Newton Polynomials

It is sometimes useful to find several approximating polynomials $P_1(x)$, $P_2(x)$, \ldots, $P_N(x)$ and then choose the one that suits our needs. If the Lagrange polynomials are used, there is no constructive relationship between $P_{N-1}(x)$ and $P_N(x)$. Each polynomial has to be constructed individually, and the work required to compute the higher-degree polynomials involves many computations. We take a new approach and construct Newton polynomials that have the recursive pattern

\begin{align*}
P_1(x) &= a_0 + a_1(x - x_0), \\
P_2(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1),
\end{align*}
Example 4.10. Given the centers \(x_0 = 1, x_1 = 3, x_2 = 4\), and \(x_3 = 4.5\) and the coefficients \(a_0 = 5, a_1 = -2, a_2 = 0.5, a_3 = -0.1\), and \(a_4 = 0.003\), find \(P_1(x), P_2(x), P_3(x), \) and \(P_4(x)\) and evaluate \(P_k(2.5)\) for \(k = 1, 2, 3, 4\).

Using formulas (1) through (4), we have
\[
\begin{align*}
P_1(x) &= 5 - 2(x - 1), \\
P_2(x) &= 5 - 2(x - 1) + 0.5(x - 1)(x - 3), \\
P_3(x) &= P_2(x) - 0.1(x - 1)(x - 3)(x - 4), \\
P_4(x) &= P_3(x) + 0.003(x - 1)(x - 3)(x - 4)(x - 4.5).
\end{align*}
\]
Evaluating the polynomials at \(x = 2.5\) results in
\[
\begin{align*}
P_1(2.5) &= 5 - 2(1.5) = 2, \\
P_2(2.5) &= P_1(2.5) + 0.5(1.5)(-0.5) = 1.625, \\
P_3(2.5) &= P_2(2.5) - 0.1(1.5)(-0.5)(-1.5) = 1.5125, \\
P_4(2.5) &= P_3(2.5) + 0.003(1.5)(-0.5)(-1.5)(-2.0) = 1.50575.
\end{align*}
\]

**Nested Multiplication**

If \(N\) is fixed and the polynomial \(P_N(x)\) is evaluated many times, then nested multiplication should be used. The process is similar to nested multiplication for ordinary polynomials, except that the centers \(x_k\) must be subtracted from the independent variable \(x\). The nested multiplication form for \(P_3(x)\) is
\[
P_3(x) = ((a_3(x - x_2) + a_2)(x - x_1) + a_1)(x - x_0) + a_0.
\]
To evaluate \( P_3(x) \) for a given value of \( x \), start with the innermost grouping and form successively the quantities

\[
S_3 = a_3, \\
S_2 = S_3(x - x_2) + a_2, \\
S_1 = S_2(x - x_1) + a_1, \\
S_0 = S_1(x - x_0) + a_0.
\]

The quantity \( S_0 \) is now \( P_3(x) \).

**Example 4.11.** Compute \( P_3(2.5) \) in Example 4.10 using nested multiplication.

Using (6), we write

\[
P_3(x) = ((-0.1(x - 4) + 0.5)(x - 3) - 2)(x - 1) + 5.
\]

The values in (7) are

\[
S_3 = -0.1, \\
S_2 = -0.1(2.5 - 4) + 0.5 = 0.65, \\
S_1 = 0.65(2.5 - 3) - 2 = -2.325, \\
S_0 = -2.325(2.5 - 1) + 5 = 1.5125.
\]

Therefore, \( P_3(2.5) = 1.5125 \). ■

**Polynomial Approximation, Nodes, and Centers**

Suppose that we want to find the coefficients \( a_k \) for all the polynomials \( P_1(x), \ldots, P_N(x) \) that approximate a given function \( f(x) \). Then \( P_k(x) \) will be based on the centers \( x_0, x_1, \ldots, x_k \) and have the nodes \( x_0, x_1, \ldots, x_{k+1} \). For the polynomial \( P_1(x) \) the coefficients \( a_0 \) and \( a_1 \) have a familiar meaning. In this case

\[
(8) \quad P_1(x_0) = f(x_0) \quad \text{and} \quad P_1(x_1) = f(x_1).
\]

Using (1) and (8) to solve for \( a_0 \), we find that

\[
(9) \quad f(x_0) = P_1(x_0) = a_0 + a_1(x_0 - x_0) = a_0.
\]

