**Newton interpolation**

The Newton method for polynomial interpolation is yet another alternative which enables both efficient evaluation and allows for incremental construction. Additionally, it allows both the coefficients $a_i$, as well as the derivative $P'(x)$ to be evaluated efficiently.

Here is the basic idea: We want to interpolate data points

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$$

**Step 0:** Define a 0-degree polynomial $P_0(x)$ that just interpolates $(x_0, y_0)$. Obviously, we can achieve that by simply selecting $P_0(x) = y_0$.

**Step 1:** Define a 1-degree polynomial $P_1(x)$ that now interpolates both $(x_0, y_0)$ and $(x_1, y_1)$. We also want to take advantage of the previously defined $P_0(x)$ by constructing $P_1$ as:

$$P_1(x) = P_0(x) + M_1(x)$$

$M_1(x)$ is a 1-degree polynomial and it needs to satisfy:

$$P_1(x_0) = P_0(x_0) + M_1(x_0) = y_0$$

Thus, $x_0$ is a root of $M_1(x)$ and we must have $M_1(x) = c_1(x - x_0)$. We can determine $c_1$ by substituting:

$$P_1(x_1) = P_0(x_1) + c_1(x_1 - x_0) \Rightarrow c_1 = \frac{P_1(x_1) - P_0(x_1)}{x_1 - x_0} = \frac{y_1 - P_0(x_1)}{x_1 - x_0}$$

Note that $P_0(x_1) (= y_0)$ is an expression we already know to evaluate, as this polynomial was fully defined in the previous step.

**Step 2:** Now construct $P_2(x)$ which interpolates the three points $(x_0, y_0), (x_1, y_1)$ and $(x_2, y_2)$. Define it as:

$$P_2(x) = P_1(x) + M_2(x)$$

where $M_2(x)$ is a polynomial of degree 2. Once again, we observe that:

$$\begin{align*}
P_2(x_0) &= P_1(x_0) + M_2(x_0) \\ &= y_0 \quad (= y_0) \\
P_2(x_1) &= P_1(x_1) + M_2(x_1) \\ &= y_1 \quad (= y_1)
\end{align*}$$

$\Rightarrow M_2(x_0) = M_2(x_1) = 0$

Thus, $M_2(x)$ must have the form:

$$M_2(x) = c_2(x - x_0)(x - x_1)$$
Substituting $x \leftarrow x_2$, we get an expression for $c_2$:

$$y_2 = P_2(x_2) = P_1(x_2) + c_2(x_2 - x_0)(x_2 - x_1) \Rightarrow c_2 = \frac{y_2 - P_1(x_2)}{(x_2 - x_0)(x_2 - x_1)}$$

**Step k:** In the previous step, we constructed a polynomial $P_{k-1}$ of degree $k-1$ that interpolates $(x_0, y_0), \ldots, (x_{k-1}, y_{k-1})$. We will use this $P_{k-1}(x)$ and now define an $n$-degree polynomial $P_k(x)$ such that all of $(x_0, y_0), \ldots, (x_k, y_k)$ are now interpolated. Again,

$$P_k(x) = P_{k-1}(x) + M_k(x) \text{ where } M_k \text{ has degree } k$$

Now, for any $i \in \{0, 1, \ldots, k-1\}$ we have:

$$\frac{P_k(x_i)}{=y_i} = \frac{P_k(x_i) + M_k(x_i)}{=y_i} \Rightarrow M_k(x_i) = 0, \ \forall i = 0, 1, \ldots, k - 1$$

Thus, the $k$-degree polynomial $M_k$ must have the form

$$M_k(x) = c_k(x - x_0) \ldots (x - x_{k-1})$$

Substituting $x \leftarrow x_k$, we get:

$$y_k = P_k(x_k) = P_{k-1}(x_k) + c_k(x_k - x_0) \ldots (x_k - x_{k-1}) \Rightarrow c_k = \frac{y_k - P_{k-1}(x_k)}{\prod_{j=0}^{k-1}(x_k - x_j)}$$

Since we generated each of the polynomials $P_k(x)$ by adding the “incremental” polynomial $M_k(x)$ to the previous result, the interpolating polynomial generated after $n$ steps is simply

$$P(x) = P_n(x) = M_0(x) + M_1(x) + M_2(x) + \cdots + M_n(x)$$

where we have defined $M_0(x) = P_0(x)$ for uniformity of notation. Every polynomial $M_i(x)$ in this process is written as:

$$M_i(x) = c_i n_i(x) \text{ where } n_i(x) = \prod_{j=0}^{i-1}(x - x_j)$$

With this notation, the interpolating polynomial $P_n(x)$ is expressed as:

$$P(x) = c_0 n_0(x) + c_1 n_1(x) + \cdots + c_n n_n(x)$$
Where
\[
\begin{align*}
n_0(x) &= 1 \\
n_1(x) &= (x - x_0) \\
n_2(x) &= (x - x_0)(x - x_1) \\
& \vdots \\
n_k(x) &= (x - x_0)(x - x_1) \cdots (x - x_{k-1})
\end{align*}
\]
These are the Newton polynomials (compare with the Lagrange polynomials \( l_i(x) \)). Note the \( x_i \)'s are called the centers.

