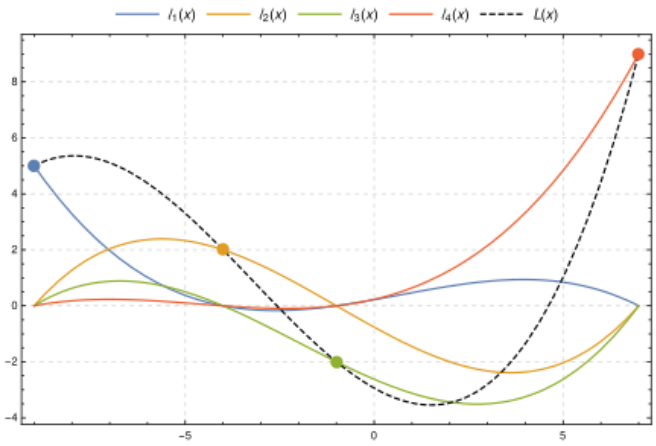
Lagrange polynomial

In numerical analysis, **Lagrange polynomials** are used for polynomial interpolation. For a given set of points (x_j, y_j) with no two x_j values equal, the Lagrange polynomial is the polynomial of lowest degree that assumes at each value x_j the corresponding value y_j.

Although named after Joseph-Louis Lagrange, who published it in 1795, the method was first discovered in 1779 by Edward Waring.[1] It is also an easy consequence of a formula published in 1783 by Leonhard Euler.[2]

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. As changing the points x_j requires recalculating the entire interpolant, it is often easier to use Newton polynomials instead.



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.

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 $0 \leq j \leq k$. Note how, given the initial assumption that no two x_j are the same, then (when $m \neq j$) $x_j - x_m \neq 0$, so this expression is always well-defined. The reason pairs $x_i = x_j$ with $y_i \neq y_j$ 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 $i \neq j$, $\ell_j(x)$ includes the term $(x - x_i)$ in the numerator, so the whole product will be zero at $x = x_i$:

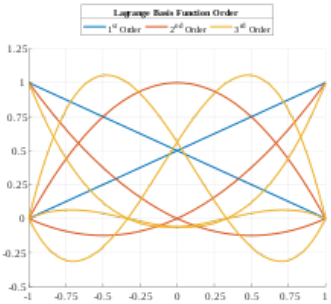
$$\forall (j \neq i) : \ell_j(x_i) = \prod_{m \neq j} \frac{x_i - x_m}{x_j - x_m} = \frac{(x_i - x_0)}{(x_j - x_0)} \cdots \frac{(x_i - x_i)}{(x_j - x_i)} \cdots \frac{(x_i - x_k)}{(x_j - x_k)} = 0.$$

On the other hand,

$$\ell_j(x_j) := \prod_{m \neq j} \frac{x_j - x_m}{x_j - x_m} = 1$$

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

It follows that $y_j \ell_j(x_j) = y_j$, so at each point x_j, $L(x_j) = y_j + 0 + 0 + \cdots + 0 = y_j$, showing that L interpolates the function exactly.



Here we plot the Lagrange basis functions of 1st, 2nd, and 3rd order on a bi-unit domain. Linear combinations of Lagrange basis functions are used to construct Lagrange interpolating polynomials. Lagrange basis functions are commonly used in finite element analysis as the bases for the element shape-functions. Furthermore, it is common to use a bi-unit domain as the natural space for the finite-element's definition.

Proof

The function $L(x)$ being sought is a polynomial in x of the least degree that interpolates the given data set; that is, it assumes the 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:

- 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 be a polynomial of degree at most k.
- $\ell_j(x_i) = \prod_{\substack{m=0 \\ m \neq j}}^{k} \frac{x_i - x_m}{x_j - x_m}$.

Expand this product. Since the product omits the term where $m = j$, if $i = j$ then all terms that appear are $\frac{x_j - x_m}{x_j - x_m} = 1$. Also, if $i \neq j$ then one term in the product *will* be (for $m = i$), $\frac{x_i - x_i}{x_j - x_i} = 0$, zeroing the entire product. So,

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

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.

