To prove that the interpolating polynomial passes through the data points, we substitute $x = x_j$ into Eq. 3.1a) and then utilize Eq. 3.2). The result is

$$P_n\ x_j) = \sum_{i=0}^{n} y_i \ell_i\ x_j) = \sum_{i=0}^{n} y_i \delta_{ij} = y_j$$

It can be shown that the error in polynomial interpolation is

$$f\ x) - P_n\ x) = \frac{x - x_0)\ x - x_1) \cdots\ x - x_n)}{n+1)!} f^{\ n+1)}\ \xi) \qquad 3.3)$$

where ξ lies somewhere in the interval $\ x_0, x_n)$; its value is otherwise unknown. It is instructive to note that the farther a data point is from x, the more it contributes to the error at x.

Newton's Method

Although Lagrange's method is conceptually simple, it does not lend itself to an efficient algorithm. A better computational procedure is obtained with Newton's method, where the interpolating polynomial is written in the form

$$P_n\ x) = a_0 + \ x - x_0)a_1 + \ x - x_0)\ x - x_1)a_2 + \cdots + \ x - x_0)\ x - x_1) \cdots\ x - x_{n-1})a_n$$

This polynomial lends itself to an efficient evaluation procedure. Consider, for example, four data points $\ n = 3)$. Here the interpolating polynomial is

$$\begin{aligned} P_3\ x) &= a_0 + \ x - x_0)a_1 + \ x - x_0)\ x - x_1)a_2 + \ x - x_0)\ x - x_1)\ x - x_2)a_3 \\ &= a_0 + \ x - x_0)\{a_1 + \ x - x_1)\,[a_2 + \ x - x_2)a_3]\} \end{aligned}$$

which can be evaluated backwards with the following recurrence relations:

$$\begin{aligned} P_0\ x) &= a_3 \\ P_1\ x) &= a_2 + \ x - x_2)\,P_0\ x) \\ P_2\ x) &= a_1 + \ x - x_1)\,P_1\ x) \\ P_3\ x) &= a_0 + \ x - x_0)\,P_2\ x) \end{aligned}$$

For arbitrary n we have

$$P_0\ x) = a_n \qquad P_k\ x) = a_{n-k} + \ x - x_{n-k})\,P_{k-1}\ x), \quad k = 1, 2, \ldots, n \qquad 3.4)$$

Denoting the x-coordinate array of the data points by `xData` and the degree of the polynomial by `n`, we have the following algorithm for computing $P_n\ x)$:

```
p = a[n]
for k in range 1,n+1):
    p = a[n-k] +  x - xData[n-k])*p
```

The coefficients of P_n are determined by forcing the polynomial to pass through each data point: $y_i = P_n\ x_i),\quad i = 0, 1, \ldots, n$. This yields the simultaneous equations

$$
\begin{aligned}
y_0 &= a_0\\
y_1 &= a_0 + \ x_1 - x_0)a_1\\
y_2 &= a_0 + \ x_2 - x_0)a_1 + \ x_2 - x_0)\ x_2 - x_1)a_2\\
&\ \vdots\\
y_n &= a_0 + \ x_n - x_0)a_1 + \cdots + \ x_n - x_0)\ x_n - x_1)\cdots\ x_n - x_{n-1})a_n
\end{aligned}
\qquad \text{a)}
$$

Introducing the *divided differences*

$$
\begin{aligned}
\nabla y_i &= \frac{y_i - y_0}{x_i - x_0}, \quad i = 1, 2, \ldots, n\\
\nabla^2 y_i &= \frac{\nabla y_i - \nabla y_1}{x_i - x_1}, \quad i = 2, 3, \ldots, n\\
\nabla^3 y_i &= \frac{\nabla^2 y_i - \nabla^2 y_2}{x_i - x_2}, \quad i = 3, 4, \ldots n\\
&\ \vdots\\
\nabla^n y_n &= \frac{\nabla^{n-1} y_n - \nabla^{n-1} y_{n-1}}{x_n - x_{n-1}}
\end{aligned}
\qquad 3.5)
$$

the solution of Eqs. a) is

$$
a_0 = y_0 \qquad a_1 = \nabla y_1 \qquad a_2 = \nabla^2 y_2 \quad \cdots \quad a_n = \nabla^n y_n \qquad 3.6)
$$

If the coefficients are computed by hand, it is convenient to work with the format in Table 3.1 shown for $n = 4$).

x_0	y_0				
x_1	y_1	∇y_1			
x_2	y_2	∇y_2	$\nabla^2 y_2$		
x_3	y_3	∇y_3	$\nabla^2 y_3$	$\nabla^3 y_3$	
x_4	y_4	∇y_4	$\nabla^2 y_4$	$\nabla^3 y_4$	$\nabla^4 y_4$

Table 3.1

The diagonal terms $y_0, \nabla y_1, \nabla^2 y_2, \nabla^3 y_3$ and $\nabla^4 y_4$) in the table are the coefficients of the polynomial. If the data points are listed in a different order, the entries in the table will change, but the resultant polynomial will be the same—recall that a polynomial of degree n interpolating $n+1$ distinct data points is unique.

