Polynomial Interpolation

1. The Vector Space of Polynomials.

Here is a collection of handy facts about polynomials that can help to cut fog in some of one’s thinking.

1.A. Long division of polynomials is possible: if \( D(x) \) and \( P(x) \) are polynomials, then one can divide \( D(x) \) into \( P(x) \) by the long-division algorithm famous from grade-school courses in arithmetic.\(^{(1)}\) The result is that one writes

\[
P(x) = Q(x) \cdot D(x) + R(x)
\]

“\( P(x) \) equals the quotient times the divisor plus the remainder”, where \( 0 < \deg(R(x)) < \deg(P(x)) \). (If the degree of the divisor is larger than that of the dividend, the quotient is zero and the remainder equals the divisor, but more than that happens in the interesting cases.)

1.B. Long division by a linear divisor is interesting in particular. For any (real or complex) number \( \alpha_1 \) one can long-divide \((x - \alpha_1)\) into a given \( P(x) \), and the division will come out

\[
P(x) = (x - \alpha_1)Q(x) + R \quad \deg(Q(x)) = \deg(P(x)) - 1
\]

where \( R \) is either a polynomial of degree 0, i.e., a number (a nonzero constant), or the identically-zero polynomial (which is a polynomial that either is not assigned a degree or is assigned the degree \(-\infty\)). To say the remainder is zero is to say that the division “came out even”, i.e., that \( P(x) = (x - \alpha_1)Q(x) \). This division comes out even if and only if \( P(\alpha_1) = 0 \), i.e., \( \alpha_1 \) is a root of \( P(x) = 0 \): plugging \( \alpha_1 \) in for the \( x \) in the result of the division gives

\[
P(\alpha_1) = (\alpha_1 - \alpha_1)Q(\alpha_1) + R = 0 + R = R
\]

and the extreme l. h. s. is zero—i.e., \( \alpha_1 \) is a root—if and only the extreme r. h. s. is zero, i.e., \( R = 0 \), there is no remainder, and \((x - \alpha_1)\) divided \( P(x) \) evenly. We also see that the remainder can be computed without long division, simply by plugging \( \alpha_1 \) into \( P(x) \). These facts are collectively known as the

Remainder Theorem: If \( P(x) = K[x] \) is a polynomial with coefficients in a field \( K \), then the remainder of \( P(x) \) on long-division by \((x - \alpha_1)\) is the number (= polynomial of degree \( \leq 0 \)) given by \( R = P(\alpha_1) \in K \). In particular, \((x - \alpha_1)\) divides \( P(x) \) if and only if \( P(\alpha_1) = 0 \).

The result “\( b_0 = p(z) \)” of nested multiplication or Horner’s method as presented by Atkinson on pp. 96–97 is essentially the remainder theorem.

1.C. The number of distinct roots is bounded by the degree.\(^{(2)}\) If \( P(x) = a_0 + a_1x + \cdots + a_nx^n \) is a polynomial of degree at most \( n \), with coefficients in a field \( K \), then either all its coefficients are zero or else there can be no more than \( n \) different numbers \( \alpha_k \) for which \( P(\alpha_k) = 0 \), i.e., the equation \( P(x) = 0 \) can have at most \( n \) roots. This is true even for roots in a “bigger” field; e.g., a polynomial of degree \( n \) with coefficients belonging to \( \mathbb{R} \) can have at most \( n \) distinct roots in \( \mathbb{C} \). The reason is that if \( P(x) \) is constant—i.e., \( P(x) \equiv a_0 \)—then it has any roots at all if and only if that constant is zero. Otherwise \( n = \deg(P(x)) > 0 \) \((a_0 \neq 0 \text{ and } n \geq 1)\), and then one may successively apply the remainder theorem as many times as one has roots: at each point one gets a polynomial (quotient) of smaller degree. One has successively

\[
P(x) = (x - \alpha_1) \cdot Q_1(x) + 0
\]

\[
0 = P(\alpha_2) = (\alpha_2 - \alpha_1) \cdot Q_1(\alpha_2) \quad \text{so} \quad \alpha_2 \text{ is a root of } Q_1(x)
\]

\[
Q_1(x) = (x - \alpha_2) \cdot Q_2(x) + 0
\]

\[
0 = Q_2(\alpha_3) = (\alpha_3 - \alpha_2) \cdot Q_2(\alpha_3) \quad \text{so} \quad \alpha_3 \text{ is a root of } Q_2(x)
\]

\[
\cdots
\]

\[
\cdots
\]

\[
\cdots
\]

\(^{(1)}\) But soon to be extinct, if the “mathematical education experts” get their way. After all, if you want to divide one number into another, all you have to do is plug them into your calculator and push the \( \div \) key. “But then how shall we know how to do long division of polynomials, for we know no elementary analogue?” Glad you asked that question . . .

\(^{(2)}\) This statement holds even over finite fields, which is why we state things for “a field \( K \)” rather than just for real or complex numbers.
At each step, the degree of the new quotient is one smaller than: first the degree \( n \) of \( P(x) \), then the degree of the old quotient. Therefore, this process can be repeated at most \( n \) times—so a list of distinct roots of \( P(x) = 0 \) can contain at most \( n \) numbers.

1.D. Roots can have multiplicity. Here we restrict our considerations to polynomials with real or complex coefficients.\(^{(3)}\) Observe that if \( \deg(P(x)) \leq n \) then its derivatives \( P^{(k)}(x) = 0 \) for \( k > n \), and therefore the Taylor-series-with-remainder

\[
P(x) = P(\alpha_1) + \frac{P'(\alpha_1)}{1!}(x-\alpha_1) + \cdots + \frac{P^{(n)}(\alpha_1)}{n!}(x-\alpha_1)^n + \frac{P^{(n+1)}(\xi)}{(n+1)!}(x-\alpha_1)^{n+1}
\]

always has a zero remainder term. (If you prefer a complex-variables approach, then \( P(z) \) is a complex-differentiable function on all of \( C \) and therefore is an entire function, so its Taylor series about \( \alpha_1 \) converges to \( P(z) \) everywhere in \( C \); but its Taylor series has at most its first \((n+1)\) terms nonzero, again because \( P^{(k)}(z) = 0 \) for \( k > n \).) Either way, we have

\[
P(x) \equiv P(\alpha_1) + \frac{P'(\alpha_1)}{1!}(x-\alpha_1) + \cdots + \frac{P^{(n)}(\alpha_1)}{n!}(x-\alpha_1)^n.
\]

Now \( \alpha_1 \) is a root of \( P(x) = 0 \) if and only if the index-0 term of that Taylor expansion is zero. It may happen that \( \alpha_1 \) is also a root of some derivatives of \( P(x) \): if \( P(\alpha_1), P'(\alpha_1), \ldots, P^{(k-1)}(\alpha_1) = 0 \) but \( P^{(k)}(\alpha_1) \neq 0 \), then \( \alpha_1 \) is said to be a zero or root of order \( k \) or of multiplicity \( k \). Evidently when this happens one can write

\[
P(x) \equiv (x-\alpha_1)^k \cdot \left[ \frac{P^{(k)}(\alpha_1)}{k!} + \cdots + \frac{P^{(n)}(\alpha_1)}{n!}(x-\alpha_1)^{n-k} \right] = (x-\alpha_1)^k Q(x) \quad \text{where} \quad Q(\alpha_1) \neq 0,
\]

\( Q(x) \) being a polynomial of degree \( n-k \). The converse is also true: if one applies Leibniz’ formula for \( j = 1, \ldots, k \), that is

\[
\frac{d^j}{dx^j} [g(x) \cdot h(x)] = g^{(j)}(x)h(x) + \binom{j}{1} g^{(j-1)}(x)h^{(1)}(x) + \cdots + \binom{j}{j-1} g^{(1)}(x)h^{(j-1)}(x) + g(x)h^{(j)}(x)
\]

to the case \( g(x) = (x-\alpha_1)^k \), \( h(x) = Q(x) \) where \( Q(\alpha_1) \neq 0 \), one sees that if \( P(x) \equiv (x-\alpha_1)^k Q(x) \) where \( Q(\alpha_1) \neq 0 \), then \( P(\alpha_1), P'(\alpha_1), \ldots, P^{(k-1)}(\alpha_1) = 0 \) but \( P^{(k)}(\alpha_1) = k! \cdot Q(\alpha_1) \neq 0 \). If we now repeat the “successive-division”argument at the end of the last paragraph, making the obvious changes, we find that if \( \alpha_1, \ldots, \alpha_r \) is a list of the roots of \( P(x) = 0 \), each \( \alpha_j \) with multiplicity \( k_j \geq 1 \), then

\[
P(x) \equiv (x-\alpha_1)^{k_1} \cdots (x-\alpha_r)^{k_r} Q_r(x)
\]

where \( Q(x) \) has no roots. If we agree to look for roots in the field \( \mathbb{C} \), then by the Fundamental Theorem of Algebra, this can happen only if \( Q(x) \) is a constant. Comparing coefficients of \( x^n \) tells us the constant is the leading coefficient \( a_n \) in the standard representation \( P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \), so we have the complete factorization into linear factors

\[
P(x) \equiv a_n \cdot (x-\alpha_1)^{k_1} \cdots (x-\alpha_r)^{k_r}
\]

with each factor counted according to multiplicity. By comparing degrees on both sides, we see that \( \sum_{i=1}^r k_i = n \).

1.E. The coefficients of a real or complex polynomial are determined by its values as a function. If \( I \) is an interval of real numbers that contains more than a single point—open, closed, half-line or whatever—then it contains infinitely many points. Therefore, the coefficients of the polynomial \( P(x) \)

\(^{(3)}\) The algebra gets a bit more involved when one can’t use analysis, because relations like \((\beta-\alpha)^p = \beta^p - \alpha^p\) hold in fields of characteristic \( p \). These matters are of no concern in this course but can be of interest in such fields as coding theory.
are uniquely determined by its values as a function: if two polynomials $P(x) = a_0 + a_1 x + \cdots + a_n x^n$ and $Q(x) = b_0 + b_1 x + \cdots + b_n x^n$ take the same values at all points of an interval $I$, then $P(x) - Q(x) = (a_0 - b_0) + (a_1 - b_1) x + \cdots + (a_n - b_n) x^n$ has all of the infinitely many points of $I$ as roots; but it can only have $n$ roots without having all its coefficients equal to zero, so they are all zero, i.e., each $a_j$ equals its corresponding $b_j$. (Again, another way to look at this would be to observe that if $\deg(P(x)) \leq n$ then its derivatives $P^{(k)}(x) \equiv 0$ for $k > n$, and therefore the Taylor-series-with-remainder

$$P(x) = P(0) + \frac{P'(0)}{1!} x + \cdots + \frac{P^{(n)}(0)}{n!} x^n + \frac{P^{(n+1)}(\xi)}{(n+1)!} x^{n+1}$$

always has a zero remainder term, so that the values of $P(x)$ near zero determine the coefficients of $P(x)$; look at it from this point of view if you must, but one doesn’t need calculus to get this result [and one doesn’t have to work near zero].) Indeed—and we shall use this fact shortly—if $\{x_0, \ldots, x_n\}$ are $n + 1$ distinct points of $I$ (note the numbering scheme, which uses the numbers from 0 to $n$ to index $n + 1$ points) then the values of $P(x)$ at those $n + 1$ points determine the coefficients in $P(x) = a_0 + a_1 x + \cdots + a_n x^n$ uniquely. (The proof is the same: a polynomial of degree at most $n$ can’t have $n + 1$ or more roots unless it is identically zero.)

1.F. We know the dimension of the vector spaces of polynomials. The set $\mathcal{P}_n$ of all polynomials of degree $\leq n$ is a vector space of dimension $n + 1$ in a natural way, in view of the obvious correspondence $(a_0, \ldots, a_n) \mapsto a_0 + a_1 x + \cdots + a_n x^n$. As we just saw, the elements of $\mathcal{P}_n$ can be identified with the functions they determine on any interval with infinitely many points $I$ (or, indeed, on any interval with $n + 1$ or more points). Because the coefficients of $P(x)$ are uniquely determined, this set is a finite-dimensional vector space(4) of functions with basis $\{1, x, x^2, \ldots, x^n\}$, and therefore $\dim(\mathcal{P}_n) = n + 1$. It follows that any set of $n + 1$ polynomials of degree $\leq n$ that is either linearly independent or that spans $\mathcal{P}_n$ will be a basis of that space, and the coefficients in any expansion of a polynomial $P(x)$ into a linear combination of elements of that set will be uniquely determined by $P(x)$. This fact is not merely handy; it is fundamental.

An easy way to recognize a certain kind of basis (not the only possibility: cf. the Lagrange interpolating polynomials) of $\mathcal{P}_n$ is the following

**Proposition:** If $\mathcal{B} = \{Q_0(x), \ldots, Q_n(x)\}$ is a(n indexed) family of polynomials such that the degree of $Q_k(x)$ is exactly $k$ for $k = 0, \ldots, n$, then $\mathcal{B}$ is a basis of $\mathcal{P}_n$.

**Proof.** By induction on $n$: if $n = 0$ then $Q_0(x)$ is some nonzero constant, and obviously $\mathcal{P}_0$ consists of the constants, each of which is a multiple of $Q_0$. Suppose the assertion is known for $\mathcal{P}_{n-1}$. Let $P(x) \in \mathcal{P}_n$ be given, so

$$P(x) = a_0 + a_1 x + \cdots + a_n x^n.$$  

The polynomial $Q_n(x) = b_0 + \cdots + b_n x^n$ has $b_n \neq 0$, so we can divide by $b_n$ and write for suitable coefficients $c_j$ (the values of the coefficients are not important—what matters is the fact that there is no $x^n$ term)

$$P(x) - \frac{a_n}{b_n} Q_n(x) = c_0 + \cdots + c_{n-1} x^{n-1} \in \mathcal{P}_{n-1}.$$

By the induction hypothesis we can write for suitable coefficients $h_j$

$$P(x) - \frac{a_n}{b_n} Q_n(x) = c_0 + \cdots + c_{n-1} x^{n-1} = h_0 Q_0 + \cdots + h_{n-1} Q_{n-1}(x)$$

$$P(x) = h_0 Q_0 + \cdots + h_{n-1} Q_{n-1}(x) + \frac{a_n}{b_n} Q_n(x)$$

so $\mathcal{B}$ spans $\mathcal{P}_n$, and since $\mathcal{B}$ has $n + 1$ elements it “has the right number of elements to be a basis” and must therefore be a basis of $\mathcal{P}_n$. (Recall the Theorem: if $V$ is a vector space of dimension $s$, then for an indexed family $V = \{v_1, \ldots, v_k\} \subseteq V$ to be a basis of $V$ it is sufficient either that $V$ spans $V$ and $k \leq s$ or that $V$ is linearly independent and $k \geq s$. Of course in either case, then $k = s$.)

(4) With the field $\mathbb{K}$ as scalars. Note that $\mathcal{P}_n$ is *not* a ring (or algebra): the product of two of its elements may have degree $> n$. 

3
2. Polynomial Interpolation, First at Distinct Points.

This material is basically that of Atkinson’s Theorem 3.1 and what follows it, looked at from perhaps a slightly different viewpoint.