Let us evaluate Newton interpolation as we did with the other methods:

- Cost of evaluating \( P(x) \) for an arbitrary \( x \): EASY. This can be accelerated using a modification of Horner’s scheme.

\[
P(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)(x - x_1) \cdots (x - x_{n-1})
\]
\[
= c_0 + (x - x_0)[c_1 + (x - x_1)[c_2 + (x - x_2)[c_3 + \cdots + c_n + (x - x_{n-1})c_n]]]
\]
e.g. for \( n = 3 \)
\[
P(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + c_3(x - x_0)(x - x_1)(x - x_2)
\]
\[
= c_0 + (x - x_0)[c_1 + (x - x_1)[c_2 + (x - x_2)c_3]]
\]
We can easily see that \( P(x) \) is computed at a cost of \( 3n \) operations.

- Cost of determining \( P(x) \) (i.e. the coefficients \( \{c_i\} \))

We saw one way of computing them when describing the overall method. There is, however, another efficient and systematic way to compute them called divided differences. A divided difference is a function defined over a set of sequentially indexed centers, e.g. \( x_i, x_{i+1}, \ldots, x_{i+j-1}, x_{i+j} \). The divided difference of these values is denoted by:
\[
f[x_{i}, x_{i+1}, \ldots, x_{i+j-1}, x_{i+j}]
\]
The value of this symbol is defined recursively. We start with divided differences with just one argument:
\[
f[x_i] := f(x_i) = y_i
\]
with two arguments:
\[
f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}
\]
with three arguments:

\[ f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} \]

with \( j + 1 \) arguments:

\[ f[x_i, x_{i+1}, \ldots, x_{i+j-1}, x_{i+j}] = \frac{f[x_{i+1}, \ldots, x_i, x_{i+j-1}] - f[x_i, \ldots, x_{i+j-1}]}{x_i - x_{i+j}} \]

The fact that makes divided differences so useful is that \( f[x_i, x_{i+1}, \ldots, x_{i+j-1}, x_{i+j}] \) can be shown to be the coefficient of the highest power of \( x \) in a polynomial that interpolates through

\[(x_1, y_1), (x_{i+1}, y_{i+1}), \ldots, (x_{i+j-1}, y_{i+j-1}), (x_{i+j}, y_{i+j})\]

Why is this useful? Remember the polynomial interpolating \((x_0, y_0), \ldots, (x_k, y_k)\) is

\[ P_k(x) = \frac{P_{k-1}(x)}{x^{k-1}} + c_k (x - x_0) \ldots (x - x_{k-1}) \]

Thus, we must have \( c_k = f[x_0, x_1, x_2, \ldots, x_k] \). Consequently

\[
P(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + f[x_0, x_1, \ldots, x_2](x - x_0) \ldots (x - x_{n-1})
\]

So, if we can quickly evaluate the divided differences, we have determined \( P(x) \).

**Example**

\((x_0, y_0) = (-2, -27), \ (x_1, y_1) = (0, -1), \ (x_2, y_2) = (1, 0)\)

\[
\begin{align*}
f[x_0] &= y_0 = -27 \\
f[x_1] &= y_1 = -1 \\
f[x_2] &= y_2 = 0 \\
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{-1 - (-27)}{0 - (-2)} = 13 \\
f[x_1, x_2] &= \frac{f[x_2] - f[x_1]}{x_2 - x_1} = \frac{0 - (-1)}{1 - 0} = 1 \\
f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{1 - 13}{1 - (-2)} = -4
\end{align*}
\]
Thus
\[ P(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) = -27 + 13(x + 2) - 4(x + 2)x \]

We can compute all divided differences more easily, by tabulating them as follows:

<table>
<thead>
<tr>
<th>(x)</th>
<th>(f[\cdot])</th>
<th>(f[\cdot, \cdot])</th>
<th>(f[\cdot, \cdot, \cdot])</th>
<th>(f[\cdot, \cdot, \cdot, \cdot])</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_0)</td>
<td>(f[x_0])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x_1)</td>
<td>(f[x_1])</td>
<td>(f[x_0, x_1])</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x_2)</td>
<td>(f[x_2])</td>
<td>(f[x_1, x_2])</td>
<td>(f[x_0, x_1, x_2])</td>
<td></td>
</tr>
<tr>
<td>(x_3)</td>
<td>(f[x_3])</td>
<td>(f[x_2, x_3])</td>
<td>(f[x_1, x_2, x_3])</td>
<td>(f[x_0, x_1, x_2, x_3])</td>
</tr>
<tr>
<td>(x_4)</td>
<td>(f[x_4])</td>
<td>(f[x_3, x_4])</td>
<td>(f[x_2, x_3, x_4])</td>
<td>(f[x_1, x_2, x_3, x_4])</td>
</tr>
</tbody>
</table>

Note that the first column of the table is just the \(x_k\) values of data points to be interpolated. The second column contains the respective \(y\)-values, \(y_0 \ldots y_n\) since \(f[x_i] := y_i\). Finally, the computed Newton coefficients \(c_k\) appear on the diagonal.

The recursive definition of the divided differences translates directly to a more “geometrical” method of calculation, defined directly on the divided difference table. To compute an entry, say \(Y\) of the table, we first locate the 4 coefficients \(a, b, c, d\) according to the following scheme:

\[ \begin{array}{cccc}
    & d & \leftarrow & \leftarrow \\
    & \downarrow & & \leftarrow \\
    & c & \leftarrow & \leftarrow \\
\end{array} \]

Then, the new entry \(Y\) is computed as follows:

\[ Y = \frac{a - b}{c - d} \]

**Example**

\((x_0, y_0) = (-2, -27), (x_1, y_1) = (0, -1), (x_2, y_2) = (1, 0)\)

<table>
<thead>
<tr>
<th>(x_i)'s</th>
<th>(y_i)'s</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>-27</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Thus, the Newton coefficients are \(c_0 = -27, c_1 = 13, c_2 = -4\) as before.