Lecture Notes for Math 5630/6630 **Fall 2024**

Note 8: Newton's Form

Tags: math.na *Date: 10/01/2024*

Disclaimer: *This lecture note is for math 5630/6630 class only.*

Once the data points are given $(x_i, y_i)_{0\le i\le n}$, the Lagrange polynomial basis provides a way to compute interpolation with $\mathcal{O}(n)$ flops but requiring a precomputation of $\mathcal{O}(n^2)$ flops.

However, the pre-computation step makes the Lagrange polynomial interpolation unfriendly to the dynamic addition of data points.

1 Newton's Form

The Newton's form is useful when we dynamically add interpolation nodes. Consider the following scenario: we already have found an interpolation polynomial f_k through (x_0, y_0), (x_1, y_1),..., (x_k, y_k), then we are provided an addition pair (x_{k+1}, y_{k+1}), how to effectively transform f_k to f_{k+1}? If we write

$$f_{k+1}(x) = f_k(x) + c_{k+1}(x - x_0)(x - x_1)\dots(x - x_k),$$

then $f_{k+1}(x_j) = f_k(x_j)$, $j = 0, 1, \dots, k$, automatically. Therefore we only need to take care of $f_{k+1}(x_{k+1}) = y_{k+1}$, which means

$$c_{k+1} = \frac{y_{k+1} - f_k(x_{k+1})}{\prod_{j=0}^{k}(x_{k+1} - x_j)}.$$

Such an inductive procedure produces the Newton's form:

$$f_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \dots + c_n(x - x_0)\dots(x - x_{n-1}).$$

where the constant c_j depends on $x_0, x_1, \dots, x_j$ only. The polynomials $\prod_{j=0}^{k}(x - x_j)$ are called *Newton polynomials*. When the coefficients c_k are known, Newton's form can be evaluated by the famous *Horner's scheme*, which is

$$f_n(x) = c_0 + (x - x_0)(c_1 + (x - x_1)(c_2 + (x - x_2)(c_3 + \dots))),$$

the evaluation order starts from the innermost part $c_n(x - x_{n-1})$. This formulation has a complexity of $3n$ flops once c_k are given.

The computation of c_k is not cheap if using the previous formula

$$c_{k+1} = \frac{y_{k+1} - f_k(x_{k+1})}{\prod_{j=0}^{k}(x_{k+1} - x_j)}.$$

A naive algorithm with Horner's scheme roughly takes $5n^2/2 + \mathcal{O}(n)$ flops to compute all coefficients. The *divided differences* is a better way to compute c_k.

1.1 Divided Difference

Definition 1.1. *Let the interpolation nodes be $\{x_0, x_1, \ldots, x_n\}$, and the divided differences are defined recursively as follows (the square bracket is used to distinguish from the usual bracket):*

$$\begin{aligned} f[x_j] &:= f(x_j), \\ f[x_j, \ldots, x_{j+k}] &:= \frac{f[x_{j+1}, \ldots, x_{j+k}] - f[x_j, \ldots, x_{j+k-1}]}{x_{j+k} - x_j}, \end{aligned}$$

where $0 \le j, k \le n$ and $j + k \le n$.

The following example graph helps understand the relationships among the divided differences.

$$\begin{array}{lllll} f[x_0] & & & & \\ \quad \searrow & & & & \\ f[x_1] \to f[x_0, x_1] & & & & \\ \quad \searrow & \searrow & & & \\ f[x_2] \to f[x_1, x_2] & \to f[x_0, x_1, x_2] & & & \\ \quad \searrow & \searrow & \searrow & & \\ f[x_3] \to f[x_2, x_3] & \to f[x_1, x_2, x_3] & \to f[x_0, x_1, x_2, x_3] & & \\ \quad \searrow & \searrow & \searrow & \searrow & \\ f[x_4] \to f[x_3, x_4] & \to f[x_2, x_3, x_4] & \to f[x_1, x_2, x_3, x_4] & \to f[x_0, x_1, x_2, x_3, x_4] \end{array}$$

Computing all of the divided differences requires $\frac{3n^2}{2} + \mathcal{O}(n)$ flops. The following theorem is the main statement for the Newton form.