2.A. Solutions of the interpolation problem exist. First, let’s beat the polynomial-interpolation problem to death with theory, just to make sure it’s possible to do it. We claim that the following is a

**Theorem:** Let \( \{x_0, \ldots, x_n\} \) be an indexed family of \( n+1 \) distinct points of \( \mathbb{R} \) or \( \mathbb{C} \) and \( \{f_0, \ldots, f_n\} \) be a corresponding set of numbers of the same scalar field. Then there is a unique polynomial \( P(x) \in \mathbb{P}_n \) for which \( P(x_j) = f_j \) for each index \( j = 0, \ldots, n \).

**Proof.** This is a variant of Atkinson’s Proof (ii) on pp. 132–133. To keep our ideas concrete, let’s assume the scalars are \( \mathbb{R} \). Consider the map from the real vector space \( \mathbb{P}_n \) to \( \mathbb{R}^{n+1} \) given by

\[
A : \mathbb{P}_n \rightarrow (P(x_0), P(x_1), \ldots, P(x_n))^T
\]

As we repeatedly saw above, if a polynomial \( P(x) \in \mathbb{P}_n \) has \( n+1 \) distinct roots, then it is the zero polynomial. Clearly \( A(P) = (0, \ldots, 0)^T = 0 \in \mathbb{R}^{n+1} \) if and only if all of the \( n+1 \) points \( \{x_0, \ldots, x_n\} \) are roots of \( P(x) = 0 \), and thus \( \text{Ker} A \) consists only of the zero polynomial: \( \text{Ker} A = \{0\} \). By the fundamental relation \( \dim \mathbb{P}_n = \dim \text{Im} A + \dim \text{Ker} A \) we have \( n+1 = \dim \mathbb{P}_n = \dim \text{Im} A \). Thus \( \text{Im} A \subseteq \mathbb{R}^{n+1} \) is a subspace of \( \mathbb{R}^{n+1} \) whose dimension is \( n+1 \), and therefore \( \text{Im} A = \mathbb{R}^{n+1} \). In particular, \( (f_0, \ldots, f_n)^T \in \text{Im} A \), a fact which says precisely that there exists a polynomial \( P(\cdot) \in \mathbb{P}_n \) with \( AP = (f_0, \ldots, f_n)^T \), i.e., \( P(x_j) = f_j \) for \( j = 0, \ldots, n \). (The same argument would be valid for any scalar field including the finite fields, although in this course we only consider \( \mathbb{R} \) and \( \mathbb{C} \).) In view of \( \text{Ker} A = \{0\} \), we already knew that the solution of the problem was unique.

The situation in which one typically uses an interpolating polynomial of the kind that theorem tells us exists is the following. One has a function \( f(x) \) defined on an interval \([a, b]\) of the real line, one has \( n+1 \) points \( \{x_0, \ldots, x_n\} \subseteq [a, b] \), and one wants a polynomial \( P(x) \in \mathbb{P}_n \) for which \( P(x_j) = f(x_j) \) holds for all \( j = 0, \ldots, n \); in other words, the \( f_j \)’s are the values of \( f(x) \) at the corresponding \( x_j \)’s. (The unspoken hope is that \( P(x) \) can be used as an approximation to \( f(x) \) for all \( x \in [a, b] \), and we shall investigate the goodness of approximation shortly.)

2.B. Lagrange Interpolating Polynomials. This is the approach that Atkinson takes to exhibit polynomial interpolators explicitly in his Proof (iii) on p. 133. While these are easy to think with, they are not easy to compute with, for reasons that we shall explore below.

Let \( \{x_0, \ldots, x_n\} \) be the usual \( n+1 \) points (“nodes”) in an interval \( I \) in which a function \( f(x) \) is defined. Lagrange’s interpolation idea goes as follows: suppose that for each particular \( x_k \) of our set of nodes we could construct a polynomial \( \ell_k(x) \in \mathbb{P}_n \) with the property that

\[
\ell_k(x) = \begin{cases} 
1 & \text{if } x = x_k \\
0 & \text{if } x = x_i, \text{ } i \neq k 
\end{cases}
\]

Then simply writing

\[
P_n(x) = \sum_{k=0}^{n} f(x_k)\ell_k(x)
\]

would produce a polynomial whose value at \( x = x_i \) would be \( f(x_i) \), because for that value of \( x \) all the terms of the sum would equal zero except the \( i \)-th, and that one would equal \( f(x_i) \cdot 1 = f(x_i) \). Constructing these polynomials is easy: first, for each \( 0 \leq k \leq n \), write a polynomial that has roots at and only at each \( x_i \) with \( i \neq k \). Except for the first and last \( k \), its form could be taken as

\[
(x - x_0) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)
\]
Thus for \( x \neq x_i \) for any \( i \) we can write

\[
P_n(x) = \Psi(x) \cdot \sum_{k=0}^{n} \frac{f(x_k)}{(x - x_k) \Psi'(x_k)}.
\]

The \( \Psi'(x_k) \)'s can be computed once and stored (doing work comparable to the work required to compute divided differences), so evaluation of \( P_n(x) \) would require an evaluation of \( \Psi(x) \) \((n + 1)\) multiplications) and a multiplication and division for each term of the sum.

Unfortunately, even this amount of plastic surgery does not help the fact that when the sum in a Lagrange-polynomial expansion takes a small value, it typically does it by large cancellation. The reader can check that when the polynomial \( 2x^2 - 1 \) is expressed in Lagrange polynomials with nodes \([-1, 0, +1]\) in the form

\[
2x^2 - 1 = \left[ x^3 - x \right] \cdot \left[ \frac{1}{2(x + 1)} + \frac{1}{x} + \frac{1}{2(x - 1)} \right]
\]

(5) Some authors, for example Burden & Faires, denote the same polynomial we called \( \Psi(x) \) by the name \( \omega(x) \), and it has even been called \( w(x) \), or \( w_n(x) \) when the degree of the interpolating polynomial was important. Such names as \( \Psi(x) \) or \( \Psi_n(x) \), which Atkinson uses, are closer to being standard in the literature. In this context we should probably note that the Lagrange polynomial \( \ell_k(x) \) is occasionally called \( L_k(x) \) or even \( \ell_k(x) \) (hard to read!) by various authors. You may have to go back and re-read the definitions when you read an unfamiliar author in numerical analysis.
and evaluated with 4-decimal-digit arithmetic at $x = 0.002$, the fourth digit is lost. The Newton form of the same polynomial

$$2x^2 - 1 = 1 - 2(x + 1) + 2(x + 1)x$$

gives four-digit accuracy. For reasons like this, most professional numerical analysts caution against relying on Lagrange polynomials for computational purposes. They have other disadvantages: if you add a node to \(\{x_0, \ldots, x_n\}\) you have to produce the \(\{\ell_k(x)\}_{k=0}^{n+1}\) all over again. Some computation-by-recursion can be done by the use of Neville’s method, which we shall use briefly in connection with showing how divided-difference tables are constructed; a fuller discussion of it can be found, e.g., in Burden & Faires (6th ed.), pp. 116–119.

Unfortunately, many numerical analysts caution against the use of Neville’s method: it is expensive in terms of the number of operations required to implement it (count operations in Burden & Faires’ Algorithm 3.1!) and since it gets its accuracy by cancellation, it is subject to disturbance by roundoff error. On the other hand, if the reader looks at the way Neville’s method operates when we discuss divided-difference tables below, (s)he will see that the basic idea is the same one that gives us the recursion

$$f[x_0, \ldots, x_k] = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}.$$  

Whether the Lagrange method or the (superior) Newton method is used to find an interpolating polynomial, the interpolating polynomial is uniquely determined by the interpolation problem. Thus the error term in

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

(a full derivation of which is given in Atkinson’s Theorem 3.2 pp. 134–136, and a representative special case of which we shall give below) is the same whether one has obtained \(P_n(x)\) in the form

$$\sum_{k=0}^{n} f[x_0, \ldots, x_k](x - x_0) \cdots (x - x_{k-1})$$

in which Newton interpolation produces it, or in the Lagrange form \(\sum_{k=0}^{n} f(x_k) \ell_k(x)\). Note, by the way, that the polynomial \(\Psi(x) = (x - x_0) \cdots (x - x_n)\) introduced above occurs in the error formula, which one may rewrite as

$$f(x) = P_n(x) + \frac{f^{(n+1)}(\xi)}{(n + 1)!}\Psi(x).$$

This is the basic reason for referring to \(\Psi(x)\) as the error factor. As a preview of coming attractions, the reader might consider the following problem: given a function \(f(x)\) defined on—for example—\([-1, 1]\), is there a “good choice of the nodes \(\{x_0, \ldots, x_n\}\)” that would minimize the number

$$\max \{|\Psi(x)|: -1 \leq x \leq 1\}?$$

Such a choice of nodes would make the worst case of the error factor as small as possible, and thus tend to minimize the error made by approximating \(f(x)\) by the interpolating polynomial \(P_n(x)\). (Of course the answer is “yes”, and the points are not equally spaced—in fact, equal spacing can result in increasingly bad interpolation error as the number of nodes increases. The standard example [due to the Runge-Kutta methods] is the function \(f(x) = 1/(x^2 + 1)\) on the interval \([-5, 5]\): as the number of equally spaced interpolation points increases, the maximum error on the interval actually diverges to \(\infty\)! The details of an approach to this fact using real-variable methods are given by Isaacson & Keller (op. cit, p. 275 ff.); it would be inappropriate to work through them in class, but everyone should be aware of the perils of interpolation at large numbers of equally-spaced nodes. A somewhat more comprehensible complex-variable approach to the Runge example can be found in J. F. Epperson, On the Runge Example, Amer. Math. Monthly 94 (1987), pp. 329–341.)

---

Thus our new polynomial will be

\[ P_0(x) = f_0 \quad (n = 0) \]

that will give us a new polynomial whose value at \( x_0 \) is not changed but whose value at \( x_1 \) will be \( f_1 \). In order not to change the value at \( x_0 \) we use a multiple of the polynomial \( (x - x_0) \in \mathcal{P}_1 \), which is zero at \( x_0 \). Thus our new polynomial will be

\[
P_1(x) = P_0(x) + a_1(x - x_0) = f_0 + a_1(x - x_0)
\]

for some choice of \( a_1 \). But there is no “choice”, since the interpolating polynomial is unique; indeed, we can set \( x = x_1 \) and solve for \( a_1 \):

\[
f_1 = P_1(x_1) = f_0 + a_1(x_1 - x_0)
\]

\[
f_1 = P_1(x_1) = f_0 + a_1(x_1 - x_0)
\]

\[
\frac{f_1 - f_0}{x_1 - x_0} = a_1. \quad (n = 1)
\]

Again, we may try to go on to the next point \( x_2 \) by adding a single term: in order not to change the value on \( \{x_0, x_1\} \) we use a multiple of the polynomial \( (x - x_0)(x - x_1) \in \mathcal{P}_2 \), which is zero on \( \{x_0, x_1\} \). Thus our new polynomial will be

\[
P_2(x) = P_1(x) + a_2(x - x_0)(x - x_1) = f_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)
\]

where \( a_1 \) was determined in the previous step. It is automatic that \( P_2(x_j) = f_j \) for \( j = 0, 1 \), and to get things to come out right at \( x_2 \) we need (again we can divide by \( (x_2 - x_0)(x_1 - x_0) \) under the assumption that all the \( x_j \) were distinct)

\[
f_2 = P_2(x_2) = P_1(x) + a_2(x - x_0)(x - x_1) = f_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)
\]

\[
(f_2 - f_0) - a_1(x_2 - x_0) = a_2(x_2 - x_0)(x_2 - x_1)
\]

\[
\frac{f_2 - f_0}{x_2 - x_0} - \frac{f_1 - f_0}{x_1 - x_0} = a_2 \quad (n = 2)
\]

which doesn’t look too nice (at this point) but is still quite easily computed.

It is evident at this point that nothing obstructs our continuing in this fashion until we use up all the data. For the formal induction step, suppose that we have found coefficients \( \{a_1, a_2, \ldots, a_k\} \) such that the polynomial of degree \( \leq k \)

\[
P_k(x) = f_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_k(x - x_0)(x - x_1) \cdots (x - x_{k-1}) \in \mathcal{P}_k
\]

has the property \( P_k(x_j) = f_j \) for \( j = 0, 1, \ldots, k \). The polynomial \( (x - x_0)(x - x_1) \cdots (x - x_k) \) of degree \( k + 1 \) takes the value zero for \( x = x_0, \ldots, x_k \) but is not zero for \( x = x_{k+1} \) and therefore it is possible (and simple) to solve the equation

\[
f_{k+1} = f_0 + a_1(x_{k+1} - x_0) + a_2(x_{k+1} - x_0)(x_{k+1} - x_1) + \cdots + a_k(x_{k+1} - x_0)(x_{k+1} - x_{k-1}) + a_{k+1}(x_{k+1} - x_0) \cdots (x_{k+1} - x_k)
\]

for the unknown coefficient \( a_{k+1} \). The polynomial of degree at most \( k + 1 \) obtained by adding this new term to \( P_k(x) \),

\[
P_{k+1}(x) = P_k(x) + a_{k+1}(x - x_0) \cdots (x - x_k) \in \mathcal{P}_{k+1}
\]
does not lose the property that its value at \( x_j \) is \( f_j \), for \( j = 0, \ldots, k \), but gains the property that its value at \( x_{k+1} \) is \( f_{k+1} \). So that is the induction (or recursion) step, and we simply repeat this step at \( k = 0, \ldots, n - 1 \) to construct \( P_n(x) \) that satisfies \( P_n(x_j) = f_j \) for \( j = 0, \ldots, n \).

As we saw on p. 3 ff. above, the polynomials

\[
\{1, (x-x_0), (x-x_0)(x-x_1), \ldots, (x-x_0)\cdots(x-x_{n-1})\}
\]

form a basis for \( \mathcal{P}_n \), so any given polynomial \( P \in \mathcal{P}_n \) can be written in a *unique* way in the form

\[
P(x) = a_0 + a_1(x-x_0) + \cdots + a_n(x-x_0)\cdots(x-x_{n-1}).
\]

Uniqueness tells us that the \( a_k \)'s are the same ones we would get by employing the Newton construction to get a polynomial \( P_n(x) \) for which we had \( P_n(x_j) = P(x_j) \) for each \( j = 0, \ldots, n \); so these coefficients are effectively computable from the values of the given \( P(x) \) at the \( x_j \)'s. That is already a good thing to know: even though we had to indicate a lot of equation-solving in the inductive construction of the \( a_k \)'s, the equations were linear in one variable and the computation was recursive. However, something interesting happened in the course of computing the coefficients \( a_k \) that deserves our attention. At each step, the polynomial \( P_k(x) \in \mathcal{P}_k \) was the *unique* polynomial of degree \( \leq k \) that satisfies \( P_k(x_j) = f_j \) for \( j = 0, \ldots, k \). The only \( (x-x_0)\cdots(x-x_j) \) in the expansion

\[
P_k(x) = f_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_k(x-x_0)(x-x_1)\cdots(x-x_{k-1}) \quad \text{(Newton)}
\]

that had degree \( k \) is the last one: each of the others had degree strictly smaller than \( k \). Therefore, when \( P_k(x) \) is written out in terms of the standard basis of monomials

\[
P_k(x) = b_0 + b_1x + b_2x^2 + \cdots + b_kx^k \quad \text{(standard)}
\]

it must be true that \( b_k = a_k \), because only in the last term of the Newton expansion can \( P_k(x) \) pick up an \( x^k \)-term. Because \( P_k(x) \) is uniquely determined by the property that \( P_k(x_j) = f_j \) for \( j = 0, \ldots, k \) and this interpolation property is unaltered when the indices on the \( x_j \)'s and \( f_j \)'s are permuted, the same is true of \( b_k \) —but \( b_k \) is \( a_k \), so it is unaltered. In other words,

**Proposition:** The coefficient \( a_k \) in the Newton expansion of the polynomial \( P_k(x) \) that satisfies the interpolation condition \( P_k(x_j) = f_j \) for \( j = 0, \ldots, k \) is unaltered when the indices on those \( x_j \)'s and \( f_j \)'s are permuted (with the same permutation for the \( f_j \)'s as for the \( x_j \)'s, of course). Consequently, if \( P_k(x) \) is produced by interpolating the values \( f_j = f(x_j) \) of a given function at \( k + 1 \) given points, the coefficient \( a_k \) in the Newton expansion of the polynomial \( P_k(x) \) that satisfies the interpolation condition \( P_k(x_j) = f(x_j) \) for \( j = 0, \ldots, k \) is a symmetric function of \( x_0, \ldots, x_k \).

