# Lecture `interp3`: Interpolation: Lagrange's Method

October 17, 2022

**Summary**: Polynomial interpolation using the Lagrange basis. Discussion of determination and evaluation of the interpolant. Comparison with Newton's method; nested interpolation.

**References**: Some aspects taken from C. F. Van Loan's *Introduction to Scientific Computing*. See also T. Sauer's *Numerical Analysis*, Section 3.1.1, first edition, pages 140-141.

## Lagrange Interpolation

The previous lecture discussed Newton's interpolation method, which reduces the full Vandermonde system to a lower–triangular system (achieving an upper–triangular system with Newton's approach is also possible). This lecture looks at polynomial interpolation via a different set of basis functions: Lagrange polynomials. These are tailored to trivialize the Vandermonde matrix, that is to give $V = I$ (identity)!

Consider data $D_N = \{(x_i, y_i)\}_{i=1}^N$ and a polynomial basis set $\mathcal{B}_N = \{\phi_i\}_{i=1}^N$. Our problem (by now familiar) is to find coefficients $\{c_i\}_{i=1}^N$ such that the function (polynomial)

$$p(x) = \sum_{i=1}^{N} c_i \phi_i(x)$$

satisfies $p(x_i) = y_i$ for every $i = 1, 2, \ldots, N$. Consider a new data–dependent basis $\phi_i(x) = \ell_i(x)$, where the $i$th *Lagrange polynomial* is

$$\begin{aligned} \ell_i(x) &= \frac{(x - x_1)(x - x_2)\ldots(x - x_{i-1})(x + x_{i+1})\ldots(x - x_{N-1})(x - x_N)}{(x_i - x_1)(x_i - x_2)\ldots(x_i - x_{i-1})(x_i + x_{i+1})\ldots(x_i - x_{N-1})(x_i - x_N)} \\ &= \prod_{j=1, j\neq i}^{N} \left(\frac{x - x_j}{x_i - x_j}\right) \end{aligned} \tag{1}$$

Notice that $\ell_i(x)$ depends on all of the data points $x_i$ and is a degree–$(N-1)$ polynomial. This would seem an unnecessarily complicated form for the basis functions; however, it proves golden for interpolation. Indeed, from (1) we observe that

$$\ell_i(x_j) = \begin{cases} 0 & \text{if } j \neq i \\ 1 & \text{if } j = i. \end{cases}$$

In other words, we have constructed a polynomial which is zero at every data node except $x = x_i$, and at this node the polynomial takes the value 1. Why is this useful? Well, for the basis $\mathcal{B}_N = \{\ell_i(x)\}_{i=1}^N$, the Vandermonde matrix associated with $D_N$ and $\mathcal{B}_N$ is simply the identity matrix,

$$V = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

We don't need to invert this matrix to solve for the $c_i$ coefficients: the identity matrix is its own inverse. Then our familiar interpolation equation $V\boldsymbol{c} = \boldsymbol{y}$ becomes

$$\boldsymbol{c} = \boldsymbol{y}.$$

Whence the coefficients $c_i$ are simply the data values $y_i$. There is no system to solve! The Lagrange polynomial basis makes the process of determining the interpolant trivial.

**Example 1**
Using the Lagrange basis, find the polynomial which interpolates the data $D_4 = \{(-1, 2), (0, 1), (1, 3), (2, 2)\}$. With (1), or from first principles, we find that

$$\begin{aligned} \ell_1(x) &= -\frac{1}{6}x(x-1)(x-2) \\ \ell_2(x) &= \frac{1}{2}(x+1)(x-1)(x-2) \\ \ell_3(x) &= -\frac{1}{2}x(x+1)(x-2) \\ \ell_4(x) &= \frac{1}{6}x(x+1)(x-1). \end{aligned}$$

Note that these are all degree–3 polynomials. A Lagrange polynomial for $N$ data points is always of degree $N - 1$. We have now determined the basis functions for our interpolation, and can straightaway write down the interpolant from the data:

$$\begin{aligned} p(x) &= y_1\ell_1(x) + y_2\ell_2(x) + y_3\ell_3(x) + y_4\ell_4(x) \\ &= 2\ell_1(x) + \ell_2(x) + 3\ell_3(x) + 2\ell_4(x) \end{aligned}$$

The example shows how easily one can write down the Lagrange interpolant. But what if we want to graph the interpolant? We have seen in previous lectures that polynomials have nested forms for which Horner's method may be used for evaluation. However, with the Lagrange basis there is no natural multiplicative nesting of functions. Indeed, we must simply evaluate the functions explicitly. This is Lagrange interpolation's disadvantage: function evaluations are expensive. In addition, coding the basis functions is more difficult than for the Newton or monomial bases. There is another more subtle disadvantage of Lagrange interpolation which we will discuss in the next section. We point out that there is a more sophiscated approach to Lagrange interpolation, one based on the *barycentric form* of the Lagrange representation. However, this topic lies beyond the scope of the course.