Theorem 1.2. *The interpolation polynomial f_n in Newton form is given by*

$$\begin{aligned} f_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + \\ f[x_0, \ldots, x_n](x - x_0)(x - x_1) \ldots (x - x_{n-1}). \end{aligned}$$

In other words, $c_k = f[x_0, \ldots, x_k]$.

We derive this by induction. Assume the statement is true for n and interpolation node and corresponding values $(x_0, f[x_0]), (x_1, f[x_1]), \ldots, (x_n, f[x_n])$. For a new node and value $(x_{n+1}, f[x_{n+1}])$, it is known from that c_{n+1} is the coefficient of leading power. Let g_n be the interpolation polynomial in Newton's form through nodes $(x_1, f[x_1]), \ldots, (x_{n+1}, f[x_{n+1}])$, then

$$\psi(x) := g_n(x)(x - x_0) - f_n(x)(x - x_{n+1})$$

satisfies that $\psi(x_j) = f[x_j](x_{n+1} - x_0)$ for $0 \le j \le n + 1$. Therefore

$$f_{n+1}(x) = \frac{g_n(x)(x - x_0) - f_n(x)(x - x_{n+1})}{x_{n+1} - x_0}$$

The leading power's coefficient is then

$$\frac{f[x_1, \ldots, x_{n+1}] - f[x_0, \ldots, x_n]}{x_{n+1} - x_0} = f[x_0, x_1, \ldots, x_{n+1}].$$

The divided difference $f[x_j, \ldots, x_{j+k}]$ is the coefficient of leading power of the interpolating polynomial through $(x_j, f[x_j]), \ldots, (x_{j+k}, f[x_{j+k}])$. It can be shown (by Rolle's Theorem) that

$$f[x_j, \ldots, x_{j+k}] = \frac{1}{k!} f^{(k)}(\xi)$$

for some $\xi \in [x_j, x_{j+k}]$.

Let $f_k(x)$ be the interpolating polynomial on nodes $(x_j, f[x_j])$, $\ldots$, $(x_{j+k}, f[x_{j+k}])$, then $f_k(x_s) = f(x_s)$ for $s = j, \cdots, j + k$, which means $h(x) := f_k(x) - f(x)$ has at least $(k + 1)$ roots in the range $\mathbb{I} := [x_j, x_{j+k}]$. By Rolle's Theorem, $h'(x)$ has at least k roots on $\mathbb{I}$. After applying the Rolle's Theorem k times, we find $h^{(k)}(x)$ has at least a root $\xi \in \mathbb{I}$, which means

$$f_k^{(k)}(\xi) = f^{(k)}(\xi).$$

Since f_k is a polynomial of degree k, $f_k^{(k)}(\xi) = k!\, f[x_j, \ldots, x_{j+k}]$.

The error estimate can be derived as

$$f(x) - f_n(x) = f[x_0, x_1, \ldots, x_n, x](x - x_0) \ldots (x - x_n) = \frac{1}{n!} f^{(n)}(\xi)(x - x_0) \ldots (x - x_n)$$

for certain ξ, which is a direct consequence of Newton's form on nodes $x_0, \cdots, x_n, x$.

Newton's form does not require distinct nodes. The divided difference can be defined as a limit for repeated nodes:

$$f[x_0, x_0] = \lim_{x_1 \to x_0} \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f'(x_0).$$

Moreover, using Taylor expansion, $f[\underbrace{x_0, \ldots, x_0}_{(k+1)\text{ times}}] = \frac{1}{k!} f^{(k)}(x_0)$. However, in such cases, the divided differences are not possible to be computed if the derivative values are not provided.