It is customary to define the symbol \( f[x_0, \ldots, x_k] \) by

\[
f[x_0, \ldots, x_k] = a_k.
\]

The expansion of the polynomial \( P_n(x) \in \mathcal{P}_n \) for which \( P_n(x_j) = f(x_j) \) for \( j = 0, \ldots, n \) then becomes

\[
P_n(x) = \sum_{k=0}^{n} f[x_0, \ldots, x_k] (x-x_0)\cdots(x-x_{k-1}).
\]

We actually computed \( a_1 \) above: it appeared as the slope of the line passing through the points \( (x_0, f(x_0)) \) and \( (x_1, f(x_1)) \) and it was obviously unchanged if \( x_0 \) and \( x_1 \) were. With \( a_2 \) the symmetry was less obvious (the reader can easily check it): but now we see that symmetry holds in general. **Note, however, the following fact which can be the source of some confusion:** if the \( x_j \)'s are permuted, the points at which interpolation is done and the Newton basis itself may change: so only the highest-order coefficient is guaranteed to be the same. For example, if we are passing a parabola through three
points with $x$-coordinates $x_0$, $x_1$, $x_2$ and we permute the points into the order $x_1$, $x_2$, $x_0$, then the Newton basis will change to $\{1, (x-x_1), (x-x_1)(x-x_2)\}$ and of course we should expect the basis coefficients (except for the last one) to be different. Indeed, the polynomial that we would have called $P_1(x)$ for the points in this new order will give a line passing through the pair of points $(x_1, f(x_1))$ and $(x_2, f(x_2))$, which is a different pair of points from the one we originally would have considered. However, we still know that $f[x_0, x_1, x_2] = f[x_1, x_2, x_0]$ because of the argument we gave about the highest-degree term of the interpolating polynomial.

2.D. It would be nice to try to minimize the amount of computation required to produce the $f[x_0, \ldots, x_k]$’s from given data. In fact, there is a neat recursive formula for them, which is derived in pretty much the same way that the two-point formula for the equation of a line in the plane is derived: the recursion can be regarded as a generalization of that formula. Recall that if one wants to pass a line through the points $(x_0, f_0)$ and $(x_1, f_1)$, one produces the two linear functions of $x$

$$\frac{x-x} {x_1-x_0} \quad \text{and} \quad \frac{x-x_0} {x_1-x_0}$$

that tell one in what proportion the point $x$ divides the interval $[x_0, x_1]$. Then one simply “puts the r. h. s. of the equation of the line together in the same proportion”:

$$P_1(x) = \frac{x-x} {x_1-x_0} f_0 + \frac{x-x_0} {x_1-x_0} f_1.$$  \hfill (line)

Now, suppose one has $k+1$ points $x_0, x_1, \ldots, x_k$ and one has two interpolating polynomials of degree $\leq k$ each for a function $f(x)$, say their names are $P_{k-1}$ and $Q_{k-1}$, satisfying the following interpolation conditions:

$$P_{k-1}(x_j) = f(x_j) \quad \text{for} \quad j = 0, \ldots, k-1;$$

$$Q_{k-1}(x_j) = f(x_j) \quad \text{for} \quad j = 1, \ldots, k.$$  

If one uses the approach of “(line)” above, but uses it on functions, and forms

$$P_k(x) = \frac{x-x} {x_k-x_0} P_{k-1}(x) + \frac{x-x_0} {x_k-x_0} Q_{k-1}(x),$$  \hfill (polynomial)

what happens? First of all, the resulting polynomial $P_k(x)$ is an element of $\mathcal{P}_k$: the degree of each term is at most $(k-1)+1=k$. Second, $P_k(x_j) = f(x_j)$ for all $j = 0, 1, \ldots, x_k$, because: if $x = x_0$, the second term in the definition of $P_k(x_0)$ drops out and the value is just $[(x-x_0)/(x_k-x_0)] \cdot P_{k-1}(x_0) = f(x_0)$; if $x = x_k$, the first term in the definition of $P_k(x_k)$ drops out and the value is just $[(x_k-x_0)/(x_k-x_0)] \cdot Q_{k-1}(x_k) = f(x_k)$; and in all the remaining cases we have both $P_{k-1}(x_j) = Q_{k-1}(x_j) = f(x_j)$, so

$$P_k(x_j) = \frac{x-x_0} {x_k-x_0} f(x_j) + \frac{x-x_0} {x_k-x_0} f(x_j) = \left(\frac{x-x_j} {x_k-x_0} \right) f(x_j) = 1 \cdot f(x_j).$$

Well, then, $P_k(x) \in \mathcal{P}_k$ is the unique polynomial of degree $\leq k$ that does that interpolation. Consequently we have

$$P_k(x) = \sum_{i=0}^k f[x_0, x_1, \ldots, x_i](x-x_0) \cdots (x-x_i-1)$$

$$= f[x_0, \ldots, x_k] x^k + \text{(lower order terms)}.$$
On the other hand, we have just produced a competing expansion

\[
P_k(x) = \frac{x_k - x}{x_k - x_0} P_{k-1}(x) + \frac{x - x_0}{x_k - x_0} Q_{k-1}(x)
\]

\[
= \frac{1}{x_k - x_0} \left[ (x_k - x) \sum_{i=0}^{k-1} f[x_0, \ldots, x_i](x - x_0) \cdots (x - x_{i-1}) \\
+ (x - x_0) \sum_{i=1}^{k} f[x_1, \ldots, x_i](x - x_1) \cdots (x - x_{i-1}) \right]
\]

\[
= \frac{1}{x_k - x_0} \left[ - \sum_{i=0}^{k-1} f[x_0, \ldots, x_i](x - x_0) \cdots (x - x_{i-1})(x - x_k) \\
+ \sum_{i=1}^{k} f[x_1, \ldots, x_i](x - x_0)(x - x_1) \cdots (x - x_{i-1}) \right].
\]

Look at the unique term in each of those sums that can possibly have degree \( k \), namely the one with \( i = k - 1 \) in the first sum and the one with \( i = k \) in the second. Clearly

\[
P_k(x) = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0} x^k + \text{(lower order terms)}.
\]

Comparing these two ways of writing the highest-degree term of \( P_k(x) \), we see that

\[
f[x_0, \ldots, x_k] = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}.
\]

This relation makes computation of divided differences easy (which made a difference in the time of hand computation) and cheap in the sense of requiring little computation and storage (which makes a difference now). The format for the computation is the divided difference table. Given points \( \{x_0, \ldots, x_n\} \) and a function \( f(x) \) to be interpolated at them, we can produce a triangular table of all its divided differences at these points in the following way. The table will have the appearance of a matrix

\[
\begin{array}{cccccc}
  x_0 & x_1 & x_2 & \cdots & x_{n-1} & x_n \\
  0 & f(x_0) & f(x_1) & f(x_2) & \cdots & f(x_{n-1}) & f(x_n) \\
  1 & f[x_0, x_1] & f[x_1, x_2] & f[x_2, x_3] & \cdots & f[x_{n-1}, x_n] \\
  2 & f[x_0, x_1, x_2] & f[x_1, x_2, x_3] & f[x_2, x_3, x_4] & \cdots & \\
  \vdots & & & & & \\
  n & f[x_0, \ldots, x_n] \\
\end{array}
\]

The rows are labeled by the orders of the divided differences; the columns are labeled by the indices (or, equivalently, by the \( x_i \)’s). The rows are formed recursively according to the following rule: the zero-th row contains the function values in order. After that, the \( j \)-th entry in the \( i \)-th row is formed by taking the \( (j + 1) \)-st entry in the \( (i - 1) \)-st row, subtracting from it the \( j \)-th entry in the \( (i - 1) \)-st row, and dividing the difference by \( x_{i+j} - x_j \). It is easy to see that we have labeled the first row in the matrix correctly: for \( i = 1 \), the recipe for the \( i = 1 \)-st row puts the definition of \( f[x_j, x_{j+1}] \) in the \( j \)-th column. “Recursion” above guarantees that for every \( i = 0, \ldots, n \) the entry in the \( j \)-th column of the \( i \)-th row will be \( f[x_i, \ldots, x_{i+j}] \), although there will only be enough data to compute it if \( j \leq n - i \)—which is why the table winds up triangular. (By the way, if one wants to add another interpolation point—call it \( x_{n+1} \), of course—one just extends the already-computed table by one row and one column, puts \( f(x_{n+1}) \) in the \((0, n+1)\) position, and
then updates each row working downward in exactly the same way that the original table was constructed.)

The coefficients of the expansion

\[ P_n(x) = \sum_{i=0}^{n} f[x_0, \ldots, x_i] (x - x_0) \cdots (x - x_{i-1}) \]

appear as the entries in the 0-th column when the work is arranged as we did it above.

There are various ways of arranging the work of constructing a divided-difference table. The one we gave above is well adapted to machine computation—the table is an array, and the recursion that computes the differences is easy to code with nested do-loops (there are other ways to code it, of course). Atkinson’s algorithm on p. 141 is very economical of storage because it overwrites intermediate results. For an algorithm in which intermediate results are retained, see Burden & Faires’ Algorithm 3.2, p. 125–126: the same computations are done in a different order and the coefficients form the diagonal of their rectangular array. The version given as a matrix above stores in columns what Burden & Faires’ versions, which adapt the classical hand-calculation algorithm, would store in rows—in a time of slow computers this actually made a difference in the speed at which values of the interpolating polynomial could be calculated, because Fortran “stores arrays by columns”. Hand calculations of divided-difference tables were arranged by putting the \( f(x_j) \)’s in the first column and then intercalating new rows for the divided differences, as in Atkinson’s Tables 3.1 and 3.2, pp. 140–141; the coefficients for the Newton-basis expansion of the interpolating polynomial then appear as the entries in the 0-th column when the work is arranged as we did it above.

3. Error of Approximation with Interpolating Polynomials.

A proof for \((n + 1)\)-point interpolation and the general form of the error term is given in Atkinson’s Theorem 3.2, pp. 134–136. Because the proof swims in indices, I thought it would be appropriate to give a slight variation of his proof for the case \( n = 3 \) and look at it carefully enough that we might see what makes the proof go.

So let us assume that \( f(x) \) is a real-valued function defined on some real interval \( I = (a, b) \) in which it is four-times-differentiable. Let \( \{x_0, x_1, x_2, x_3\} \) be four distinct points in the interval, and form the (unique) cubic polynomial \( P_3(x) \) for which \( P_3(x_i) = f(x_i) \), \( i = 0, 1, 2, 3 \). To get a handle on the difference between \( f(x) \) and \( P_3(x) \) at other points of the interval, let \( x \in (a, b) \) be given and fixed—from now on until further notice is given, we shall use “\( t \)” as the variable argument in functions. Assume first that \( x \) is not one of the interpolation points \( x_0, x_1, x_2, x_3 \), since there is no approximation error at those points anyway. Then \( (x - x_0)(x - x_1)(x - x_2)(x - x_3) \neq 0 \) so we can legitimately divide by it; consequently, there is a unique number \( K_x \) for which

\[ f(x) - P_3(x) = K_x \cdot (x - x_0)(x - x_1)(x - x_2)(x - x_3). \]

(Remember, \( x \) is known, constant, and fixed!) Consider the function defined for \( t \in (a, b) \) by

\[ G(t) = [f(t) - P_3(t)] - K_x \cdot (t - x_0)(t - x_1)(t - x_2)(t - x_3) \]

or equivalently

\[ G(t) = [f(t) - P_3(t)] - K_x \cdot \Psi(t) \]

where \( \Psi(t) = (t - x_0)(t - x_1)(t - x_2)(t - x_3) \) is the error factor for the nodes \( \{x_0, \ldots, x_n\} \) introduced above. It is obvious that \( G(t) = 0 \) for \( t = x_0, x_1, x_2, x_3 \), because each term in the difference defining it is zero for those values of \( t \); also \( G(t) = 0 \) for \( t = x \), by our choice of \( K_x \). We have thus exhibited 5 distinct values of \( t \) for which \( G(t) = 0 \). By Rolle’s theorem—according to which there is at least one zero of \( G'(t) \) between any two zeros of \( G(t) \)—the derivative \( G'(t) \) has at least 4 distinct zeros in \((a, b)\). Repeating this argument, we see that \( G''(t) \) has at least 3 distinct zeros, \( G^{(3)}(t) \) has at least 2 distinct zeros, and so finally there is at least one \( \xi_x \in (a, b) \) for which \( G^{(4)}(\xi_x) = 0 \). Now

\[ G^{(4)}(t) = \frac{d^4}{dt^4} [f(t) - P_3(t)] - K_x \cdot \frac{d^4}{dt^4} [(t - x_0)(t - x_1)(t - x_2)(t - x_3)]. \]
The fourth derivative of $P_3(t)$ is identically zero, because the degree of $P_3(t)$ is 3. Similarly, since the function
\[ \Psi(t) = (t-x_0)(t-x_1)(t-x_2)(t-x_3) = t^4 + \text{ (lower degree terms)), its fourth derivative is simply } \frac{d^4}{dt^4} \Psi(t) = 4! \].
Thus for $t = \xi_x$ we get
\[
0 = G^{(4)}(t) = \left[ \frac{d^4}{dt^4} [f(t) - P_3(t)] - K_x \cdot \frac{d^4}{dt^4} [(t - x_0)(t - x_1)(t - x_2)(t - x_3)] \right]_{t=\xi_x}
= f^{(4)}(\xi_x) - K_x \cdot 4! .
\]
This is an equation in which we can think of $K_x$ as unknown and the other quantities as known; if we solve it for $K_x$, we get
\[
K_x = \frac{f^{(4)}(\xi_x)}{4!} .
\]
So, finally,
\[
f(x) - P_3(x) = K_x \cdot (x - x_0)(x - x_1)(x - x_2)(x - x_3)
\]
\[
f(x) = P_3(x) + \frac{f^{(4)}(\xi_x)}{4!} (x - x_0)(x - x_1)(x - x_2)(x - x_3)
\]
\[
= P_3(x) + \frac{f^{(4)}(\xi_x)}{4!} \Psi(x)
\]
as advertised. While we excluded the possibility that $x$ was one of the $x_i$ in the derivation, the last-written equation holds without that restriction, since the “remainder term” takes the value zero and $P_3(\cdot)$ was determined by requiring that $P_3(x_i) = f(x_i)$ for $i = 0, 1, 2, 3$. Note that (as usual in remainder terms) the choice of $\xi_x$ depends on $x$, there may be more than one possible choice of $\xi_x$ (because the auxiliary function $G(t)$ may have had more than one local maximum or minimum—so it is not well-determined and should be thought of as $\xi_x$, not as a function of $x$) and generally all that can be said about the location of $\xi_x$ is that
\[
\min \{ x_0, x_1, x_2, x_3, x \} < \xi_x < \max \{ x_0, x_1, x_2, x_3, x \} .
\]
By the way: when $\min \{ x_0, x_1, x_2, x_3 \} \leq x \leq \max \{ x_0, x_1, x_2, x_3 \}$, evaluating $P_3(x)$ as a substitute for evaluating $f(x)$ is usually called interpolation of $f(x)$, while if $x$ is “beyond all the $x_i$” the corresponding evaluation is called extrapolation. The form of the error term shows us that the latter should be done with extreme care, because the absolute value of the product $(x - x_0)(x - x_1)(x - x_2)(x - x_3)$ grows very rapidly (like the fourth power of the distance from the $x_i$’s) when extrapolation is attempted. The same caveat holds in the general situation of $n + 1$ points and an interpolating polynomial of degree $\leq n$, as the general form of the error term shows (see Atkinson’s Theorem 3.2, bottom of p. 135).

