal_1_0_0_0_0_m1_0_scale_3.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Bairstow-fractal_1_0_0_0_0_m1_0_scale_3.png *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: LutzL

File:Bairstow-fractal_6_11_m33_m33_11_6_scale_03.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Bairstow-fractal_6_11_m33_m33_11_6_scale_03.png *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Lutz Lehmann User:LutzL

File:24x60.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:24x60.svg *License*: Creative Commons Zero *Contributors*: Michael Hardy

File:least common multiple.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Least_common_multiple.svg *License*: GNU Free Documentation License *Contributors*: Morn

License