The algorithm to compute the divided difference can be made more efficient with a single column to store the diagonal elements. $\rightsquigarrow$ represents the number that is not changing.

$$\begin{array}{llll}
f[x_0] \rightsquigarrow f[x_0] & \rightsquigarrow f[x_0] & \rightsquigarrow f[x_0] & \rightsquigarrow f[x_0] \\
\quad \searrow & & & \\
f[x_1] \to f[x_0, x_1] & \rightsquigarrow f[x_0, x_1] & \rightsquigarrow f[x_0, x_1] & \rightsquigarrow f[x_0, x_1] \\
\quad \searrow & \quad \searrow & & \\
f[x_2] \to f[x_1, x_2] & \to f[x_0, x_1, x_2] & \rightsquigarrow f[x_0, x_1, x_2] & \rightsquigarrow f[x_0, x_1, x_2] \\
\quad \searrow & \quad \searrow & \quad \searrow & \\
f[x_3] \to f[x_2, x_3] & \to f[x_1, x_2, x_3] & \to f[x_0, x_1, x_2, x_3] & \rightsquigarrow f[x_0, x_1, x_2, x_3] \\
\quad \searrow & \quad \searrow & \quad \searrow & \quad \searrow \\
f[x_4] \to f[x_3, x_4] & \to f[x_2, x_3, x_4] & \to f[x_1, x_2, x_3, x_4] & \to f[x_0, x_1, x_2, x_3, x_4]
\end{array}$$

1.2 Hermite Interpolation

The usual polynomial interpolation only requires the values of the data function h at each node. It can be generalized when the derivative values of h are also available.

Let the tuple $(h(x_j), h^{(1)}(x_j), \ldots, h^{(m_j)}(x_j))$ be the provided derivative values at the interpolation node x_j, $j = 0, \ldots, n$ and $m_j \geq 0$. $N = \sum_{j=0}^{n}(m_j + 1)$ is the total number of constraints. It can be shown that there exists a unique polynomial $H_{N-1} \in \Pi_{N-1}$ satisfies

$$H_{N-1}^{(k)}(x_j) = y_j^k := h^{(k)}(x_j), \quad j = 0, \ldots, n, \quad 0 \leq k \leq m_j.$$

This polynomial is called *Hermite interpolation polynomial*. The idea to construct the Hermite interpolation polynomial borrows from the Lagrange polynomials, which is to find basis L_{jk} such that

$$\frac{d^p}{dx^p} L_{jk}(x_l) = \begin{cases} 1, & l = j, k = p \\ 0, & \text{otherwise.} \end{cases}$$

Once these polynomials are obtained, the Hermite interpolation is straightforward:

$$H_{N-1}(x) = \sum_{j=0}^{n} \sum_{k=0}^{m_j} y_j^k L_{jk}(x).$$

Its uniqueness can be concluded from the linear independence of the basis L_{jk}. However, the above construction method is not the simplest. It is known that Newton's form works for repeated nodes as long as the diagram's diagonal $f[x_0, x_1, \cdots, x_j]$ can be filled. Therefore, we can arrange the nodes

$$\underbrace{x_0, \ldots, x_0}_{(m_0+1)\,\text{times}}, \quad \underbrace{x_1, \ldots, x_1}_{(m_1+1)\,\text{times}}, \quad \ldots, \quad \underbrace{x_n, \ldots, x_n}_{(m_n+1)\,\text{times}}$$

In this way, all of the necessary divided differences can be computed by

$$f[x_j, \cdots, x_{j+k}] = \begin{cases} \frac{f[x_{j+1}, \cdots, x_{j+k}] - f[x_j, \cdots, x_{j+k-1}]}{x_{j+k} - x_j}, & x_j \neq x_{j+k}, \\ \frac{f^{(k)}(x_j)}{k!}, & x_j = x_{j+k}. \end{cases}$$

The error estimate for Hermite polynomial interpolation will be the same form as the usual case as if the nodes are not repeated.

Example 1.3. *Consider the Hermite cubic interpolation at data points $(x_0, f(x_0), f'(x_0))$, $(x_1, f(x_1), f'(x_1))$. Assume the resulting cubic polynomial is $h(x)$, the interpolation error can be estimated by*

$$h(x) - f(x) = f[x_0, x_0, x_1, x_1](x - x_0)^2(x - x_1)^2.$$

which implies

$$|h(x) - f(x)| \leq \frac{\max_{\zeta \in [x_0, x_1]} |f^{(4)}(\zeta)|}{384} |x_0 - x_1|^4.$$