On the other hand, the general form of the error term tells us something interesting about the divided differences of a sufficiently-differentiable function $f(x)$ defined on an interval $(a, b)$. Let \{ $x_0, x_1, \ldots, x_k$ \} $\subseteq (a, b)$ be the usual $k \geq 1$ points, and form the usual
\[
P_{k-1}(x) = \sum_{j=0}^{k-1} f[x_0, \ldots, x_j](x - x_0) \cdots (x - x_{j-1}) .
\]
Now let $x_k$ be any new given point in $(a, b)$. Then to get an interpolating polynomial of degree $k + 1$ that also takes the correct value at $x_k$, one would simply add an additional term to $P_{k-1}(\cdot)$, obtaining
\[
P_k(x) = P_{k-1}(x) + f[x_0, \ldots, x_{k-1}, x_k](x - x_0) \cdots (x - x_{k-1}) .
\]
If we plug $x_k$ in for $x$ in $P_k(x)$, we will get the value $f(x_k)$. But now, using the general form of the error term, we have two ways to write the value of $f(x_k)$: taking “$n$” to be $k - 1$ in the general form of the error term, we have both
\[
f(x_k) = P_{k-1}(x_k) + \frac{f^{(k)}(\xi_x)}{k!} (x_k - x_0) \cdots (x_k - x_{k-1})
\]
\[
f(x_k) = P_k(x_k) + f[x_0, \ldots, x_{k-1}, x_k](x_k - x_0) \cdots (x_k - x_{k-1}) .
\]
Comparing the two lines, we get

\[ f[x_0, \ldots, x_k] = \frac{f^{(k)}(\xi)}{k!} \]

in words, the value of every \( k \)-th order divided difference of a sufficiently-differentiable \( f(x) \) is one of the values taken by \( f^{(k)}(\xi)/k! \) at some point \( \min \{x_0, \ldots, x_k\} < \xi < \max \{x_0, \ldots, x_k\} \). This gives us a way to put a bound on the size of the divided differences, if we can put bounds on the derivatives of \( f(x) \), and thereby to give estimates of the error made by approximating a value of \( f(x) \) by the value \( P_n(x) \) of an interpolating polynomial of \( f(x) \).

Just to give some famous examples of the use of the error term (two-point case): Given points \( x_0 < x_1 \) in an interval in which \( f(x) \) is defined and twice-differentiable, we have

\[ f(x) = [f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)] + \frac{f''(\xi)}{2!}(x - x_0)(x - x_1) \]

because the function inside the square brackets is the interpolating polynomial \( P_1(x) \); the last term is the error term. Suppose we know that \( f''(\xi) \) is bounded for \( \xi \in (a, b) \), say \( |f''(\xi)| \leq M_2 \); then the absolute value of the error term cannot exceed \( M_2 \cdot |(x - x_0)(x - x_1)| \). Since \( y = (x - x_0)(x - x_1) \) is a parabola, we see that for \( x_0 \leq x \leq x_1 \) the absolute value \( |(x - x_0)(x - x_1)| \) is maximized at the center of the interval, with maximum value \( [(x_1 - x_0)/2]^2 \). The error in using the linear interpolant as an approximation to the value of \( f(x) \) in \([x_0, x_1]\) then cannot exceed \( (M_2/8) \cdot (x_1 - x_0)^2 \) in absolute value. By contrast, the Taylor-series linear approximation (tangent line) with base point \( x_0 \), say, would have estimated absolute error \( |f''(\xi)/2!| \cdot (x_1 - x_0)^2 \leq (M_2/2)(x_1 - x_0)^2 \), four times as large. (Of course the comparison is unfair: the tangent-line approximation with point of tangency equal to the midpoint of the interval \([x_0, x_1]\) would have the same error-term estimate as that of the linear interpolant. But constructing it would require knowledge and evaluation of \( f'(x) \) at the midpoint, while \( P_1(x) \) is constructed from function values.) If \( f''(\xi) \geq 0 \) throughout the interval, then since \( (x - x_0)(x - x_1) \leq 0 \) for \( x_0 \leq x \leq x_1 \) the linear interpolant = secant line \( y = P_1(x) \) satisfies

\[ f(x) = P_1(x) + \frac{f''(\xi)}{2!}(x - x_0)(x - x_1) \leq P_1(x) \]

throughout the interval: the graph of \( f(x) \) lies below the secant line. Thus \( f(x) \) is convex (above) in the sense that any (secant) line that cuts through the graph of \( f(x) \) at two points lying over \((a, b)\), say \((x_0, f(x_0))\) and \((x_1, f(x_1))\), lies above the graph of \( f(x) \) over the interval \([x_0, x_1]\).

A penultimate thing about error (while this is essentially “true by definition”, it is occasionally useful to know. We actually proved it back on p. 12 but the points had different names.) Let \( \{x_0, \ldots, x_n\} \) and an additional point \( x \) be given in an interval \((a, b)\) on which a function \( f(\cdot) \) is defined. Then we have (using the dummy variable \( t \) because \( x \) is being used for a certain number)

\[ f(t) = P_n(t) + \text{[error]}_n(t) \]

as usual. By adding one additional term to \( P_n(t) \), we can get the unique polynomial in \( t \) of degree \( \leq n + 1 \) that also interpolates \( f(\cdot) \) at the value \( t = x \):

\[ f(t) = P_n(t) + f[x_0, \ldots, x_n, x](t - x_0) \cdots (t - x_n) \]

Plug in \( t = x \) and you get both

\[ f(x) = P_n(x) + \text{[error]}_n(x) \]
\[ f(x) = P_n(x) + f[x_0, \ldots, x_n, x](x - x_0) \cdots (x - x_n) \]

Comparing these two lines, you get

\[ \text{[error]}_n(x) = f[x_0, \ldots, x_n, x](x - x_0) \cdots (x - x_n) \]
the error made in approximating \( f(x) \) by \( P_n(x) \) is just the divided-difference-times-the-differences \( f[x_0, \ldots, x_n, x](x - x_0) \cdots (x - x_n) \). Why is this true by definition? Look at the recursion back at the bottom of p. 7 (and the top of p. 8) above: this just says that “\( a_{n+1} \) is what you get by solving \( f(x_{n+1}) - P_n(x_{n+1}) = a_{n+1}(x_{n+1} - x_0) \cdots (x_{n+1} - x_n) \)” for the unknown coefficient \( a_{n+1} \); in this case, we’re using our given “\( x \)” as “\( x_{n+1} \)”. But this is sometimes a handy expression to use for the error of approximating \( f(x) \) by \( P_n(x) \). And of course, we now see that the “unknown coefficient \( K_x \)” we dealt with when we derived the 4th-derivative error term above was actually \( f[x_0, x_1, x_2, x_3, x] \).

Incidentally, this is another way to see that

\[
f[x_0, \ldots, x_k] = \frac{f^{(k)}(x_k)}{k!};
\]

in words, the value of every \( k \)-th order divided difference of a sufficiently-differentiable \( f(x) \) is one of the values taken by \( f^{(k)}(\xi)/k! \) at some point \( \min \{x_0, \ldots, x_k\} < \xi < \max \{x_0, \ldots, x_k\} \), and thus to put a bound on the size of the divided differences if we can put bounds on the derivatives of \( f(x) \).

The last thing we should do before we leave the land of error is to observe that when the function \( f(x) \) that is to be interpolated is analytic in a neighborhood \( U \subseteq \mathbb{C} \) of an interval \( I \subseteq \mathbb{R} \) containing the nodes \( \{x_0, \ldots, x_n\} \), it is possible to use complex function theory to estimate error. Let \( \Gamma \) be a curve in \( U \) that winds once around \( I \); then the Cauchy integral formula is valid for \( f(z) \), so

\[
P_n(z) = \sum_{k=0}^{n} f(x_k) \cdot \frac{\Psi(z)}{\Psi'(x_k)(z - x_k)} = \sum_{k=0}^{n} \left[ \frac{1}{2\pi i} \int_{\Gamma} \frac{f(\zeta) d\zeta}{\zeta - x_k} \right] \cdot \frac{\Psi(z)}{\Psi'(x_k)(z - x_k)}
\]

and thus

\[
f(z) - P_n(z) = \frac{1}{2\pi i} \int_{\Gamma} \left[ 1 - \sum_{k=0}^{n} \frac{\Psi(z)}{\Psi'(x_k)(z - x_k)} \frac{\Psi(z)(\zeta - z)}{(\zeta - x_k)(\zeta - x_k)} \right] \frac{1}{\zeta - z} f(\zeta) d\zeta.
\]

Now one of the consequences of the general theory of residues and integrals is that the function of \( \zeta \) inside the large braces in that contour integral can be replaced by any other function of \( \zeta \) that is analytic in \( U \) except for poles at the same points having the same residues, and the value of the integral will not change. The poles of that function occur at \( \zeta = z \) (where we may temporarily agree that \( z \) is not one of the \( x_k \)'s) and at \( \zeta = x_k \), \( k = 0, \ldots, n \), and it is obvious that they are simple poles. The residue at \( \zeta = z \) is clearly 1. For a fixed index \( j \), the residue at \( \zeta = x_j \) is given by

\[
\lim_{\zeta \to x_j} \left\{ \left( \frac{\Psi(z)}{\Psi'(x_j)(z - x_j)} \right) \frac{1}{\zeta - z} \right\} = -\frac{\Psi(z)(z_j - z)}{\Psi'(x_j)(z - x_j)} \frac{1}{z_j - z} = \frac{-\Psi(z)}{\Psi'(x_j)(z - x_j)}.
\]

But those residues are easy to recognize: they are the same as the residues of \( \Psi'(\zeta) \frac{1}{\zeta - z} \) at those poles. The equality of the residues at \( \zeta = z \) is obvious, and for the others, we have (in view of the fact that \( \Psi(x_j) = 0 \))

\[
\lim_{\zeta \to x_j} \left[ \frac{\Psi(z)}{\Psi'(x_j)} \frac{1}{\zeta - z} \right] = \frac{\Psi(z)}{\Psi'(x_j)} \frac{1}{x_j - z},
\]

the same thing written slightly differently. Hence we have the contour-integral expression (clearly also valid if \( z \) is an \( x_k \))

\[
f(z) - P_n(z) = \frac{1}{2\pi i} \int_{\Gamma} \left[ \frac{\Psi(z)}{\Psi'(\zeta)} \frac{1}{\zeta - z} \right] f(\zeta) d\zeta
\]

for the error of the interpolating polynomial at points \( z \in I \). The integrand in this contour integral lends itself easily to estimation, and this error formula has been the focus in approximation theory of considerable investigation, some of which still continues.
4. Evaluating Newton-Form Polynomials.

This is going to be very short: the point is just that an appropriate modification of the nested-multiplication algorithm on Atkinson’s pp. 96–97 will evaluate a polynomial written in terms of a Newton basis

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

as efficiently as Horner's method evaluates a polynomial written in terms of the standard basis of monomials. Written in a sort of line-number-less BASIC, here’s the algorithm, which inputs the degree \( N \) as efficiently as Horner’s method evaluates a polynomial written in terms of the standard basis of monomials.

```
Y = A(N)
FOR K = N-1 TO 0 STEP -1
Y = Y*(X - XSUB(K)) + A(K)
NEXT K
```

In applications, the array \( A(0:N) \) would probably have been produced by computation of a divided-difference table in an earlier routine of the program.

5. Difference Calculus with Equally Spaced Nodes and some Rudimentary Considerations about Error.

It can be argued that the real error here was that I ever brought up the subject of error. In any event, §§5.2 ff. below offer an introduction to some of the ways that errors inevitably enter machine computation, and why these errors place restrictions that are even more limiting than those of theoretical nonconvergence (e.g., Runge’s example) on the usefulness of polynomial interpolation as a number-crunching tool.[7]

5.1. Equally Spaced Nodes, Tabular Differencing, and Linear Algebra. This material is complementary to Atkinson’s §3.3 (p. 147 ff.) and the two sources should be read together.

The computations required to do interpolation with polynomials in Newton form undergo some significant simplifications when the increments between nodes are all equal to some constant \( h > 0 \), so that

\[ x_1 = x_0 + h, \ldots, x_n = x_0 + n \cdot h \]

these simplifications even help keep the computations accurate, as we shall see later. First of all, there is “no need to do the dividing in a divided-difference table”: if we set

\[ f_1 = f(x_1) = f(x_0 + h), \ldots, f_n = f(x_n) = f(x_0 + n \cdot h) \]

then since \( f[x_i, x_{i+1}] = (f_{i+1} - f_i)/h \) with a denominator independent of \( i \), it sufficiently to compute the numerators of the first divided differences when one is making a table; the divisions by \( h \) can be carried out later (or, as we shall see, need not be carried out at all). More generally (cf. Atkinson’s p. 148) the numerators \( \Delta^k f_i \) of the divided differences \( f[x_i, \ldots, x_{i+k}] \) are the only things that need to be computed, since the denominators (after reduction to a single fraction) will all have the form \( kh^k \) and that division can be carried out later or not at all. And if we then introduce a new coordinate \( \mu \) such that “\( \mu \) has origin \( x_0 \) and unit-of-length \( h \)”—that is, \( \mu = (x - x_0)/h \), or \( x = x_0 + \mu \cdot h \)—we find that each term of the Newton form looks much nicer with the “tabular differences” \( \Delta^k f_0 \) and the new coordinate:

\[
\begin{align*}
    f[x_0, \ldots, x_k] \cdot (x - x_0)(x - x_1) \cdots (x - x_{k-1}) &= \left[ \frac{\Delta^k f_0}{kh^k} \right] \cdot \mu(\mu - 1) \cdots (\mu - (k - 1)) \cdot h^k \\
    &= (\Delta^k f_0) \frac{\mu(\mu - 1) \cdots (\mu - k + 1)}{k!} = (\Delta^k f_0) \binom{\mu}{k}
\end{align*}
\]

[7] This portion of these notes is a rewritten version of a previously existing set. (The references have been updated.) There is consequently some overlap with material of Atkinson’s Ch. 1. Readers pressed for time may pass over the material dealing with error, but they might still want to give it a superficial glance.
so the divisions by \( h^k \) do not appear, and

\[
P_n(x) = \sum_{k=0}^{n} (\Delta^k f_0) \binom{\mu}{k} = \sum_{k=0}^{n} (\Delta^k f_0) \cdot \frac{\mu(\mu-1)\cdots(\mu-k+1)}{k!}
\]

is what the Newton-form interpolating polynomial looks like when it is expressed in terms of the new “normalized” coordinate \( \mu \).