Machine computations can be carried out within a one-dimensional array **a** employing the following algorithm we use the notation $m = n+1$ = number of data points):

```
a = yData.copy )
for k in range 1,m):
    for i in range k,m):
        a[i] =  a[i] - a[k-1])/ xData[i] - xData[k-1])
```

Initially, **a** contains the y-coordinates of the data, so that it is identical to the second column in Table 3.1. Each pass through the outer loop generates the entries in the next column, which overwrite the corresponding elements of **a**. Therefore, **a** ends up containing the diagonal terms of Table 3.1, i.e., the coefficients of the polynomial.

newtonPoly

This module contains the two functions required for interpolation by Newton's method. Given the data point arrays `xData` and `yData`, the function `coeffts` returns the coefficient array **a**. After the coefficients are found, the interpolant P_n x) can be evaluated at any value of x with the function `evalPoly`.

```
## module newtonPoly
''' p = evalPoly a,xData,x).
    Evaluates Newton's polynomial p at x. The coefficient
    vector {a} can be computed by the function 'coeffts'.

    a = coeffts xData,yData).
    Computes the coefficients of Newton's polynomial.
'''
def evalPoly a,xData,x):
    n = len xData) - 1  # Degree of polynomial
    p = a[n]
    for k in range 1,n+1):
        p = a[n-k] +  x -xData[n-k])*p
    return p

def coeffts xData,yData):
    m = len xData)  # Number of data points
    a = yData.copy )
    for k in range 1,m):
        a[k:m] =  a[k:m] - a[k-1])/ xData[k:m] - xData[k-1])
    return a
```

Neville's Method

Newton's method of interpolation involves two steps: computation of the coefficients, followed by evaluation of the polynomial. This works well if the interpolation is carried out repeatedly at different values of x using the same polynomial. If only one point is to interpolated, a method that computes the interpolant in a single step, such as Neville's algorithm, is a better choice.

Let $P_k[x_i, x_{i+1}, \ldots, x_{i+k}]$ denote the polynomial of degree k that passes through the $k+1$ data points $x_i, y_i), x_{i+1}, y_{i+1}), \ldots, x_{i+k}, y_{i+k})$. For a single data point, we have

$$P_0[x_i] = y_i \qquad 3.7)$$

The interpolant based on two data points is

$$P_1[x_i, x_{i+1}] = \frac{x - x_{i+1})\, P_0[x_i] + \; x_i - x)\, P_0[x_{i+1}]}{x_i - x_{i+1}}$$

It is easily verified that $P_1[x_i, x_{i+1}]$ passes through the two data points; that is, $P_1[x_i, x_{i+1}] = y_i$ when $x = x_i$, and $P_1[x_i, x_{i+1}] = y_{i+1}$ when $x = x_{i+1}$.

The three-point interpolant is

$$P_2[x_i, x_{i+1}, x_{i+2}] = \frac{x - x_{i+2})\, P_1[x_i, x_{i+1}] + \; x_i - x)\, P_1[x_{i+1}, x_{i+2}]}{x_i - x_{i+2}}$$

To show that this interpolant does intersect the data points, we first substitute $x = x_i$, obtaining

$$P_2[x_i, x_{i+1}, x_{i+2}] = P_1[x_i, x_{i+1}] = y_i$$

Similarly, $x = x_{i+2}$ yields

$$P_2[x_i, x_{i+1}, x_{i+2}] = P_1[x_{i+1}, x_{i+2}] = y_{i+2}$$

Finally, when $x = x_{i+1}$ we have

$$P_1[x_i, x_{i+1}] = P_1[x_{i+1}, x_{i+2}] = y_{i+1}$$

so that

$$P_2[x_i, x_{i+1}, x_{i+2}] = \frac{x_{i+1} - x_{i+2})\, y_{i+1} + \; x_i - x_{i+1})\, y_{i+1}}{x_i - x_{i+2}} = y_{i+1}$$