It's also true that:

$$\sum_{j=0}^{k} \ell_j(x) = 1 \qquad \forall x$$

since it must be a polynomial of degree, at most, k and passes through all these $k + 1$ data points:

$$(x_0, 1), \ldots, (x_j, 1), \ldots, (x_k, 1)$$

resulting in a horizontal line, since a straight line is the only polynomial of degree less than $k + 1$ that passes through $k + 1$ aligned points.

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.

Furthermore, when the order is large, Fast Fourier Transformation can be used to solve for the coefficients of the interpolated polynomial.

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:

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

Example 2

We wish to interpolate $f(x) = x^3$ over the range $1 \le x \le 4$, given these four points:

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

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

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

$$x_3 = 4 \quad f(x_3) = 64$$

The interpolating polynomial is:

$$\begin{aligned} L(x) &= 1 \cdot \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} \cdot \frac{x-4}{1-4} + 8 \cdot \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} \cdot \frac{x-4}{2-4} + 27 \cdot \frac{x-1}{3-1} \cdot \frac{x-2}{3-2} \cdot \frac{x-4}{3-4} + 64 \cdot \frac{x-1}{4-1} \cdot \frac{x-2}{4-2} \cdot \frac{x-3}{4-3} \\ &= x^3 \end{aligned}$$

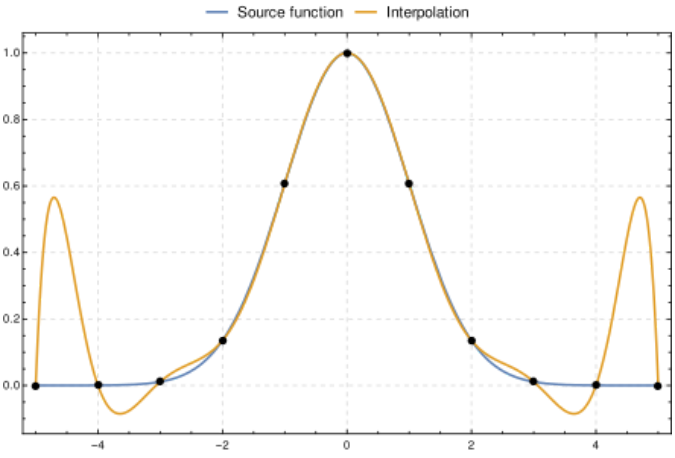
Notes

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.

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.[3]

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



Example of interpolation divergence for a set of Lagrange polynomials.

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*[4]

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

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 $\mathcal{O}(k)$ operations (evaluating $\ell(x)$ and the weights $w_j/(x - x_j)$) as opposed to $\mathcal{O}(k^2)$ 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 k at the nodes $x_0, \ldots, x_k$ we get the remainder $R(x) = f(x) - L(x)$ which can be expressed as[5]

$$R(x) = f[x_0, \ldots, x_k, x]\ell(x) = \ell(x)\frac{f^{(k+1)}(\xi)}{(k+1)!}, \qquad x_0 < \xi < x_k,$$

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

$$R(x) = \frac{\ell(x)}{2\pi i}\int_C \frac{f(t)}{(t-x)(t-x_0)\cdots(t-x_k)}dt = \frac{\ell(x)}{2\pi i}\int_C \frac{f(t)}{(t-x)\ell(t)}dt.$$

The remainder can be bound as

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

Derivation[6]

Clearly, $R(x)$ is zero at nodes. To find $R(x)$ at a point x_p, define a new function $F(x) = R(x) - \tilde{R}(x) = f(x) - L(x) - \tilde{R}(x)$ and choose $\tilde{R}(x) = C \cdot \prod_{i=0}^{k}(x - x_i)$ where C is the constant we are required to determine for a given x_p. We choose C so that $F(x)$ has $k+2$ zeroes (at all nodes and x_p) between x_0 and x_k (including endpoints). Assuming that $f(x)$ is $k+1$-times differentiable, since $L(x)$ and $\tilde{R}(x)$ are polynomials, and therefore, are infinitely differentiable, $F(x)$ will be $k+1$-times differentiable. By Rolle's theorem, $F^{(1)}(x)$ has $k+1$ zeroes, $F^{(2)}(x)$ has k zeroes... $F^{(k+1)}$ has 1 zero, say ξ, $x_0 < \xi < x_k$. Explicitly writing $F^{(k+1)}(\xi)$:

$$F^{(k+1)}(\xi) = f^{(k+1)}(\xi) - L^{(k+1)}(\xi) - \tilde{R}^{(k+1)}(\xi)$$

$L^{(k+1)} = 0, \tilde{R}^{(k+1)} = C \cdot (k+1)!$ (Because the highest power of x in $\tilde{R}(x)$ is $k+1$)

$$0 = f^{(k+1)}(\xi) - C \cdot (k+1)!$$

The equation can be rearranged as

$$C = \frac{f^{(k+1)}(\xi)}{(k+1)!}$$

Since $F(x_p) = 0$ we have $R(x_p) = \tilde{R}(x_p) = \frac{f^{k+1}(\xi)}{(k+1)!}\prod_{i=0}^{k}(x_p - x_i)$

Derivatives

The dth derivatives of the Lagrange polynomial can be written as

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

For the first derivative, the coefficients are given by

$$\ell_j^{(1)}(x) := \sum_{\substack{i=0 \\ i \neq j}}^{k}\left[\frac{1}{x_j - x_i}\prod_{\substack{m=0 \\ m \neq (i,j)}}^{k}\frac{x - x_m}{x_j - x_m}\right]$$

and for the second derivative

$$\ell_j^{(2)}(x) := \sum_{\substack{i=0 \\ i \neq j}}^{k}\frac{1}{x_j - x_i}\left[\sum_{\substack{m=0 \\ m \neq (i,j)}}^{k}\left(\frac{1}{x_j - x_m}\prod_{\substack{l=0 \\ l \neq (i,j,m)}}^{k}\frac{x - x_l}{x_j - x_l}\right)\right].$$

Through recursion, one can compute formulas for higher derivatives.

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 polynomial
- Carlson's theorem
- Lebesgue constant (interpolation)
- The Chebfun system
- Table of Newtonian series
- Frobenius covariant
- Sylvester's formula
- Finite difference coefficient

References

1. Waring, Edward (9 January 1779). "Problems concerning interpolations" (https://doi.org/10.1098%2Frstl.1779.0008). *Philosophical Transactions of the Royal Society*. **69**: 59–67. doi:10.1098/rstl.1779.0008 (https://doi.org/10.1098%2Frstl.1779.0008).
2. Meijering, Erik (2002). "A chronology of interpolation: from ancient astronomy to modern signal and image processing" (http://bigwww.epfl.ch/publications/meijering0201.pdf) (PDF). *Proceedings of the IEEE*. **90** (3): 319–342. doi:10.1109/5.993400 (https://doi.org/10.1109%2F5.993400).
3. 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 978-3-540-44363-6..
4. Berrut, Jean-Paul; Trefethen, Lloyd N. (2004). "Barycentric Lagrange Interpolation" (https://people.maths.ox.ac.uk/trefethen/barycentric.pdf) (PDF). *SIAM Review*. **46** (3): 501–517. doi:10.1137/S0036144502417715 (https://doi.org/10.1137%2FS0036144502417715).
5. Abramowitz, Milton; Stegun, Irene Ann, eds. (1983) [June 1964]. "Chapter 25, eqn 25.2.3" (http://www.math.ubc.ca/~cbm/aands/page_878.htm). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Applied Mathematics Series. **55** (Ninth reprint with additional corrections of tenth original printing with corrections (December 1972); first ed.). Washington D.C.; New York: United States Department of Commerce, National Bureau of Standards; Dover Publications. p. 878. ISBN 978-0-486-61272-0. LCCN 64-60036 (https://lccn.loc.gov/64-60036). MR 0167642 (https://www.ams.org/mathscinet-getitem?mr=0167642). LCCN 65-12253 (https://lccn.loc.gov/65012253).
6. "Interpolation" (https://sam.nitk.ac.in/sites/default/Numerical_Methods/Interpolation/interpolation.pdf) (PDF).

External links

- "Lagrange interpolation formula" (https://www.encyclopediaofmath.org/index.php?title=Lagrange_interpolation_formula), *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]
- 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" (https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html). *MathWorld*.
- Lagrange polynomial at ProofWiki
- Dynamic Lagrange interpolation with JSXGraph (http://jsxgraph.uni-bayreuth.de/wiki/index.php/Lagrange_interpolation)
- Numerical computing with functions: The Chebfun Project (https://web.archive.org/web/20101013180326/http://www2.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)