The computation of the \( \Delta^k f_0 \)'s is obviously linear computation with vectors. If the difference table is written in “computer form,” so that the first differences are written on the same lines as the function values instead of between them, and if the columns of the difference table are thought of as vectors—of somewhat undetermined dimension, to be sure—then we see that passing from the (leftmost) column of function values to the (next-right) column of first tabular differences can be thought of as follows: first apply the “[backward] shift operator” \( S \) to the column of function values, then subtract the original column from the result. In linear-operator notation, if \([f]\) denotes the column of function values, then

\[
[\Delta f] = (S - I)[f],
\]

where \( I \) is the identity operator:

\[
\begin{array}{cccccc}
 f_0 & f_1 & f_1 - f_0 \\
 f_1 & f_2 & f_2 - f_1 \\
 \vdots & \vdots & \vdots \\
 f_i & f_{i+1} & f_{i+1} - f_i \\
 f_{i+1} & f_{i+2} & f_{i+2} - f_{i+1} \\
 \vdots & \vdots & \vdots \\
\end{array}
\]

\[
[f] \quad S[f] \quad (S - I)[f] = [\Delta f]
\]

{This may help to explain the use of \( \Delta f_0 \) to denote \( f_1 - f_0 \), etc.: it is the zero-th coordinate of the vector \( S[f] - [f] \). These vectors can be thought of as having negative-integer coordinates too, written “above” the zero-coordinates; then the backward differences of Atkinson p. 151 can be formalized the same way, using the forward-shift operator \( S^* \) instead of the backward-shift \( S \) introduced above.} While we are writing these vectors as if they stretched out infinitely far above (negative indices) and below (positive indices), in fact we only compute with finitely many indices, so \( S, I \) and \( \Delta \) can be thought of as big matrices: \( S \) looks like

\[
\begin{bmatrix}
 0 & 1 & 0 & 0 & 0 & \cdots \\
 0 & 0 & 1 & 0 & 0 & \cdots \\
 0 & 0 & 0 & 1 & 0 & \cdots \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \\
\end{bmatrix}
\]

The operation of computing the \( \Delta^2 f_i \)'s from the \( f_i \)'s is obviously the same operation, so in the sense of the \( k \)-th power of a matrix (i.e., a linear operator) being the result of applying the matrix \( k \) times, we have

\[
[\Delta^k f] = (S - I)^k[f];
\]

“repeat the operation that makes the vector of first differences \( k \) times, and you get the vector of \( k \)-th differences.” {The reader who doesn’t see this abstractly should compute a few columns in the difference table.}

Now if \( \Delta^k = (S - I)^k \), then—since the powers of the matrix \( S \) commute with each other—the binomial theorem applied to \( (S - I)^k \) says

\[
\Delta^k[f] = \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} S^j[f] \tag{*}
\]
and $S^j[f]$—the result of applying the shift operator $S$ a total of $j$ times to the vector $[f]$—is obviously $[f_j, f_{j+1}, \ldots]^T$. \{Note that the $i$-th coordinate of $S^j[f]$ is $f_{i+j}$, so although $S$ shifts vectors backward it appears to shift indices forward. This may be confusing until you think about it a bit.\} It follows that the $i$-th coordinate of the vector $\Delta^k[f]$ is given by

$$[\Delta^k f]_i = \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} f_{i+j}$$

so we have a closed-form formula for the $k$-th tabular difference (with increment $h$) of $f(x)$ at the point $x_i$.

If you think about it for a moment, however, you will see that the interesting row of the difference table is the zeroth row, because that’s where the coefficients of the Newton polynomial are. The formula

$$[\Delta^k f]_0 = \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} f_j \tag{\$}$$

relates the vector $[f_0, \ldots, f_n]^T$ of function values at $x_0, \ldots, x_n$ and the vector of coefficients in the Newton forward-difference form of the interpolating polynomial (Atkinson’s formula (3.3.7)). The vector of coefficients depends linearly on the vector of values, and the matrix that produces the former from the latter is the $(n+1) \times (n+1)$ matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots \\ -1 & 0 & 0 & 0 & \cdots \\ 1 & -2 & 1 & 0 & \cdots \\ -1 & 3 & -3 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

or, in closed form, $A = \left[ (-1)^{i+j} \binom{i}{j} \right]$—which is lower-triangular since $\binom{i}{j} = 0$ for $j > i$. $A$ is invertible, and in fact computing its inverse is trivial unless you try to do it with the matrices: the formula

$$P_n(x) = \sum_{k=0}^{n} [\Delta^k f]_0 \binom{n}{k}$$

with $x$ set equal to $x_k$—and therefore $\mu = k$—gives

$$f_k = P_n(x_k) = \sum_{j=0}^{n} \binom{k}{j} [\Delta^j f]_0$$

so $A^{-1}$ is the matrix

$$A^{-1} = \begin{bmatrix} 1 & 2 & 1 & 0 & \cdots \\ 1 & 3 & 3 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

for some moderate-sized $n$. Note that this gives a proof of the not-entirely-trivial combinatorial identity

$$\sum_{k=i}^{j} \binom{j}{k} \binom{k}{i} (-1)^{j-k} = 0, \quad i < j.$$
It also says that if \( D = \text{diag}[1, -1, 1, -1, \ldots] \), then \( A^{-1} = DAD = DAD^{-1} = DAD^* \), so \( A \) and \( A^{-1} \) are even unitarily equivalent. But both \( A \) and \( A^{-1} \) are now seen to have large coefficients, and that is the reason that—for theoretical reasons which will have nothing to do with how bad machine operations are, but just with how bad floating-point numbers are, the practical upper limit on the degree of computable interpolating polynomials is surprisingly small.

\[ \{ I \text{ cannot resist the temptation to give a proof of the Newton interpolation formula—} \text{for equally spaced nodes only} \} \text{—by linear algebra. The relation } \Delta = S - I \text{ is equivalent to } S = I + \Delta, \text{ and therefore implies } S^k = (I + \Delta)^k \text{ for any } k \in \mathbb{N}. \text{ For simplicity, consider the case for which } x_0 = 0 \text{ and } h = 1, \text{ since this involves only a (possible) translation and/or a change of scale. Fix any } 0 \leq n \in \mathbb{N}; \text{ for } 0 \leq k \leq n \text{ we have} \]

\[
f_k = [S^k f]_0 = [(I + \Delta)^k]_0 = \sum_{j=0}^{k} \binom{k}{j} [\Delta^j f]_0
\]

\[
= f_0 + [\Delta f]_0 \frac{k}{1!} + [\Delta^2 f]_0 \frac{k(k-1)}{2!} + \cdots + [\Delta^k f]_0 \frac{k(k-1)\cdots2\cdot1}{k!}.
\]

Consider the polynomial obtained by taking the r. h. side of the \( n \)-th case of what we just obtained and replacing the “\( k \)” by “\( \mu \)” (or “\( x \)”):

\[
P(\mu) = f(0) + [\Delta f]_0 \frac{\mu}{1!} + [\Delta^2 f]_0 \frac{\mu(\mu-1)}{2!} + \cdots + [\Delta^n f]_0 \frac{\mu(\mu-1)\cdots2\cdot1}{n!}.
\]

This is a polynomial in \( \mu \) of degree at most \( n \). If one sets \( \mu = k \) for a natural number \( k < n \), all the terms of the evaluated polynomial \( P(k) \) after the \( k \)-th contain a factor of the form \( (\mu - k) \) and consequently drop out. The surviving terms thus give us the \( k \)-th case

\[ P(k) = \sum_{j=0}^{k} \binom{k}{j} [\Delta^j f]_0 = [S^k f]_0 = f_k = f(k) \]

so \( P(\mu) \) interpolates \( f(\mu) \) at \( \mu = 0, 1, \ldots, n \), and as we know, the polynomial \( P(x) \in \mathcal{P}_n \) that does that interpolation is unique.\}

5.2. Floating-Point Numbers; Unit Roundoff\(^{(8)}\) As is obvious to anyone who thinks about it even superficially, computers and calculators do not have the whole real number field at their disposal. They make do with a subset of the real numbers defined in essentially the following way. Let \( \beta > 1 \) be a natural number called the base—in most cases = 2, but sometimes 10 or 16. Any positive real number can be written in the form

\[ \beta^e \times (0.d_1d_2 \ldots d_sd_{s+1} \ldots) \]

where \( e \) is an integer exponent and the \( d_i \) are natural numbers with \( 0 \leq d_i < \beta \) for all \( i \), and also \( d_1 \geq 1 \); the thing in parentheses is an abbreviation for the infinite series

\[ d_1\beta^{-1} + d_2\beta^{-2} + \cdots + d_s\beta^{-s} + \cdots. \]

If \( \beta = 10 \) this is almost the same as “scientific notation” except that it has the form

\[ 10^e \times 0.d_1d_2 \ldots \text{ instead of } d_1.d_2d_3 \ldots \times 10^{e-1}; \]

the situation for other bases is analogous. The floating-point numbers used internally by the computer or calculator are those that have the form

\[ \pm \beta^e \times (0.d_1 \ldots d_s) \]

\(^{(8)}\) Much of the early part of this material will be redundant in the presence of Atkinson’s Ch. 1.
where $e$ is also restricted to a certain interval of (signed) integers. Clearly the set of numbers that can be written in this form can be subjected to rational operations ($+, - , \times, \div$) as long as they don’t get too big or too small. It is conventional to regard real numbers $y$ as being mapped to “floating-point approximations $\text{fl}(y)$" inside the machine by some function $\text{fl}(\cdot)$ which rounds off $y$ at $s$ significant digits; that is,

$$\text{fl}(y) - y = \frac{\beta^s \times \epsilon}{\beta^s \times (0.d_1d_2 \ldots)}$$

if $y = \beta^s \times (0.d_1d_2 \ldots d_{s+1} \ldots)$, then $\text{fl}(y) = y + \beta^s \cdot \epsilon$ is an $s$-digit number

where $\epsilon$ is machine-selected in such a way as to be no larger in absolute value than $1/2$ the largest number which could have been represented using the digits from the $(s + 1)$-st on preceded by $s$ zeros. The number in base 10 that has the form $0.0 \ldots 099999 \ldots$ with $s$ leading zeros is the same number as $0.0 \ldots 1$ with 1 in the $s$-th place, and a similar situation occurs with respect to other bases; so $|\epsilon| \leq \frac{1}{2} \beta^{-s}$. Thus

$$\left| \text{fl}(y) - y \right| \leq \frac{\beta^s \times |\epsilon|}{\beta^s \times (0.1000 \ldots)}$$

and since the smallest value the denominator could have is $\beta^s \times (0.1000 \ldots)$,

$$\left| \text{fl}(y) - y \right| \leq \frac{\beta^s \times |\epsilon|}{\beta^s \times \beta^{-1}} \leq \frac{1}{2} \beta^{-(s-1)}.$$

The number $\delta = \frac{1}{2} \beta^{-(s-1)}$ is called the unit roundoff for $s$-digit numbers in base $\beta$, and we have established the

**Proposition:** The relative error in the $s$-digit floating-point representation of a real number in base $\beta$ is bounded by the unit roundoff: that is,

$$|\text{fl}(y) - y| \leq \delta \cdot |y|.$$  

or

$$\text{fl}(y) = y + \epsilon y = y \cdot (1 + \epsilon),$$

where $|\epsilon| \leq \delta$.

{Remark: Instead of machine-selected nearby numbers, some machines, particularly Big Blue mainframes, simply used to (?) replace $(0.d_1d_2 \ldots d_{s+1} \ldots)$ by $(0.d_1 \ldots d_s)$, an operation called truncation or chopping (rather than rounding). The effect was to replace $\frac{1}{2} \beta^{-(s-1)}$ by $\beta^{-(s-1)}$, as the reader could easily show. That number would be the unit roundoff for such machines.}

For true binary arithmetic, the unit roundoff in single precision is typically $2^{-24}$, in double precision $2^{-53}$. This number is hard- and software specific, and the only sources of reliable information are the manuals and (perhaps more reliable) experimental hexadecimal dumps.

**5.3 Differentiating of Erroneous Data.** Recall the matrix we called $A$ in §5.1 above. It is an early part—and an easy one—of numerical linear algebra that if we define the $\ell_\infty$ norm of a vector by

$$\|x_0, \ldots, x_n\|_\infty = \max_{0 \leq i \leq n} |x_i|$$

and the $\ell_\infty$-norm of a matrix $A = [a_{ij}]$ by

$$\|A\| = \max_{0 \leq i \leq n} \left\{ \sum_{j=0}^{n} |a_{ij}| \right\},$$

then $\|Ax\| \leq \|A\| \cdot \|x\|_\infty$, and the bound is sharp.\(^{(9)}\) A little harder—but not much—is the fact that if we change $x$ by an error vector $\tilde{e}$, then the relative size of the error in the image vector can be as large as $\|A\| \cdot \|A^{-1}\|$ times the size of $\tilde{e}$ relative to $x$. The details are in Atkinson, pp. 529 ff. For the matrix $A$ in

\(^{(9)}\) See pp. 8–11 of the numerical linear algebra notes for this course.
the \( n \)-th degree interpolation problem, it is evident that the largest absolute row sums for \( A \) and \( A^{-1} \) are the same, and are equal to

\[
\sum_{j=0}^{n} \binom{n}{j} = (1 + 1)^n = 2^n
\]

so the relative size of errors fed through either \( A \) or \( A^{-1} \) can be multiplied by a factor as large as \( 2^n \cdot 2^n = 2^{2n} \). But we can see this happening in the differencing process without appealing to general facts about error magnification in linear algebra. Direct calculation—which the reader can easily carry out—shows that \( A \) maps the vector (or sequence of function values) \([1, -1, 1, \ldots, 1]^T\) to \([1, 2, 4, \ldots, 2^n]^T\), while \( A^{-1} \) sends \([1, 1, \ldots, 1]^T\) to the vector \([1, 2, 4, \ldots, 2^n]^T\) or, equivalently, \( A \) sends \([1, 2, 4, \ldots, 2^n]^T\) to \([1, 1, \ldots, 1]^T\). Thus if \( f(x) = 2^x \) is subjected to errors of alternatingly \( \pm 2^{-n} \) at each \( x = 0, \ldots, n \), then the relative error in the vector \([f_0, f_1, \ldots, f_n]^T\) will be \( 1/2^{2n} \), but the relative error in the vector \([f_0, f_0, \ldots, 2^n f_0]^T\) will be 1; the relative error has been increased by a factor of \( 2^{2n} \).

