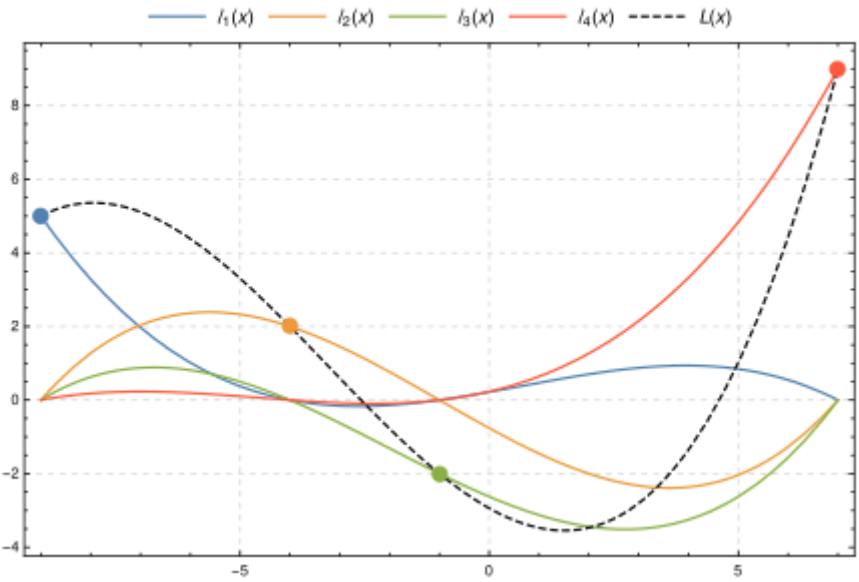
Lagrange polynomial

From Wikipedia, the free encyclopedia

In numerical analysis, **Lagrange polynomials** are used for polynomial interpolation. For a given set of distinct points x_j and numbers y_j, the **Lagrange polynomial** is the polynomial of lowest degree that assumes at each point x_j the corresponding value y_j (i.e. the functions coincide at each point). The interpolating polynomial of the least degree is unique, however, and since it can be arrived at through multiple methods, referring to "the Lagrange polynomial" is perhaps not as correct as referring to "the Lagrange form" of that unique polynomial.



This image shows, for four points (−9, 5), (−4, 2), (−1, −2), (7, 9)), the (cubic) interpolation polynomial $L(x)$ (dashed, black), which is the sum of the *scaled* basis polynomials $y_0\ell_0(x)$, $y_1\ell_1(x)$, $y_2\ell_2(x)$ and $y_3\ell_3(x)$. The interpolation polynomial passes through all four control points, and each *scaled* basis polynomial passes through its respective control point and is 0 where x corresponds to the other three control points.

Although named after Joseph Louis Lagrange, who published it in 1795, the method was first discovered in 1779 by Edward Waring. It is also an easy consequence of a formula published in 1783 by Leonhard Euler.[1]

Uses of Lagrange polynomials include the Newton–Cotes method of numerical integration and Shamir's secret sharing scheme in cryptography.

Lagrange interpolation is susceptible to Runge's phenomenon of large oscillation. And changing the points x_j requires recalculating the entire interpolant, so it is often easier to use Newton polynomials instead.

Contents

Definition

Given a set of $k + 1$ data points

$$(x_0, y_0), \ldots, (x_j, y_j), \ldots, (x_k, y_k)$$

where no two x_j are the same, the **interpolation polynomial in the Lagrange form** is a linear combination

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x)$$

of Lagrange basis polynomials

$$\ell_j(x) := \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)},$$

where $0 \le j \le k$. Note how, given the initial assumption that no two x_i are the same, $x_j - x_m \ne 0$, so this expression is always well-defined. The reason pairs $x_i = x_j$ with $y_i \ne y_j$ are not allowed is that no interpolation function L such that $y_i = L(x_i)$ would exist; a function can only get one value for each argument x_i. On the other hand, if also $y_i = y_j$, then those two points would actually be one single point.

For all $i \ne j$, $\ell_j(x)$ includes the term $(x - x_i)$ in the numerator, so the whole product will be zero at $x = x_i$:

$$\ell_{j \ne i}(x_i) = \prod_{m \ne j} \frac{x_i - x_m}{x_j - x_m} = \frac{(x_i - x_0)}{(x_j - x_0)} \cdots \frac{(x_i - x_i)}{(x_j - x_i)} \cdots \frac{(x_i - x_k)}{(x_j - x_k)} = 0.$$

On the other hand,

$$\ell_i(x_i) := \prod_{m \ne i} \frac{x_i - x_m}{x_i - x_m} = 1$$

In other words, all basis polynomials are zero at $x = x_i$, except $\ell_i(x)$, for which it holds that $\ell_i(x_i) = 1$, because it lacks the $(x - x_i)$ term.

