

Lagrange Interpolation: Modified and Barycentric Formula

1. Polynomial interpolation

Let \mathcal{P}_k denote the polynomials of degree at most k :

$$\mathcal{P}_k := \{a_0 + a_1x + \cdots + a_kx^k \mid a_0, \dots, a_k \in \mathbb{R}\}.$$

For n distinct nodes $x_1, \dots, x_n \in \mathbb{R}$ and given values $y_1, \dots, y_n \in \mathbb{R}$ there exists a unique polynomial $p \in \mathcal{P}_{n-1}$ such that

$$p(x_j) = y_j \quad j = 1, \dots, n \quad (1)$$

2. Lagrange interpolation

The so-called Lagrange polynomials

$$\ell_j(x) := \prod_{\substack{k=1, \dots, n \\ k \neq j}} \frac{x - x_k}{x_j - x_k} \quad (2)$$

obviously satisfy $\ell_j \in \mathcal{P}_{n-1}$ and

$$\ell_j(x_k) = \begin{cases} 0 & \text{for } k \neq j \\ 1 & \text{for } k = j \end{cases}$$

Hence we have the **Lagrange interpolation formula**

$$p(x) = \sum_{j=1}^n y_j \ell_j(x) \quad (3)$$

since this function obviously satisfies $p \in \mathcal{P}_{n-1}$ and (1).

Computational work:

- evaluating (3) takes $O(n^2)$ operations for each x -value
- if we add a new data pair (x_{n+1}, y_{n+1}) we have to start from scratch, and we need $O(n^2)$ operations.

Usually we want to evaluate the interpolating polynomial $p(x)$ for many x -values. Therefore we would prefer to have an algorithm where we first set up the interpolating polynomial using $O(n^2)$ operations, but then need only $O(n)$ operations to evaluate $p(x)$ for each evaluation point x .

We will perform our computations in machine arithmetic. There are several algorithms for computing $p(x)$ (Newton formula, Lagrange formula, Modified Lagrange formula, Barycentric Lagrange formula). In exact arithmetic they give the same result, but in machine arithmetic some algorithms cause unnecessarily large roundoff error (“numerically unstable algorithm”).

3. Modified Lagrange formula

With

$$\ell(x) := \prod_{j=1}^n (x - x_j)$$

$$w_j := \frac{1}{\prod_{\substack{k=1, \dots, n \\ k \neq j}} (x_j - x_k)} \quad (4)$$

we obtain from (3) the **modified Lagrange formula**

$$p(x) = \ell(x) \sum_{j=1}^n \frac{w_j}{x - x_j} y_j \quad (5)$$

Computational work:

- first: compute w_1, \dots, w_n with $O(n^2)$ operations
then: for each x -value evaluate (5) using $O(n)$ operations
- if we add a new data pair (x_{n+1}, y_{n+1}) :
update w_1, \dots, w_n : n operations
compute w_{n+1} : n operations
evaluate (5): $O(n)$ operations
so the total work is $O(n)$

We can use the updating idea to compute w_1, \dots, w_n as follows: (a loop “for $k = a$ to b ” with $b < a$ is not executed)

```
for j = 1 to n:
  w_j := 1
  for k = 1 to j - 1:
    w_k := w_k / (x_k - x_j)
    w_j := w_j / (x_j - x_k)
```

4. Barycentric formula

Note that in the case $y_1 = \dots = y_n = 1$ the interpolating polynomial must be $p(x) = 1$, hence

$$1 = \ell(x) \sum_{j=1}^n \frac{w_j}{x - x_j}$$

Solving this for $\ell(x)$ and substituting this in (5) gives the **barycentric Lagrange formula**

$$p(x) = \frac{\sum_{j=1}^n \frac{w_j}{x - x_j} y_j}{\sum_{j=1}^n \frac{w_j}{x - x_j}} \quad (6)$$

Computational work: same as for the modified Lagrange formula.

5. Numerical stability

For the evaluation in machine arithmetic the modified Lagrange formula gives the best result. For a “good choice of interpolation nodes” (such as Chebyshev nodes) the barycentric formula also gives good results, but for a “bad choice of interpolation nodes” (such as equidistant nodes) the barycentric formula may give a larger error than the modified Lagrange formula. For details and examples see the references below.

References

- N. Higham: The numerical stability of barycentric Lagrange interpolation, IMA Journal of Numerical Analysis 24 (2004)
J.-P. Berrut, L.N. Trefethen: Barycentric Lagrange interpolation, SIAM Review 46 (2004)

A. Code for modified Lagrange formula

Note that evaluating (5) when x equals one of the nodes x_j yields $0/0$ which gives NaN in machine arithmetic.

The Matlab code evaluates the $p(x)$ for a vector of x -values. We first compute the vector of y -values using (5) which will give NaN for x -values which coincide with one of the nodes. We therefore fix the y -vector by using the given function values at the nodes.

```
function ye = modlagr(x,y,xs)
% ye = modlagr(x,y,xs)
% find interpolating polynomial for data x,y and evaluate at points xs
% input:
%   vectors x,y: given data points
%   vector xs:   evaluation points
% output:
%   vector ye: values of interpolating polynomial at points xs

% compute w_j
w = zeros(size(x));
for j=1:length(x);
    w(j) = 1;
    for k=1:j-1
        w(k) = w(k)/(x(k)-x(j));
        w(j) = w(j)/(x(j)-x(k));
    end
end

% evaluate modified Lagrange formula
s = zeros(size(xs));
P = ones(size(xs));
equalnode = zeros(size(xs));
for j=1:n
    d = xs - x(j);
    equalnode(d==0) = j;           % where xs coincides with node: store node number
    s = s + y(j)*w(j)./d;         % division by zero where xs coincides with node
    P = P.*d;
end
ye = P.*s;

% fix ye for xs-values which coincide with nodes
ind = (equalnode>0);             % indices where we had division by zero
ye(ind) = y(equalnode(ind));     % replace with nodal values
```