But this is unfair! the error in \( f(x) \) has been compared with the largest value of \( f(x) \), but it’s \( 2^n \)-times as relatively large compared to the smallest value of \( f(x) \) in the interval \([0, n] \). Also \( f(x) \) is growing like mad—exponentially—while \( f(x) \) would be more likely to stay relatively constant, or at least grow at an approximately polynomial rate! These are valid objections, but a fairly constant function would be even worse: if \( f(x) = 1 \) identically but its values at \( 0, 1, \ldots, n \) are in error by \( \pm \epsilon \) at alternate points, then the \( \Delta^k f_0 = 2^k \epsilon \) for \( k \geq 1 \), notwithstanding the fact that they should all be zero. That error is \( 2^k \)-times the error in \( f \) even with the most optimistic bookkeeping. {The curious experimental fact is that if one tries this with 12 points and an error of about \( 10^{-6} \), the error is multiplied by a factor more like \( 2^{14.6} \), this strange effect is only explained when we consider the intrinsic inaccuracies in floating-point arithmetic.}

So with the unfair but standard estimate of error growth, together with the fact that single-precision data will typically be entered in decimal with inevitable roundoff errors of relative magnitude about \( \frac{1}{2} 2^{-10^{-6}} \approx 2^{-17.6} \), it appears that a cautious person would not trust any binary digit in the difference-table entries beyond the 9-th or so in single-precision arithmetic (assuming the right rate of growth is \( 2^{2k} \)). Even if the function values are known to full single-precision floating-point accuracy, 12 differences will wipe out all the binary digits according to the standard worst-case estimate. If the right rate of growth is actually \( 2^k \), do you really believe in 24-th degree polynomials?

### 5.4. Floating-Point Addition and Subtraction: Loss of Significant Digits

\(^{(10)}\) Suppose \( y_0 \) and \( y_1 \) are two numbers known with “small” relative errors \( \epsilon_0 \) and \( \epsilon_1 \): that is, suppose that the numbers that we actually have to work with are

\[
y_0(1 + \epsilon_0) \quad \text{and} \quad y_1(1 + \epsilon_1)
\]

where \( |\epsilon_0| \leq \delta \) and \( |\epsilon_1| \leq \delta \) (you can think of \( \delta \) as being unit roundoff, if you wish). Let \( \Delta y = y_1 - y_0 \). Then the accurately computed value of the difference between the inaccurate values of \( y_1 \) and \( y_0 \), which the unimaginative would write as

\[
y_1(1 + \epsilon_1) - y_0(1 + \epsilon_0),
\]

can be written in either of the forms

\[
(y_0 + \Delta y)(1 + \epsilon_1) - y_0(1 + \epsilon_0) = \Delta y(1 + \epsilon_1) + y_0(\epsilon_1 - \epsilon_0)
\]

\[
y_1(1 + \epsilon_1) - (y_1 - \Delta y)(1 + \epsilon_0) = \Delta y(1 + \epsilon_0) + y_1(\epsilon_1 - \epsilon_0).
\]

If \( y_0 \) and \( y_1 \) are of opposite signs, then there is a number \( t \) with \( 0 \leq t \leq 1 \) for which \( (1 - t)y_0 + ty_1 = 0 \). Taking the sum of \( (1 - t) \)-times the first equation and \( t \)-times the second equation then gives

\[
y_1(1 + \epsilon_1) - y_0(1 + \epsilon_0) = \Delta y(1 + (1 - t)\epsilon_1 + t\epsilon_0) = \Delta y(1 + \epsilon)
\]

where \( \epsilon = (1 - t)\epsilon_0 + t\epsilon_1 \) and \( |\epsilon| \leq \delta \). Thus the relative error in the computed value of \( \Delta y \) is of the same (relative) size as those in \( y_0 \) and \( y_1 \). An equivalent way to say this—as one sees by replacing \( y_0 \) by \( -y_0 \), for

\(^{(10)}\) Yet another way to think about the material of Atkinson, pp. 17–39.
example—is that if \( y_0 \) and \( y_1 \) have the same sign, then the relative error in \( y_0 + y_1 \) is of the same size as the errors in the two addends.

The outcome with subtraction of two numbers of the same sign and approximately the same size is quite different, however (and that’s why we looked at a formal subtraction rather than a formal addition). In this situation one cannot “tune out” the second term in the value of \( y_1 - y_0 \) computed from the erroneous data, and indeed one has to write

\[
y_1(1 + \epsilon_1) - y_0(1 + \epsilon_0) = \Delta y(1 + \epsilon) + E
\]

where \( |\epsilon| \leq \delta \), which is fine, but where the smallest safe estimate of the “big error term” \( E \) is

\[
|E| \leq \min\{|y_0|, |y_1|\} \cdot 2 \delta.
\]

Thus the relative error \( \frac{|E|}{|\Delta y|} \) may be uncontrollable: the smaller the denominator, the worse it becomes. If \( y_i = f(x_i) \), so \( \Delta y \approx f'(x_i) \cdot h \) where \( h = x_1 - x_0 \), then

\[
\frac{|E|}{|\Delta y|} \approx \frac{|f(x_i)|}{|f'(x_i)|} \cdot 2 \frac{\delta}{h}.
\]

If for example \( f(x) = x^a \), then \( \frac{f(x)}{f'(x)} = \frac{x}{a} \), and for \( a = \frac{1}{2} \) the relative error produced by \( E \) is approximately

\[
2 \cdot \frac{x}{a} \cdot \frac{\delta}{h}.
\]

Atkinson’s Table 3.7, p. 150, is the case of this problem for 7-digit decimal numbers, where \( \delta = \frac{1}{2} \cdot 10^{-6} \), \( a = \frac{1}{2} \), and \( h = 10^{-1} \). We have

\[
2 \cdot \frac{x}{a} \cdot \frac{\delta}{h} \approx 2 \cdot \frac{2}{1} \cdot \frac{1}{2} \cdot 10^{-5} = 4 \cdot 10^{-5} \approx 2 \cdot 10^{-4}
\]

so taking first differences makes as much error as rounding off to two decimal places fewer, and although the effect is not quite so severe for differences of order \( \geq 3 \), there are both theoretical and empirical (the table!) reasons for thinking that the relative error will have reached 1 by the 5th difference, so that while a computer will probably produce differences of high order (because of underflow, a subject we have not discussed, they may all be zero after a certain order is reached), the numbers will be entirely meaningless. By the way, the fact that the computer may be working in 25-digit binary doesn’t help at all—the original errors corresponded to the 7-digit decimal system.

5.5. If It’s This Bad, Why Does It Work So Well? The fact that the \( f_i \) are extremely large compared to the \( \Delta f_i \), which makes for large and uncontrollable relative errors in the \( \Delta f_i \), is responsible for the fortunate fact that these relative errors are incapable of affecting the value of the interpolating polynomial very much. The reason is that in order to add two floating-point numbers (let’s assume both are \( > 0 \)) a computer or calculator must justify the smaller to have the same exponent as the larger.

A decimal example in scientific notation will suffice: to add \( 1.245094 \times 10^3 \) and \( 7.275110 \times 10^{-1} \), it is necessary to write the latter as \( 0.0007275110 \times 10^3 \), and then add:

\[
1.245094 \times 10^3 \\
+0.0007275110 \times 10^3 = 1.245822 \times 10^3
\]

but although rounding away the last 4 digits of the smaller number would have induced a relative error in that number of \( 7 \times 10^{-4} \), the sum will never see it. Almost exactly the same phenomenon occurs in the sum

\[
P_n(x) = \sum_{k=0}^{n} \frac{[\Delta^k f]_0}{k!} \left( \frac{\mu}{k} \right),
\]

21
whose large-index terms (in the case of such a function as \( f(x) = \sqrt{x} \)) are so small that even though they are terribly relatively inaccurate, their inaccuracies are masked by the use to which they are being put. Indeed, to make sure that I wasn’t lying more than absolutely necessary when I wrote these notes, I extended Atkinson’s Table 3.7 to all the points from 2.0 to 2.1 with 0.1 spacing. I found that errors in the values of the polynomial conformed to the errors predicted by the usual theoretical error term—at least, they conformed as much as one can see using single-precision decimal display.

But if you’re going to compute the higher-order differences erroneously and get away with it because the summation can’t see the errors, why compute them at all? It is certainly a waste of computational effort to compute the ones that have no accurate significant digits left: if they’re big they will poison your calculation, and if they’re small they might as well have been zeros—in other words, rather than building a 6th degree polynomial beginning at 2.0 to compute \( f(x) \) for \( x \) near 2.5, you might have been better off, both in terms of theoretical error-term error and in terms of machine error, to have built a 4th degree polynomial beginning at 2.3—and it would have been less work to compute because there were two coefficients fewer. Aside from this, there is the intrinsic instability of polynomial interpolation: never mind the difference table, look at what happens purely theoretically (computed with Lagrange polynomials, yet) at \( \mu = \frac{1}{2} \) [J. F. Epperson, *On the Runge example*, Amer. Math. Monthly 94 #4 (April 1987), 329-341]—see (3a), p. 334-335.

This is precisely the point of view that leads to the consideration of piecewise-polynomial approximants, of which the most familiar are piecewise-linear or “broken-line” approximants. Humans somehow want the same analytic expression for the approximant everywhere on the interval where the approximation is being carried out. Machines don’t care: from a machine’s point of view, the procedure

(i) find out what subinterval \( x \) is in, look up the coefficients for the small-degree polynomial relevant to that subinterval, and evaluate that polynomial at \( x \)

may well be faster and less erroneous than

(ii) plug \( x \) into a large-degree polynomial that is the same for the whole interval in which approximation is desired.

Binary searches and table lookups involve addition and logic, the two things at which a computer is fastest and most accurate; high-degree polynomial evaluation involves lots of floating-point multiplication (which is never fast), addition of numbers of disparate sizes, and subtraction of numbers of comparable sizes, all of which is intrinsically inaccurate.

These considerations lead us inevitably to cubic splines. But we need to begin by looking a little bit at Hermite interpolation, since it is what leads to good error estimates for the derivatives of (cubic) spline interpolants.

6. Hermite Interpolation.

Hermite Interpolation is just a name for polynomial interpolation of function values and their derivatives.\(^{(1)}\) It is possible to interpolate derivatives up to any arbitrary (finite) order, and we shall discuss that presently. To start, however, we shall examine only the problem of interpolating first-derivative values.

In the following material the reader will note that we always interpolate both a function value and the value of consecutive derivatives at a node. While it is easy to show that one could interpolate both the value and the derivative at some points and only interpolate the function value at others, it is important to realize that attempts to interpolate only the value of a derivative at a point may fail. (The standard counterexample that demonstrates this is an attempt to interpolate a function value at \( x_0 = -1 \) and a function value at \( x_2 = 1 \), while only attempting to interpolate the derivative at \( x_1 = 0 \). There are three conditions so one expects to need a polynomial of degree 2 with three coefficients. If \( p_2(x) = a_2x^2 + a_1x + a_0 \) is a polynomial of degree 2 that takes the values \( p_2(-1) = 0 \) and \( p_2(1) = 0 \), however, then by “dividing out

\(^{(1)}\) Honorific for the 19th-century French mathematician Charles Hermite, for whom hermitian matrices and Hermite polynomials are also named.
the roots” we see that $p_2(x) = a_2(x + 1)(x - 1) = a_2(x^2 - 1)$. The remaining undetermined coefficient lets us specify $p_2(0) = -a_2$ any way we please, but no matter what $a_2$ is, $p'_2(0) = [2a_2x]_{x=0} = 0$. Thus we do not have the freedom to specify $p'_2(0)$ if we require $p_2(-1) = p_2(1) = 0$. If we go up a degree and use a cubic polynomial $p_3(x)$, then—as we shall see below—we can specify $p'_3(0)$ any way we wish, but we shall also have to specify the value of $p_3(0)$ or we shall lose the uniqueness of the interpolating polynomial.) The problem of interpolating derivatives without function values is called Birkhoff interpolation; it presents many interesting complications. See Atkinson’s problems 29 and 30 on pp. 190–191.

6.1. Hermite Interpolation in Lagrange Form. Recall that when we do Lagrange-form interpolation of function values (only) at distinct nodes $\{x_0, \ldots, x_n\}$, we construct the $(n + 1)$ Lagrange “cardinal polynomials”

$$\ell_i(x) = \prod_{j \neq i} \left( \frac{x - x_j}{x_i - x_j} \right) \in \mathcal{P}_n.$$ 

These have the property that $\ell_i(x_j) = 1$ if $i = j$ and $= 0$ otherwise, and consequently one can simply write the formula

$$p_n(x) = \sum_{i=0}^{n} f(x_i) \ell_i(x)$$

and have a polynomial $p_n(x) \in \mathcal{P}_n$ for which $p_n(x_i) = f(x_i)$ at each of the nodes $x_i$. It would be splendid if we could just find polynomials $\{h_i(x)\}_{i=0}^{n}$ and $\{\tilde{h}_i(x)\}_{i=0}^{n}$ with the properties

$$h_i(x_j) = \delta_{ij}; \quad \tilde{h}_i'(x_j) = 0 \text{ for all } 0 \leq j \leq n;$$

$$\tilde{h}_i(x_j) = 0 \text{ for all } 0 \leq j \leq n; \quad \tilde{h}_i'(x_j) = \delta_{ij} \quad (*)$$

because then one could simply write the formula

$$p(x) = \sum_{i=0}^{n} f(x_i) h_i(x) + \sum_{i=0}^{n} f'(x_i) \tilde{h}_i(x)$$

and it would be obvious, by plugging the various $x_j$’s and watching the 0’s and 1’s, that $p(x_j) = f(x_j)$ and $p'(x_j) = f'(x_j)$ held for each node $x_j$, $j = 0, \ldots, n$.

It is easy to construct these new polynomials if one starts with the $\ell_i(x)$’s belonging to these nodes—we already know the properties of the $\ell_i(x)$’s that we need—and constructs the $h_i(x)$’s from them first. Consider one of the polynomials $\ell^2_i(x)$, the square of a Lagrange cardinal polynomial. It equals zero for every $x = x_j$ except $x = x_i$, for which its value is 1. Its derivative is $2 \ell_i(x) \ell'_i(x)$ which equals zero for every $x = x_j$ except $x = x_i$. If we multiply it by the factor $(x - x_i)$ we get a polynomial

$$\tilde{h}_i(x) = (x - x_i) \ell^2_i(x)$$

with derivative $\tilde{h}_i'(x) = 1 \cdot \ell^2_i(x) + (x - x_i) \cdot 2 \ell_i(x) \ell'_i(x)$. The value of $\tilde{h}_i(x)$ at $x = x_j$ is now zero for every $x_j$ including $x_i$. The derivative $\tilde{h}_i'(x)$ will clearly be zero for every $x = x_j$ except $x = x_i$, and when we plug that in we see that $\tilde{h}_i(x_i) = 1 + 0 = 1$, exactly what we wanted in (*) above.

To get the $h_i(x)$’s, we now “correct” the polynomials $\ell^2_i(x)$. As we just saw, $\ell^2_i(x)$ equals zero for every $x = x_j$ except $x = x_i$, for which its value is 1. Its derivative is $2 \ell_i(x) \ell'_i(x)$ which equals zero for every $x = x_j$ except $x = x_i$, where its value is unfortunately $2 \ell'_i(x_i)$ about which we know nothing (although it is easy to show that it is not zero). However, we can use the $h_i(x)$ we just found to subtract off this value of the derivative without disturbing either the value or the derivative at any other point. Thus

$$h_i(x) = \ell^2_i(x) - 2 \ell'_i(x_i) \tilde{h}_i(x) = \ell^2_i(x) - 2 \ell'_i(x_i)(x - x_i) \ell^2_i(x) \equiv [1 - 2 \ell'_i(x_i)(x - x_i)] \cdot \ell^2_i(x)$$

23
has exactly the properties we put in our wish list (*) above. So the Hermite interpolation problem is solved! The interpolating polynomial will belong to \( P_{2n+1} \) because the \( h_i(x) \) and \( \tilde{h}_i(x) \) do, and the polynomial that we wanted to write, namely \( \sum_{i=0}^{n} f(x_i) h_i(x) + \sum_{i=0}^{n} f'(x_i) \tilde{h}_i(x) \), is a linear combination of them; this is reasonable, because to prescribe 2 (linear) conditions at each of \( n+1 \) points is to prescribe \( 2 \times (n+1) = 2n + 2 \) conditions altogether, and a polynomial of degree \( 2n + 1 \) has \( 2n + 2 \) coefficients.