It follows that $y_i \ell_i(x_i) = y_i$, so at each point x_i, $L(x_i) = y_i + 0 + 0 + \cdots + 0 = y_i$, showing that L interpolates the function exactly.

Proof

The function $L(x)$ being sought is a polynomial in x of the least degree that interpolates the given data set; that is, assumes value y_j at the corresponding x_j for all data points j:

$$L(x_j) = y_j \qquad j = 0, \ldots, k$$

Observe that:

1. In $\ell_j(x)$ there are k factors in the product and each factor contains one x, so $L(x)$ (which is a sum of these k-degree polynomials) must also be a k-degree polynomial.

2. $\ell_j(x_i) = \prod_{m=0,\, m\neq j}^{k} \frac{x_i - x_m}{x_j - x_m}$

We consider what happens when this product is expanded. Because the product skips $m = j$, if $i = j$ then all terms are $\frac{x_j - x_m}{x_j - x_m} = 1$ (except where $x_j = x_m$, but that case is impossible, as pointed out in the definition section—in that term, $m = j$, and since $m \neq j$, $i \neq j$, contrary to $i = j$). Also if $i \neq j$ then since $m \neq j$ does not preclude it, one term in the product **will** be for $m = i$, i.e. $\frac{x_i - x_i}{x_j - x_i} = 0$, zeroing the entire product. So

1. $\ell_j(x_i) = \delta_{ji} = \begin{cases} 1, & \text{if } j = i \\ 0, & \text{if } j \neq i \end{cases}$

where δ_{ij} is the Kronecker delta. So:

$$L(x_i) = \sum_{j=0}^{k} y_j \ell_j(x_i) = \sum_{j=0}^{k} y_j \delta_{ji} = y_i.$$

Thus the function $L(x)$ is a polynomial with degree at most k and where $L(x_i) = y_i$.

Additionally, the interpolating polynomial is unique, as shown by the unisolvence theorem at the polynomial interpolation article.

A perspective from linear algebra

Solving an interpolation problem leads to a problem in linear algebra amounting to inversion of a matrix. Using a standard monomial basis for our interpolation polynomial $L(x) = \sum_{j=0}^{k} x^j m_j$, we must invert the Vandermonde matrix $(x_i)^j$ to solve $L(x_i) = y_i$ for the coefficients m_j of $L(x)$. By choosing a better basis, the Lagrange basis, $L(x) = \sum_{j=0}^{k} l_j(x) y_j$, we merely get the identity matrix, δ_{ij}, which is its own inverse: the Lagrange basis automatically *inverts* the analog of the Vandermonde matrix.

This construction is analogous to the Chinese Remainder Theorem. Instead of checking for remainders of integers modulo prime numbers, we are checking for remainders of polynomials when divided by linears.

Examples

Example 1

We wish to interpolate $f(x) = x^2$ over the range $1 \le x \le 3$, given these three points:

$$\begin{aligned} x_0 &= 1 & f(x_0) &= 1 \\ x_1 &= 2 & f(x_1) &= 4 \\ x_2 &= 3 & f(x_2) &= 9. \end{aligned}$$

The interpolating polynomial is:

$$L(x) = 1 \cdot \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} + 4 \cdot \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} + 9 \cdot \frac{x-1}{3-1} \cdot \frac{x-2}{3-2}$$
$$= x^2.$$

Example 2

We wish to interpolate $f(x) = x^3$ over the range $1 \le x \le 3$, given these three points:

$$x_0 = 1 \quad f(x_0) = 1$$

$$x_1 = 2 \quad f(x_1) = 8$$

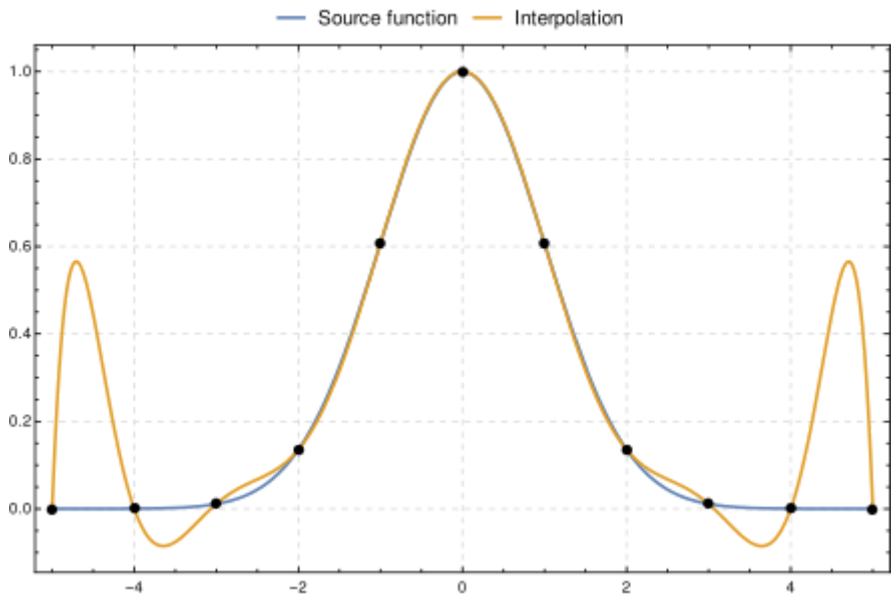
$$x_2 = 3 \quad f(x_2) = 27$$

The interpolating polynomial is:

$$L(x) = 1 \cdot \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} + 8 \cdot \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} + 27 \cdot \frac{x-1}{3-1} \cdot \frac{x-2}{3-2}$$
$$= 6x^2 - 11x + 6.$$

Notes

The Lagrange form of the interpolation polynomial shows the linear character of polynomial interpolation and the uniqueness of the interpolation polynomial. Therefore, it is preferred in proofs and theoretical arguments. Uniqueness can also be seen from the invertibility of the Vandermonde matrix, due to the non-vanishing of the Vandermonde determinant.



Example of interpolation divergence for a set of Lagrange polynomials.

But, as can be seen from the construction, each time a node x_k changes, all Lagrange basis polynomials have to be recalculated. A better form of the interpolation polynomial for practical (or computational) purposes is the barycentric form of the Lagrange interpolation (see below) or Newton polynomials.

Lagrange and other interpolation at equally spaced points, as in the example above, yield a polynomial oscillating above and below the true function. This behaviour tends to grow with the number of points, leading to a divergence known as Runge's phenomenon; the problem may be eliminated by choosing interpolation points at Chebyshev nodes.[2]

The Lagrange basis polynomials can be used in numerical integration to derive the Newton–Cotes formulas.

Barycentric form

Using

$$\ell(x) = (x - x_0)(x - x_1)\cdots(x - x_k)$$
$$\ell'(x_j) = \frac{\mathrm{d}\ell(x)}{\mathrm{d}x}\Big|_{x=x_j} = \prod_{i=0, i\neq j}^{k} (x_j - x_i)$$

we can rewrite the Lagrange basis polynomials as

$$\ell_j(x) = \frac{\ell(x)}{\ell'(x_j)(x - x_j)}$$

or, by defining the *barycentric weights*[3]

$$w_j = \frac{1}{\ell'(x_j)}$$

we can simply write

$$\ell_j(x) = \ell(x)\frac{w_j}{x - x_j}$$

which is commonly referred to as the *first form* of the barycentric interpolation formula.

The advantage of this representation is that the interpolation polynomial may now be evaluated as

$$L(x) = \ell(x)\sum_{j=0}^{k} \frac{w_j}{x - x_j} y_j$$

which, if the weights w_j have been pre-computed, requires only $\mathcal{O}(n)$ operations (evaluating $\ell(x)$ and the weights $w_j/(x - x_j)$) as opposed to $\mathcal{O}(n^2)$ for evaluating the Lagrange basis polynomials $\ell_j(x)$ individually.

The barycentric interpolation formula can also easily be updated to incorporate a new node x_{k+1} by dividing each of the w_j, $j = 0 \ldots k$ by $(x_j - x_{k+1})$ and constructing the new w_{k+1} as above.

We can further simplify the first form by first considering the barycentric interpolation of the constant function $g(x) \equiv 1$:

$$g(x) = \ell(x)\sum_{j=0}^{k} \frac{w_j}{x - x_j}.$$

Dividing $L(x)$ by $g(x)$ does not modify the interpolation, yet yields

$$L(x) = \frac{\sum_{j=0}^{k} \frac{w_j}{x - x_j} y_j}{\sum_{j=0}^{k} \frac{w_j}{x - x_j}}$$

which is referred to as the *second form* or *true form* of the barycentric interpolation formula. This second form has the advantage that $\ell(x)$ need not be evaluated for each evaluation of $L(x)$.