## Nested Interpolation

Having considered all of the canonical one–dimensional polynomial interpolation methods, we now discuss the issue of *nested interpolation*. Whereas before we used the term *nested* in regard to *nested evaluation* (a process for evaluating a polynomial), here we use the term in the different sense of adding more data points. The nested interpolation problem is the following: given a data set $D_N$ and polynomial basis $\mathcal{B}_N$, suppose that we have computed the coefficients $c_i$ and obtained an interpolant[1] $p_N(x)$. After completing this work, we decide that this interpolation is not adequate, and wish to add one more data point to the original data set. So now we have $D_{N+1} = D_N \bigcup (x_{N+1}, y_{N+1})$, and if we wish to consider the larger interpolation problem for $D_{N+1}$, then we must consider a larger basis set $\mathcal{B}_{N+1}$. Ideally, this new set would be $\mathcal{B}_N \bigcup \{\phi_{N+1}\}$, so we can reuse the previously constructed $N$ basis functions. Regarding the new size–$(N+1)$ interpolation problem, we ask: can we use our knowledge of $p_N(x)$ to inexpensively determine $p_{N+1}(x)$? In other words, in calculating $p_{N+1}(x)$, can we reuse either $\boldsymbol{c}$ or $\mathcal{B}_N$ or both? It turns out that for the monomial basis we can reuse $\mathcal{B}_N$, for the Lagrange basis we can reuse $\boldsymbol{c}$ (just the function values $\boldsymbol{y}$, of course), and for the Newton basis we can reuse both!

To determine the higher–order interpolant $p_{N+1}(x)$ with Lagrange interpolation, *all* of the basis functions need to be adjusted as

$$\ell_i(x) = \prod_{j=1, j\neq i}^{N} \frac{(x - x_j)}{(x_i - x_j)} \rightarrow \ell_i(x) \frac{(x - x_{N+1})}{(x_i - x_{N+1})},$$

so this approach has the disadvantage that $\mathcal{B}_N$ needs to be recomputed. For the monomial basis the addition of an extra data point does not affect the existing basis, as now

$$\mathcal{B}_{N+1} = \{x^{k-1}\}_{k=1}^{N} \bigcup \{x^N\} = \mathcal{B}_N \bigcup \{x^N\}.$$

However, with the monomial basis the solution $\boldsymbol{c}$ to the $N$-point interpolation problem cannot be reused to get the solution for the $(N+1)$-point problem. Consider now the issue with regard to the Newton basis. Let's assume the data values $y_i = f(x_i)$ come from a function $f(x)$. Recall that the Newton basis for the interpolation problem has the form $\{1, (x - x_1), (x - x_1)(x - x_2), \cdots, (x - x_1)(x - x_2) \cdots (x - x_{N-1})\}$. Also recall that for this basis, the solution coefficients for the interpolation problem are the Newton divided differences: $c_i = f[x_1, x_2, \cdots, x_i]$. The Newton basis is multiplicatively nested, so the $(N+1)$st basis function

$$\phi_{N+1}(x) = \phi_N(x)(x - x_N) = (x - x_1)(x - x_2) \cdots (x - x_{N-1})(x - x_N) \tag{2}$$

does not require a redefinition of the basis functions $\phi_1(x)$ through $\phi_N(x)$. The updated interpolant is then

$$p_{N+1}(x) = p_N(x) + c_{N+1}\phi_{N+1}(x), \tag{3}$$

a simply formula that only requires us to compute $c_{N+1} = f[x_1, x_2, \cdots, x_N, x_{N+1}]$. We don't have to compute this top divided difference completely from scratch, since we can reuse the divided difference table used to compute $c_N = f[x_1, x_2, \cdots, x_N]$.

[1] The polynomial $p_N(x)$ here is degree $N-1$, and Sauer would denote it as $p_{N-1}(x)$. Here we use $p_N(x)$ since we are focused on the number of data points which defines the interpolant.

**Example 2**
Let $D_3 = \{(-1,2),(0,1),(1,3)\}$ and $D_4 = \{(-1,2),(0,1),(1,3),(2,2)\}$. So $D_4$ is $D_3$ with one extra data point. Let's first compute the interpolant $p_3(x)$. Here is the divided difference table.

```
x_k   |  f(x_k)
      |
-1    |    2
      |           -1
 0    |    1            3/2
      |            2
 1    |    3
```

Therefore, we have

$$\begin{aligned} p_3(x) &= f(x_1) + f[x_1,x_2](x-x_1) + f[x_1,x_2,x_3](x-x_1)(x-x_2) \\ &= 2 - 1(x+1) + \tfrac{3}{2}(x+1)x. \end{aligned} \tag{4}$$

Adding the new data point, we have a new table.

```
x_k   |  f(x_k)
      |
-1    |    2
      |           -1
 0    |    1            3/2
      |            2            -1
 1    |    3           -3/2     ---
      |           -1     ---
 2    |    2     ---
---       ---
```

Notice to get the new table we could reuse all of the previous table (all numbers save the underlined ones). The later interpolant is therefore

$$\begin{aligned} p_4(x) &= f(x_1) + f[x_1,x_2](x-x_1) + f[x_1,x_2,x_3](x-x_1)(x-x_2) \\ &\quad + f[x_1,x_2,x_3,x_4](x-x_1)(x-x_2)(x-x_3) \\ &= p_3(x) + f[x_1,x_2,x_3,x_4](x-x_1)(x-x_2)(x-x_3) \\ &= p_3(x) - \tfrac{3}{2}(x+1)x(x-1) \\ &= 2 - 1(x+1) + \tfrac{3}{2}(x+1)x - (x+1)x(x-1). \end{aligned} \tag{5}$$

The example shows how easy it is to extend existing polynomial interpolants to include additional data using the Newton interpolation basis. We only needed to compute $N$ additional divided differences at $O(N)$ cost in order to find the $(N+1)$st order interpolant.