Hence \( a_0 = f(x_0) \). Next, using (1), (8), and (9), we have

\[
f(x_1) = P_1(x_1) = a_0 + a_1(x_1 - x_0) = f(x_0) + a_1(x_1 - x_0),
\]

which can be solved for \( a_1 \), and we get

\[
(10) \quad a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.
\]
Hence $a_1$ is the slope of the secant line passing through the two points $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

The coefficients $a_0$ and $a_1$ are the same for both $P_1(x)$ and $P_2(x)$. Evaluating (2) at the node $x_2$, we find that

\begin{align}
  f(x_2) = P_2(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1).
\end{align}

The values for $a_0$ and $a_1$ in (9) and (10) can be used in (11) to obtain

\begin{align}
  a_2 = \frac{f(x_2) - a_0 - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\
  = \left( \frac{f(x_2) - f(x_0)}{x_2 - x_0} - \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) / (x_2 - x_1).
\end{align}

For computational purposes we prefer to write this last quantity as

\begin{align}
  a_2 = \left( \frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right) / (x_2 - x_0).
\end{align}

The two formulas for $a_2$ can be shown to be equivalent by writing the quotients over the common denominator $(x_2 - x_1)(x_2 - x_0)(x_1 - x_0)$. The details are left for the reader. The numerator in (12) is the difference between the first-order divided differences. In order to proceed, we need to introduce the idea of divided differences.

**Definition 4.1.** The divided differences for a function $f(x)$ are defined as follows:

\begin{align}
  f[x_k] &= f(x_k), \\
  f[x_{k-1}, x_k] &= \frac{f[x_k] - f[x_{k-1}]}{x_k - x_{k-1}}, \\
  f[x_{k-2}, x_{k-1}, x_k] &= \frac{f[x_{k-1}, x_k] - f[x_{k-2}, x_{k-1}]}{x_k - x_{k-2}}, \\
  f[x_{k-3}, x_{k-2}, x_{k-1}, x_k] &= \frac{f[x_{k-2}, x_{k-1}, x_k] - f[x_{k-3}, x_{k-2}, x_{k-1}]}{x_k - x_{k-3}}.
\end{align}

The recursive rule for constructing higher-order divided differences is

\begin{align}
  f[x_{k-j}, x_{k-j+1}, \ldots, x_k] &= \frac{f[x_{k-j+1}, \ldots, x_k] - f[x_{k-j}, \ldots, x_k]}{x_k - x_{k-j}}
\end{align}

and is used to construct the divided differences in Table 4.8. ▲

The coefficients $a_k$ of $P_N(x)$ depend on the values $f(x_j)$, for $j = 0, 1, \ldots, k$. The next theorem shows that $a_k$ can be computed using divided differences:

\begin{align}
  a_k = f[x_0, x_1, \ldots, x_k].
\end{align}
The Newton form of this polynomial is

\[ PN(x) = a_0 + a_1(x - x_0) + \cdots + a_N(x - x_0)(x - x_1) \cdots (x - x_{N - 1}), \]

where \( a_k = f[x_0, x_1, \ldots, x_k] \), for \( k = 0, 1, \ldots, N \).

Remark. If \( \{ (x_j, y_j) \}_{j=0}^N \) is a set of points whose abscissas are distinct, the values \( f(x_j) = y_j \) can be used to construct the unique polynomial of degree \( \leq N \) that passes through the \( N + 1 \) points.

Corollary 4.2 (Newton Approximation). Assume that \( PN(x) \) is the Newton polynomial given in Theorem 4.5 and is used to approximate the function \( f(x) \), that is,

\[ f(x) = PN(x) + EN(x). \]

If \( f \in C^{N+1}[a, b] \), then for each \( x \in [a, b] \) there corresponds a number \( c = c(x) \in (a, b) \), so that the error term has the form

\[ EN(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_N)f^{(N+1)}(c)}{(N + 1)!}. \]

Remark. The error term \( EN(x) \) is the same as the one for Lagrange interpolation, which was introduced in equation (16) of Section 4.3.

It is of interest to start with a known function \( f(x) \) that is a polynomial of degree \( N \) and compute its divided-difference table. In this case we know that \( f^{(N+1)}(x) = 0 \) for all \( x \), and calculation will reveal that the \( (N + 1) \)st divided difference is zero. This will happen because the divided difference (14) is proportional to a numerical approximation for the \( j \)th derivative.
Table 4.9  Divided-Difference Table Used for Constructing the Newton Polynomial \( P_3(x) \) in Example 4.12.