6.2. Uniqueness of Solution.\(^{(12)}\) Now that we know that it is possible to interpolate \textit{both} the values \textit{and} the derivative values at a given set of distinct nodes \( \{x_0, \ldots, x_n\} \) with a polynomial \( p_{2n+1}(x) \in P_{2n+1} \), it would be good to know that the solution of this interpolation problem is unique. Suppose that \( p_{2n+1}(x) \) and \( s_{2n+1}(x) \) are two polynomials of degree \( \leq 2n+1 \) such that for a certain set \( \{x_0, \ldots, x_n\} \) of \( n+1 \) distinct nodes one has \( p_{2n+1}(x_i) = s_{2n+1}(x_i) \) for all \( i, i = 0, \ldots, n \), and also \( p'_{2n+1}(x_i) = s'_{2n+1}(x_i) \) for all \( i, i = 0, \ldots, n \). Then if we look at the difference of those two competing polynomials—call it \( q_{2n+1}(x) = p_{2n+1}(x) - s_{2n+1}(x) \)—we see a polynomial of degree at most \( 2n+1 \) for which \( q_n(x) = 0 \) for \( x = x_i, i = 0, \ldots, n \) and also \( q'_n(x) = 0 \) for \( x = x_i, i = 0, \ldots, n \). The first of those conditions says that \( q_{2n+1}(x) \) has roots at each of \( x = x_0, x_1, \ldots, x_n \), and therefore one can factor the product \( (x-x_0) \cdots (x-x_n) \) out of it:

\[
q_{2n+1}(x) = (x-x_0) \cdots (x-x_n) \cdot g_n(x)
\]

where \( g_n(x) \) is a polynomial. The degree of \( g_n(x) \) can be at most \( n \), because the product \( (x-x_0) \cdots (x-x_n) \) already has degree \( n + 1 \) while \( q_{2n+1}(x) \) has degree at most \( 2n + 1 \). Differentiation gives us

\[
q'_{2n+1}(x) = (x-x_0) \cdots (x-x_n) \cdot g'_n(x) + \frac{d}{dx} \left\{ (x-x_0) \cdots (x-x_n) \right\} \cdot g_n(x)
\]

\[
0 = q'_{2n+1}(x_i) = (x_i-x_0) \cdots (x_i-x_i) \cdots (x_i-x_n) \cdot g'_n(x_i)
\]

\[
+ \frac{d}{dx} \left\{ (x-x_0) \cdots (x-x_n) \right\} \bigg|_{x=x_i} \cdot g_n(x_i)
\]

\[
0 = 0 + \prod_{j \neq i} (x_i-x_j) \cdot g_n(x_i).
\]

Since the product inside the square brackets cannot be zero, we see that \( g_n(x_i) = 0 \) for \( i = 0, \ldots, n \). However, this means that the \( n \)-th-degree polynomial equation \( g_n(x) = 0 \) has \( n + 1 \) distinct roots, which is impossible unless \( g_n(x) \equiv 0 \)—which then implies that \( q_{2n+1}(x) \equiv 0 \), which says in turn that \( p_{2n+1}(x) \equiv q_{2n+1}(x) \). The two competitors are thus equal, and we have

**Theorem:** Given \( n + 1 \) distinct nodes \( \{x_0, \ldots, x_n\} \), \( n + 1 \) “values” \( \{f(x_i)\}_{i=0}^{n} \) and \( n + 1 \) “derivative values” \( \{f'(x_i)\}_{i=0}^{n} \), there is one and only one polynomial \( p_{2n+1}(x) \) of degree \( \leq 2n + 1 \) for which the \( 2n + 2 \) equalities \( p_{2n+1}(x_i) = f(x_i), i = 0, \ldots, n \) and \( p'_{2n+1}(x_i) = f'(x_i), i = 0, \ldots, n \), all hold.

6.3. Hermite Interpolation in Newton Form. As we have seen,\(^{(13)}\) the use of the Lagrange basis for finding polynomial interpolators is computationally inefficient (and prone to accumulation of floating-point error); using the Newton basis and divided-difference algorithm is to be preferred. We can see how to adapt the Newton technique to Hermite interpolation by considering a simple case, say with two nodes \( \{x_0 < x_1\} \). Let \( \Delta x > 0 \) be “small” (smaller than \( x_1 - x_0 \) will do) and consider the problem of interpolating the values of a function \( f(x) \) at the four nodes \( \{x_0, x_0+\Delta x, x_1, x_1+\Delta x\} \). The corresponding Newton basis is \( \{1, (x-x_0), (x-x_0)[x-(x_0+\Delta x)], (x-x_0)[x-(x_0+\Delta x)][(x-x_1)]\} \). The first column of the divided difference table will contain the quotients

\[
\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}, \quad \frac{f'(x_1)-f(x_0+\Delta x)}{x_1-x_0-\Delta x}, \quad \frac{f(x_1+\Delta x)-f(x_1)}{\Delta x}.
\]

\(^{(12)}\) Of course this is a simple application of the considerations of §1.3D above (p. 2). However, this more elementary approach to the question may help the reader toward an intuitive understanding of what’s going on.

\(^{(13)}\) See pp. 5–6 above, and what follows them.
If we formally take the limit as \( \Delta x \to 0^+ \), these become
\[
f'(x_0), f[x_0, x_1], f'(x_1);
\]
each of the formal divided differences \( f[x_0, x_0] \) and \( f[x_1, x_1] \) that we would get by simply “plugging in \( \Delta x = 0 \)” is replaced by the corresponding derivative. Similar things happen in the other columns, and we get an expansion
\[
p_3(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f[x_0, x_1] - f'(x_0)}{x_1 - x_0}(x - x_0)^2 \\
+ \frac{f'(x_0) + f'(x_1) + 2f[x_0, x_1]}{(x_1 - x_0)^2}(x - x_1)
\]
which (as the reader should check) interpolates as we wished and thus justifies the formal manipulations.

A similar argument\(^{(15)}\) would justify this method of computing “Hermite interpolators by Newton methods” in general: in fact, however, the most important case for our purposes will be the two-point, cubic polynomial case that we just worked out in detail. It should be noted that the existence and uniqueness of a Hermite-interpolating polynomial in Newton-basis form can be established by familiar abstract reasoning: if \( \{x_0, \ldots, x_n\} \) is a list of \( n + 1 \) distinct nodes, then the \( 2n + 1 \) polynomials
\[
\{1, (x - x_0), (x - x_0)^2, (x - x_0)^2(x - x_1), \ldots, (x - x_0)^2(x - x_1)^2 \cdots (x - x_n)\}
\]
form an ordered set of polynomials of which the \( n \)-th has degree \( n \), and such an ordered set is always a basis of \( P_{2n+1} \).\(^{(16)}\) From §6.1–6.2 above we already know the Hermite interpolator exists and is unique: the divided-difference table is only a computational method of determining the coefficients in the basis expansion
\[
p_{2n+1}(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)^2 + \cdots + c_{2n+1}(x - x_0)^2(x - x_1)^2 \cdots (x - x_n)
\]
of the Hermite interpolator with respect to this particular basis of the vector space \( P_{2n+1} \).

Here is an incarnation of the divided-difference algorithm for interpolating function values only: one can use the array

\[
\begin{array}{cccccccc}
 x_0 & c_{00} & c_{01} & c_{02} & c_{03} & \cdots & c_{0,n-1} & c_{0,n} \\
 x_1 & c_{10} & c_{11} & c_{12} & c_{13} & \cdots & c_{1,n-1} & \\
 x_2 & c_{20} & c_{21} & c_{22} & c_{23} & \vdots & \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
 x_{n-1} & c_{n-1,0} & c_{n-1,1} & \\
 x_n & c_{n0} & \\
\end{array}
\]

produced by the algorithm shown in the following (fragment of) pseudocode:

\[
\begin{align*}
& \text{for } j = 1 \text{ to } n \text{ do} \\
& \quad \text{for } i = 1 \text{ to } n - j \text{ do} \\
& \quad \quad c_{ij} \leftarrow (c_{i+1,j-1} - c_{i,j-1})/(x_{i+j} - x_i) \\
& \quad \end{align*}
\]

\(^{(14)}\) You might try carrying out the computations explicitly and seeing what happens.

\(^{(15)}\) Actually, there are better ones, as we shall see below, but we need the Hermite-Gennochi formula and we must know what happens when a divided difference is (partially) differentiated with respect to one of its arguments.

\(^{(16)}\) See p. 3 above.
in which the leftmost column holds the nodes, the zeroth column holds the values, and the top row holds \( c_{0k} = f[x_0, \ldots, x_k] \) in order. If the nodes in the leftmost column are doubled, so that the column becomes \([x_0, x_0, x_1, x_1, \ldots]^T\), this algorithm will attempt some formal divisions by zero for \( j = 1 \). It is not difficult to modify this algorithm in such a way as to input the data \( f(x_0), \ldots, f(x_n) \) in the appropriate places in the \( j = 1 \) column, compute the divided differences \( f[x_i, x_{i+1}] \) in the others, and then proceed with the algorithm (note that “\( n \)” will be replaced by “\( 2n \)” as well as the other changes). Readers who find this version of the divided-difference algorithm more congenial than those of Atkinson p. 141 \(^{17}\) or p. 10 above might enjoy playing with the necessary modifications here. (The algorithm on p. 10 above is easily adapted to handle first-derivative interpolation.)

### 6.4. Error Terms

Whether the Hermite-interpolating polynomial is written in Lagrange or Newton form, it will only approximate the originally-given function \( f(x) \) and so one will have

\[
f(x) = p_{2n+1}(x) + E(x)
\]

where the error term \( E(x) \) remains to be determined. In view of the way that one can find the coefficients in a Newton-basis expansion by going from \( \{x_0, x_1, \ldots, x_n\} \) to \( \{x_0, x_0 + \Delta x, x_1, x_1 + \Delta x, \ldots, x_n, x_n + \Delta x\} \) and passing to a limit, it would be natural to expect an error term of the form \( \frac{f^{(2n+2)}(\xi)}{(2n + 2)!} (x - x_0)^2 \cdots (x - x_n)^2 \), and this is in fact what one gets. We shall carry out the details only for the case \( n = 1 \), where the Hermite interpolating polynomial is a cubic: they are not significantly harder than those for linear interpolation of two function values.\(^{18}\) Let \( x_0 < x_1 \) be given, let \( f(x) \) be four-times-differentiable on an interval containing \([x_0, x_1]\), let \( H_3(x) \) be the Hermite interpolator of \( f(x) \) and \( f'(x) \) at \( \{x_0, x_1\} \), let \( x \in (x_0, x_1) \) be given (and not one of \( x_0 \) or \( x_1 \)) and solve the equation

\[
f(x) = H_3(x) + K(x - x_0)^2(x - x_1)^2
\]

for \( K \). Consider the new function of \( t \) defined by

\[
F(t) = f(t) - H_3(t) - K(t-x_0)^2(t-x_1)^2.
\]

Evidently \( F(x_0) = 0 = F(x_1) \) and also \( F(x) = 0 \), so there exist points \( x_0 < \eta_0 < x \) and \( x < \eta_1 < x_1 \) at which \( F'(\eta_i) = 0 \). But also

\[
F'(t) = f'(t) - H'_3(t) - 2K(t-x_0)(t-x_1)(2t-x_0-x_1)
\]

is zero at \( t = x_0 \) and \( t = x_1 \); so there exist three points \( \zeta_0, \zeta_1, \zeta_2 \) with \( x_0 < \zeta_0 < \eta_0 < \zeta_1 < \eta_1 < \zeta_2 < x_1 \) at which \( F''(\zeta_i) = 0 \), two points \( \theta_0, \theta_1 \) with \( \zeta_0 < \theta_0 < \zeta_1 < \theta_1 < \zeta_2 \) at which \( F'''(\theta_i) = 0 \), and finally a point \( \theta_0 < \xi_x < \theta_1 \) at which \( F^{(4)}(\xi_x) = 0 \). Since

\[
F^{(4)}(t) = f^{(4)}(t) - 0 - 4! \cdot K
\]

we have \( 0 = F^{(4)}(\xi_x) = f^{(4)}(\xi_x) - 0 - 4! \cdot K \) and thus \( K = \frac{f^{(4)}(\xi_x)}{4!} \), so

\[
f(x) = p_4(x) + \frac{f^{(4)}(\xi_x)}{4!} (x-x_0)^2(x-x_1)^2
\]

as we had hoped. Thus the error term has the form that we expected, at least in the case \( n = 1 \).\(^{19}\)

The fact that \( (x-x_0)^2(x-x_1)^2 \), or more generally \( (x-x_0)^2 \cdots (x-x_n)^2 \), is nonnegative for all \( x \) will play a fundamental role in our investigations of (error terms in) approximate integration.

---

\(^{17}\) Note that since this algorithm overwrites its data, it would require some nontrivial modification in the \( i=1 \) step.

\(^{18}\) For the cubic case, see Atkinson p. 162; for the details of the general first-derivative Hermite interpolation, see pp. 160–161.