Remainder in Lagrange interpolation formula

When interpolating a given function f by a polynomial of degree n at the nodes $x_0,...,x_n$ we get the remainder $R(x) = f(x) - L(x)$ which can be expressed as [4]

$$R(x) = f[x_0, \dots, x_n, x]\ell(x) = \ell(x)\frac{f^{n+1}(\xi)}{(n+1)!}, \qquad x_0 < \xi < x_n,$$

where $f[x_0, \dots, x_n, x]$ is the notation for divided differences. Alternatively, the remainder can be expressed as a contour integral in complex domain as

$$R(z) = \frac{\ell(z)}{2\pi i}\int_C \frac{f(t)}{(t-z)(t-z_0)\cdots(t-z_n)}dt = \frac{\ell(z)}{2\pi i}\int_C \frac{f(t)}{(t-z)\ell(t)}dt.$$

The remainder can be bound as

$$|R(x)| \leq \frac{(x_n - x_0)^{n+1}}{(n+1)!}\max_{x_0 \leq \xi \leq x_n} |f^{(n+1)}(\xi)|.$$

First derivative

The first derivative of the lagrange polynomial is given by

$$L'(x) := \sum_{j=0}^{k} y_j \ell'_j(x)$$

where $\ell'_j(x) = \ell_j(x) \cdot \sum_{m=0,\ m\neq j}^{k} \frac{1}{x - x_m}$.

Finite fields

The Lagrange polynomial can also be computed in finite fields. This has applications in cryptography, such as in Shamir's Secret Sharing scheme.

See also

- Neville's algorithm
- Newton form of the interpolation polynomial
- Bernstein form of the interpolation polynomial
- Carlson's theorem
- Lebesgue constant (interpolation)
- The Chebfun system
- Table of Newtonian series
- Frobenius covariant
- Sylvester's formula

References

1. 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 (https://doi.org/10.1

109%2F5.993400).
2. Quarteroni, Alfio; Saleri, Fausto (2003), *Scientific Computing with MATLAB* (https://books.google.com/books?id=fE1W5jsU4zoC&pg=PA66), Texts in computational science and engineering, **2**, Springer, p. 66, ISBN 9783540443636.
3. Jean-Paul Berrut & Lloyd N. Trefethen (2004). "Barycentric Lagrange Interpolation". *SIAM Review*. **46** (3): 501–517. doi:10.1137/S0036144502417715 (https://doi.org/10.1137%2FS0036144502417715).
4. Abramowitz and Stegun, "Handbook of Mathematical Functions," p.878

External links

- Hazewinkel, Michiel, ed. (2001) [1994], "Lagrange interpolation formula" (https://www.encyclopediaofmath.org/index.php?title=p/l057170), *Encyclopedia of Mathematics*, Springer Science+Business Media B.V. / Kluwer Academic Publishers, ISBN 978-1-55608-010-4
- ALGLIB (http://www.alglib.net/interpolation/polynomial.php) has an implementations in C++ / C# / VBA / Pascal.
- GSL (https://www.gnu.org/software/gsl/) has a polynomial interpolation code in C
- SO (https://stackoverflow.com/questions/11029615/lagrange-interpolation-method/11552763) has a MATLAB example that demonstrates the algorithm and recreates the first image in this article
- Lagrange Method of Interpolation — Notes, PPT, Mathcad, Mathematica, MATLAB, Maple (http://numericalmethods.eng.usf.edu/topics/lagrange_method.html) at Holistic Numerical Methods Institute (http://numericalmethods.eng.usf.edu)
- Lagrange interpolation polynomial (http://www.math-linux.com/spip.php?article71) on www.math-linux.com
- Weisstein, Eric W. "Lagrange Interpolating Polynomial" (http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html). *MathWorld*.

- Estimate of the error in Lagrange Polynomial Approximation (https://proofwiki.org/wiki/Lagrange_Polynomial_Approximation) at ProofWiki (https://proofwiki.org/)
- Dynamic Lagrange interpolation with JSXGraph (http://jsxgraph.uni-bayreuth.de/wiki/index.php/Lagrange_interpolation)
- Numerical computing with functions: The Chebfun Project (http://www.maths.ox.ac.uk/chebfun/)
- Excel Worksheet Function for Bicubic Lagrange Interpolation (http://mathformeremortals.wordpress.com/2013/01/15/bicubic-interpolation-excel-worksheet-function/)
- Lagrange polynomials in Python (http://pastebin.com/bNVcQt4x)