<table>
<thead>
<tr>
<th>( x_k )</th>
<th>( f[x_k] )</th>
<th>First divided difference</th>
<th>Second divided difference</th>
<th>Third divided difference</th>
<th>Fourth divided difference</th>
<th>Fifth divided difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_0 = 1 )</td>
<td>3</td>
<td>0</td>
<td>15</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_1 = 2 )</td>
<td>15</td>
<td>15</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_2 = 3 )</td>
<td>48</td>
<td>33</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_3 = 4 )</td>
<td>105</td>
<td>57</td>
<td>12</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( x_4 = 5 )</td>
<td>192</td>
<td>87</td>
<td>15</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Example 4.13. Construct a divided-difference table for \( f(x) = \cos(x) \) based on the five points \((k, \cos(k))\), for \( k = 0, 1, 2, 3, 4\). Use it to find the coefficients \( a_k \) and the four Newton interpolating polynomials \( P_k(x) \), for \( k = 1, 2, 3, 4\).

For simplicity we round off the values to seven decimal places, which are displayed in Table 4.10. The nodes \( x_0, x_1, x_2, x_3 \) and the diagonal elements \( a_0, a_1, a_2, a_3, a_4 \) in...
The graphs of \( y = \cos(x) \) and the linear Newton polynomial \( y = P_1(x) \) based on the nodes \( x_0 = 0.0 \) and \( x_1 = 1.0 \).

Figure 4.14 (b) The graphs of \( y = \cos(x) \) and the quadratic Newton polynomial \( y = P_2(x) \) based on the nodes \( x_0 = 0.0, x_1 = 1.0, \) and \( x_2 = 2.0 \).

Table 4.10 are used in formula (16), and we write down the first four Newton polynomials:

\[
\begin{align*}
P_1(x) &= 1.0000000 - 0.4596977(x - 0.0), \\
P_2(x) &= 1.0000000 - 0.4596977(x - 0.0) - 0.2483757(x - 0.0)(x - 1.0), \\
P_3(x) &= 1.0000000 - 0.4596977(x - 0.0) - 0.2483757(x - 0.0)(x - 1.0) + 0.1465592(x - 0.0)(x - 1.0)(x - 2.0), \\
P_4(x) &= 1.0000000 - 0.4596977(x - 0.0) - 0.2483757(x - 0.0)(x - 1.0) + 0.1465592(x - 0.0)(x - 1.0)(x - 2.0) - 0.0146568(x - 0.0)(x - 1.0)(x - 2.0)(x - 3.0).
\end{align*}
\]

The following sample calculation shows how to find the coefficient \( a_2 \).

\[
\begin{align*}
f[x_0, x_1] &= \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{0.5403023 - 0.4596977}{1.0 - 0.0} = -0.4596977, \\
f[x_1, x_2] &= \frac{f[x_2] - f[x_1]}{x_2 - x_1} = \frac{-0.4161468 - 0.5403023}{2.0 - 1.0} = -0.9564491, \\
a_2 &= f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{-0.9564491 + 0.4596977}{2.0 - 0.0} = -0.2483757.
\end{align*}
\]

The graphs of \( y = \cos(x) \) and \( y = P_1(x), y = P_2(x), \) and \( y = P_3(x) \) are shown in Figure 4.14(a), (b), and (c), respectively.

For computational purposes the divided differences in Table 4.8 need to be stored in an array which is chosen to be \( D(k, j) \), so that

\[
D(k, j) = f[x_{k-j}, x_{k-j+1}, \ldots, x_k] \quad \text{for} \ j \leq k.
\]
Relation (14) is used to obtain the formula to recursively compute the entries in the array:

\[ D(k, j) = \frac{D(k, j - 1) - D(k - 1, j - 1)}{x_k - x_{k-j}}. \]  

Notice that the value \( a_k \) in (15) is the diagonal element \( a_k = D(k, k) \). The algorithm for computing the divided differences and evaluating \( P_N(x) \) is now given. We remark that Problem 2 in Algorithms and Programs investigates how to modify the algorithm so that the values \( \{a_k\} \) are computed using a one-dimensional array. \[ \square \]
John H. Mathews and Kurtis K. Fink

Prentice-Hall Inc.
Upper Saddle River, New Jersey, USA
http://vig.prenhall.com/