\(^{19}\) For the general case, see Kincaid & Cheney, p. 369.
6.5.A. Higher-Order Hermite Approximation. We can dispense with this matter rather briefly, since we shall not be using it very much. Suppose that we are given the usual $n+1$ “nodes” \{x_0, \ldots, x_n\}, and that for each $x_j$ we are given $\alpha_j$ numbers

$$f(x_j), f'(x_j), \ldots, f^{(\alpha_j-1)}(x_j)$$

with each $\alpha_j \geq 1$. We seek a polynomial $p(x)$ of degree $N - 1$, where $N = \sum_j \alpha_j$, satisfying

$$p(x_j) = f(x_j), \quad p'(x_j) = f'(x_j), \ldots, \quad p^{(\alpha_j-1)}(x_j) = f^{(\alpha_j-1)}(x_j) \quad \text{for } 0 \leq j \leq n.$$ 

In words, at each $x_j$ we want to prescribe the values of $p(x)$ and its derivatives up to order $\alpha_j - 1$. To do this, we need to determine an element $p(\cdot)$ of the $N$-dimensional space $P_{N-1}$ for which the vector

$$[p(x_0), p'(x_0), \ldots, p^{(\alpha_0-1)}(x_0), \ldots, p(x_n), p'(x_n), \ldots, p^{(\alpha_n-1)}(x_n)]^T \in \mathbb{R}^N$$

turns out to be exactly the same as the vector

$$[f(x_0), f'(x_0), \ldots, f^{(\alpha_0-1)}(x_0), \ldots, f(x_n), f'(x_n), \ldots, f^{(\alpha_n-1)}(x_n)]^T \in \mathbb{R}^N.$$ 

This problem can be attacked and solved in exactly the same manner as the one discussed in §2.A on p. 4 above. Consider the linear map from the real vector space $P_{N-1}$ to $\mathbb{R}^N$ given by

$$A : P(\cdot) \mapsto [P(x_0), P'(x_0), \ldots, P^{(\alpha_0-1)}(x_0), \ldots, P(x_n), P'(x_n), \ldots, P^{(\alpha_n-1)}(x_n)]^T,$$

$$P_{N-1} \rightarrow \mathbb{R}^N.$$ 

As we saw on p. 2 above, if a polynomial $P(x) \in P_{N-1}$ has $N$ roots counted according to multiplicity, then in fact it must be the zero polynomial. Clearly $A(P) = [0, \ldots, 0]^T = 0 \in \mathbb{R}^N$ if and only if each $x_j$ is a root of $P(x) = 0$ of multiplicity $\alpha_j$. Suppose there were such a nonzero $P(\cdot) \in P_{N-1}$. Because the $(x-x_j)^{\alpha_j}$s are powers of distinct primes, their product

$$(x-x_0)^{\alpha_0} \cdot (x-x_1)^{\alpha_1} \cdots (x-x_n)^{\alpha_n}$$

would also divide $P(x)$ evenly: one could write

$$P(x) = (x-x_0)^{\alpha_0} \cdot (x-x_1)^{\alpha_1} \cdots (x-x_n)^{\alpha_n} \cdot Q(x)$$

(where $Q(x)$ was the quotient after division). But then $\deg P(\cdot) \geq \sum_j \alpha_j = N$, contrary to assumption; so the only such $P(\cdot)$ is the identically-zero polynomial. Thus $\ker A = \{0\}$. By the fundamental relation $\dim P_{N-1} = \dim \text{Im} A + \dim \ker A$, we have $N = \dim P_{N-1} = \dim \text{Im} A$. Thus $\text{Im} A \subseteq \mathbb{R}^N$ is a subspace of $\mathbb{R}^N$ whose dimension is $N$, and therefore $\text{Im} A = \mathbb{R}^N$. In particular,

$$[f(x_0), f'(x_0), \ldots, f^{(\alpha_0-1)}(x_0), \ldots, f(x_n), f'(x_n), \ldots, f^{(\alpha_n-1)}(x_n)]^T \in \text{Im} A,$$

a fact which tells us exactly what we wanted to know: there exists a polynomial $P(\cdot) \in P_n$ with

$$A(P) = [f(x_0), f'(x_0), \ldots, f^{(\alpha_0-1)}(x_0), \ldots, f(x_n), f'(x_n), \ldots, f^{(\alpha_n-1)}(x_n)]^T,$$

i.e., a polynomial of the desired degree that interpolates $f(\cdot)$—together with its prescribed derivatives up to the prescribed order—at the nodes $\{x_0, \ldots, x_n\}$. In view of $\ker A = \{0\}$, we already knew that the solution of the problem was unique.

(20) The setting is exactly that of Atkinson, p. 163, except that we have $n+1$ nodes instead of $n$ of them, for the sake of uniformity of notation (in comparison with what we have been doing with interpolation of function values, and indeed with simple Hermite approximation of a function and its first derivative at two points).
It is crucial that all the derivatives from the 0-th (= value) to the \((\alpha_j - 1)\)-st were interpolated at each of the nodes \(x_j\). Examples show that interpolation may fail when one “skips a derivative” at a node. However, the order to which the derivatives are interpolated may vary from node to node; e.g., using a cubic polynomial one can have \(\alpha_0 = 3\) and \(\alpha_1 = 1\), so that one interpolates \(f(x_0), f'(x_0)\) and \(f''(x_0)\) but interpolates only \(f(x_1)\).

6.5.B. Error Expressions Let us think of ourselves as remaining in the context of the previous § and think about the following subject: if the function \(f(x)\) has derivatives of order up to and including \(N\) at each of the nodes \(x_j\), then it is the fact that it exists that will be important. Since \(\Psi(x)\) is a solution of the equation \(f(x) + \alpha_j \cdot (x - x_j)^{\alpha_j}\) has a zero at each endpoint of \(I\) it is crucial that \(\Psi(t)\) be given: it will be held fixed throughout the following argument. Since \(\alpha_j\) may lie to the right of the largest \(x_j\), we can assume with no loss of generality that \(\alpha_j\) is a fixed number, not a dummy variable! Yes, one can explicitly compute \(\alpha_j\) and also \(\Psi(t)\) is non-zero at each endpoint of \(I\).

Definition: For a given set of nodes—possibly with repetitions, each then being taken with multiplicity \(\alpha_j\)—the name error factor, or error product, denotes the product

\[
\Psi(x) = (x - x_0)^{\alpha_0} \cdot (x - x_1)^{\alpha_1} \ldots (x - x_n)^{\alpha_n}.
\]

For real-valued functions the basic error estimate is given by

**Theorem:** Let \(P(x)\) be the solution of the interpolation problem of the preceding §, and suppose the interpolated function \(f(x)\) has derivatives of order up to and including \(N = \sum_j \alpha_j\). Then for each point \(x \in I\) there exists (at least one) \(\xi_x \in I\)—depending on \(x\)—for which

\[
f(x) = P(x) + \sum_{j=1}^{N} \frac{f^{(\alpha_j)}(\xi_x)}{(\alpha_j)!} (x - x_0)^{\alpha_j} \cdot (x - x_1)^{\alpha_j} \ldots (x - x_n)^{\alpha_j}.
\]

(The simplest case of this theorem is the one in which all the \(\alpha_j = 1\); it is Atkinson’s Thm. 3.2, pp. 134–135.)

**Proof.** Let \(x \in I\) be given: it will be held fixed throughout the following argument. Since \(P(x)\) takes the same value as \(f(x)\) and “the error is zero” whenever \(x\) equals some \(x_j\)—so our error term is correct for trivial reasons—we can assume with no loss of generality that \(x\) is not one of the \(x_j\)’s. Consequently the value \(\Psi(x) = (x - x_0)^{\alpha_0} \cdot (x - x_1)^{\alpha_1} \ldots (x - x_n)^{\alpha_n}\) does not = 0, and so it is trivial to find a constant \(K\) that is a solution of the equation

\[
f(x) = P(x) + K \cdot (x - x_0)^{\alpha_0} \cdot (x - x_1)^{\alpha_1} \ldots (x - x_n)^{\alpha_n}.
\]

{Remember, \(x\) is a fixed number, not a dummy variable! Yes, one can explicitly compute \(K\), but don’t bother—it’s the fact that it exists that will be important.) Now using that constant \(K\), consider the function of the “new variable” \(t\) given by

\[
G(t) = f(t) - P(t) - K \cdot \Psi(t) = f(t) - P(t) - K \cdot (t - x_0)^{\alpha_0} \cdot (t - x_1)^{\alpha_1} \ldots (t - x_n)^{\alpha_n}.
\]

The interpolation properties of \(P(t)\) relative to \(f(t)\) and its derivatives tell us that

\[
G(x_j) = 0, G'(x_j) = 0, \ldots, G^{(\alpha_j - 1)}(x_j) = 0
\]

since \(\Psi(t)\) has a zero of order \(\alpha_j\) at each \(x_j\). Moreover, \(K\) has been chosen to make \(G(x) = 0\) (remember, this means \(G(t) = 0\) when you plug in \(t = x\)). Let \(I\) be the smallest closed interval containing \(x_0, \ldots, x_n\) and also \(x\) (\(x\) may lie to the right of the largest \(x_j\) or to the left of the smallest, if we wish). Then \(G(t)\) has a zero at each endpoint of \(I\).
For a natural number \( k \), let us say that a (sufficiently differentiable) function \( H(t) \) has at least \( k \) zeros counted according to multiplicity in the interval \( I \) if there are distinct points \( t_1, \ldots, t_r \) (\( r \geq 1 \)) in \( I \) and natural numbers \( 1 \leq n_1, \ldots, n_r \) such that

\[
H(t_i) = 0, \ldots, H^{(n_i-1)}(t_i) = 0
\]

for each \( i = 1, \ldots, r \), with \( \sum_j n_j \geq k \). Then if \( H(t) \) has at least \( k \) zeros in \( I \) counted according to multiplicity, its derivative \( H'(t) \) must have at least \( k - 1 \) of them. The reason is that \( H'(t) \) has at least \( r - 1 \) zeros in \( I \) that are not the same as the \( t_i \)'s—namely, the zeros whose existence is guaranteed by Rolle’s theorem.\(^{(21)}\) Moreover, \( H'(t) \) has at least \( n_i - 1 \) zeros at \( t_i \) whenever \( n_i > 1 \)—in the obvious sense that its derivatives are zero at \( t_i \) out to order at least \( n_i - 2 \).\(^{(22)}\) Thus \( H'(t) \) has at least \( \sum_{i=1}^{r} (n_i - 1) + (r - 1) = \sum_{i=1}^{r} n_i - 1 \geq k - 1 \)

zeros in \( I \) counted according to multiplicity.

Returning to our function \( G(t) \), we observe that it has at least \( \sum_j \alpha_j + 1 \) zeros in \( I \) counted according to multiplicity. It follows by applying the argument of the preceding paragraph \( N = \sum_j \alpha_j \) times that \( G'(t) \) has at least \( \sum_j \alpha_j \) zeros, \( \ldots, \) and \( G^{(\sum_j \alpha_j)}(t) \) has at least one zero \( \xi_x \in I \). It is much easier to compute the \( N \)-th derivative of \( G(t) \) than it looks, because

\[
\Psi(x) = (x - x_0)^{\alpha_0} \cdot (x - x_1)^{\alpha_1} \cdots (x - x_n)^{\alpha_n} = x^N + \text{lower order terms}
\]

and the \( N \)-th derivative of that is obviously just \( N! \cdot 0 + \cdots + 0 = N! \). Moreover, since \( \deg P(\cdot) \leq N - 1 \), its \( N \)-th derivative is \( \equiv 0 \). So

\[
0 = G^N(\xi_x) = f^N(\xi_x) - K \cdot N!.
\]

Solve this equation for \( K = \frac{f^N(\xi_x)}{N!} \), and the original equation

\[
f(x) = P(x) + K \cdot (x - x_0)^{\alpha_0} \cdot (x - x_1)^{\alpha_1} \cdots (x - x_n)^{\alpha_n}
\]

that defined \( K \) can be rewritten—by plugging in this value of \( K \)—as

\[
f(x) = P(x) + \frac{f(\sum_j \alpha_j)(\xi_x)}{(\sum_j \alpha_j)!} \cdot (x - x_0)^{\alpha_0} \cdot (x - x_1)^{\alpha_1} \cdots (x - x_n)^{\alpha_n} = P(x) + \frac{f^{(N)}(\xi_x)}{N!} \cdot \Psi(x)
\]

which is what the theorem advertised.


This is only a small addendum to §4 above and to Atkinson’s treatment of these subjects on pp. 139–144.

(7.1) Newton coefficients can be computed by divided-difference tables even when one wants to interpolate derivatives—and therefore some of the nodes are equal. One simply replaces “indicated divided differences” involving equal nodes with appropriate values of \( \frac{f^{(k)}(x)}{k!} \). For example, to find the 3rd-degree

\(\text{(21)}\) Recall this theorem: between (meaning strictly between) any two points at which a differentiable function takes the value zero, there is a point at which its derivative takes the value zero.

\(\text{(22)}\) For example, look at the Taylor series with base point \( t_i \).
polynomial with \( f(0) = 0, f(1) = 1, f'(1) = 3, f''(1) = 6 \) {and which we know will have to be \( x^3 \)}, we put a single node \( x_0 = 1 \) and 3 nodes \( x_1 = x_2 = x_3 \) at 1; the divided-difference table takes the form

\[
\begin{array}{cccc}
    x & f & f[.] & f[.,] & f[,,] \\
    0 & 0 & 1 & 2 & 1 \\
    1 & 1 & 3/1 & 6/2 \\
    1 & 1 & 3/1 \\
    1 & 1 \\
\end{array}
\]

and consequently

\[
f(x) = 0 + (x - 0) + 2 \cdot (x - 0)(x - 1) + 1 \cdot (x - 0)(x - 1)^2 \\
= x + 2x^2 - 2x + x^3 - 2x^2 + x = x^3
\]
as one might have expected, or at least hoped.

(2) While the algorithm Interp\( (d, x, n, t, p) \) on pp. 141–142 is the efficient nested-multiplication way to compute the value of a polynomial whose Newton-divided-difference coefficients are known, one computes valuable information in the course of the evaluation and sometimes that information is worth saving. Suppose that

\[
p(x) = d_0 + d_1(x - x_0) + \cdots + d_n(x - x_0)\cdots(x - x_{n-1})
\]

and suppose that we set

\[
b_n = d_n
\]

and then for \( j = n - 1, n - 2, \ldots, 0 \) do

\[
b_j = d_j + b_{j+1} \cdot (t - x_j).
\]

This is just Interp\( (d, x, n, t, p) \), except that we are saving the result \( p \) at the \( j \)-th step, \( j = n, n - 1, \ldots, 0 \) under the name \( b_j \). If we rewrite the defining equations of the recursion—with a shift of indices—in the form

\[
d_{j-1} = b_j \cdot (x_{j-1} - t) + b_{j-1}
\]

and substitute these into the original form of \( p(x) \)—but written with its terms in decreasing order of degrees (and indices)—we get

\[
p(x) = b_n(x - x_{n-1})(x - x_{n-2})\cdots(x - x_1)(x - x_0) \\
+ [b_n(x_{n-1} - t) + b_{n-1}](x - x_{n-2})\cdots(x - x_1)(x - x_0) \\
+ \vdots \\
+ [b_2(x_1 - t) + b_1](x - x_0) \\
+ [b_1(x_0 - t) + b_0]
\]

and if we combine the term containing \( b_j \) on the \( j \)-th line \( (j = n, n - 1, \ldots \) reading downward) with the term containing \( b_j \) on the line immediately below it, this becomes

\[
p(x) = b_n(x - x_{n-2})\cdots(x - x_1)(x - x_0)(x - t) \\
+ b_{n-1}(x - x_{n-3})\cdots(x - x_1)(x - x_0)(x - t) \\
+ \vdots \\
+ b_2(x - x_0)(x - t) + b_1(x - t) + b_0
\]
which is the Newton form for \( p(x) \) with centers \( t, x_0, \ldots, x_{n-1} \): the number \( t \) is “shifted in” as the 0-th node, the previous \( n \)-th node is “shifted out,” and all the other nodes have their indices shifted up one click. If the polynomial is thought of with these new centers, then it is well adapted for calculations at points \( x \) near \( t \), since the factor \( x - t \) tends to keep the later terms small. If the process is repeated so that the nodes become \( t, t, x_0, \ldots, x_{n-2} \), then in the expansion

\[
p(x) = c_0 + c_1(x - t) + c_2(x - t)^2 + \cdots
\]

it is evident that \( c_1 = p[t, t] = p'(t) \), so one has computed the derivative of \( p(x) \) at \( x = t \); iteration of this process will eventually put \( p(x) \) into the Taylor form

\[
p(x) = p(t) + \frac{p'(t)}{1!} (x - t) + \frac{p''(t)}{2} (x - t)^2 + \cdots.
\]

A clever person can use this algorithm in all sorts of interesting ways. At the very least, it gives a non-obvious way of differentiating Newton polynomials—or indeed polynomials written in the usual way, since (if repeated nodes are allowed) the usual Taylor-power-basis monomials \((x - a)^k\) are Newton polynomials.