Numerical Mathematics and Computing
(Ward Cheney David Kincaid) International Edition.

at each of the points $x_0, x_1, \ldots, x_k$. (Its form is chosen for precisely this reason.) Now we adjust the parameter c so that the new polynomial takes the value y_{k+1} at x_{k+1}. Imposing this condition, we obtain

$$p(x_{k+1}) + c(x_{k+1} - x_0)(x_{k+1} - x_1) \cdots (x_{k+1} - x_k) = y_{k+1}$$

The proper value of c can be obtained from this equation because none of the factors $x_{k+1} - x_i$, for $0 \leq i \leq k$, can be zero. Remember our original assumption that the x_i's are all distinct.

easoning

This analysis is an example of inductive reasoning. We have shown that the process can be started and that it can be continued. Hence, the following formal statement has been partially justified:

■ Theorem 1

Theorem on Existence of Polynomial Interpolation

If points $x_0, x_1, \ldots, x_n$ are distinct, then for arbitrary real values $y_0, y_1, \ldots, y_n$, there is a unique polynomial p of degree at most n such that $p(x_i) = y_i$ for $0 \leq i \leq n$.

$p(x)$

Two parts of this formal statement must still be established. First, the degree of the polynomial increases by at most 1 in each step of the inductive argument. At the beginning, the degree was at most 0, so at the end, the degree is at most n.

s of $p(x)$

Second, we establish the uniqueness of the polynomial p. Suppose that another polynomial q claims to accomplish what p does; that is, q is also of degree at most n and satisfies $q(x_i) = y_i$ for $0 \leq i \leq n$. Then the polynomial $p - q$ is of degree at most n and takes the value 0 at $x_0, x_1, \ldots, x_n$. Recall, however, that a *nonzero* polynomial of degree n can have at most n roots. We conclude that $p = q$, which establishes the uniqueness of p.

Interpolating Polynomial: Newton Form

In Example 2, we found the Lagrange form of the interpolating polynomial:

$$p_2(x) = -36\left(x - \frac{1}{4}\right)(x - 1) - 16\left(x - \frac{1}{3}\right)(x - 1) + 14\left(x - \frac{1}{3}\right)\left(x - \frac{1}{4}\right)$$

It can be simplified to

$$p_2(x) = -\frac{79}{6} + \frac{349}{6}x - 38x^2$$

Now, we learn that this polynomial can be written in another form called the nested Newton form:

wton's Form

$$p_2(x) = 2 + \left(x - \frac{1}{3}\right)\left[36 + \left(x - \frac{1}{4}\right)(-38)\right]$$

It involves the fewest arithmetic operations and is recommended for evaluating $p_2(x)$. It cannot be overemphasized that the Newton and Lagrange forms are just two different derivations for precisely the same polynomial. The Newton form has the advantage of easy extensibility to accommodate additional data points.

The preceding discussion provides a method for constructing an interpolating polynomial. The method is known as the **Newton algorithm**, and the resulting polynomial is the **Newton form of the interpolating polynomial**.

EXAMPLE 3 Using the Newton algorithm, find the interpolating polynomial of least degree for this table:

x	0	1	−1	2	−2
y	−5	−3	−15	39	−9

Solution In the construction, five successive polynomials appear; these are labeled p_0, p_1, p_2, p_3, and p_4. The polynomial p_0 is defined to be

$$p_0(x) = -5$$

Polynomials p_0, p_1, p_2, p_3, p_4

The polynomial p_1 has the form

$$p_1(x) = p_0(x) + c(x - x_0) = -5 + c(x - 0)$$

The interpolation condition placed on p_1 is that $p_1(1) = -3$. Therefore, we have $-5 + c(1 - 0) = -3$. Hence, $c = 2$, and p_1 is

$$p_1(x) = -5 + 2x$$

The polynomial p_2 has the form

$$p_2(x) = p_1(x) + c(x - x_0)(x - x_1) = -5 + 2x + cx(x - 1)$$

The interpolation condition placed on p_2 is that $p_2(-1) = -15$. Hence, we have $-5 + 2(-1) + c(-1)(-1 - 1) = -15$. This yields $c = -4$, so

$$p_2(x) = -5 + 2x - 4x(x - 1)$$

The remaining steps for $p_3(x)$ are similar. The final result is the Newton form of the interpolating polynomial:

$$p_4(x) = -5 + 2x - 4x(x - 1) + 8x(x - 1)(x + 1) + 3x(x - 1)(x + 1)(x - 2) \quad \blacksquare$$

Adding a New Term

Later, we develop a better algorithm for constructing the Newton interpolating polynomial. Nevertheless, the method just explained is a systematic one and involves very little computation. An important feature to notice is that each new polynomial in the algorithm is obtained from its predecessor by adding a new term. Thus, at the end, the final polynomial exhibits all the previous polynomials as constituents.

Nested Form

Before continuing, let's rewrite the Newton form of the interpolating polynomial for efficient evaluation.

EXAMPLE 4 Write the polynomial p_4 of Example 3 in *nested* form and use it to evaluate $p_4(3)$.

Solution We write p_4 as

$$p_4(x) = -5 + x\big(2 + (x - 1)\big(-4 + (x + 1)\big(8 + (x - 2)3\big)\big)\big)$$

Nested Form $p_4(x)$

Therefore, we obtain

$$\begin{aligned} p_4(3) &= -5 + 3\big(2 + 2\big(-4 + 4(8 + 3)\big)\big) \\ &= 241 \end{aligned}$$

Another solution, also in nested form, is

$$p_4(x) = -5 + x\big(4 + x\big(-7 + x(2 + 3x)\big)\big)$$

from which we obtain

$$p_4(3) = -5 + 3\big(4 + 3\big(-7 + 3(2 + 3 \cdot 3)\big)\big) = 241$$

This form is obtained by expanding and systematic factoring of the original polynomial. It is also known as a **nested form**, and its evaluation is by **nested multiplication**. ■

To describe nested multiplication in a formal way (so that it can be translated into a pseudocode), consider a general polynomial in the Newton form. It might be

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

The nested form of $p_n(x)$ is

$$p_n(x) = a_0 + (x - x_0)\big(a_1 + (x - x_1)\big(a_2 + \cdots + (x - x_{n-2})\big(a_{n-1} + (x - x_{n-1})(a_n)\big)\cdots\big)\big)$$

The **Newton interpolation polynomial** can be written succinctly as

$$p_n(x) = \sum_{i=0}^{n} a_i \prod_{j=0}^{i-1}(x - x_j) \tag{3}$$

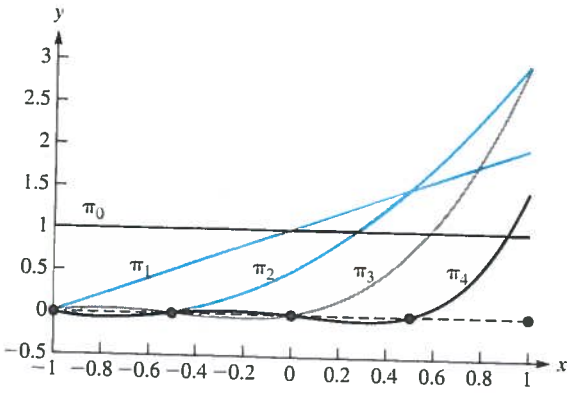
Here $\prod_{j=0}^{-1}(x - x_j)$ is interpreted to be 1. Also, we can write it as

$$p_n(x) = \sum_{i=0}^{n} a_i \, \pi_i(x)$$

where

$$\pi_i(x) = \prod_{j=0}^{i-1}(x - x_j) \tag{4}$$

Figure 4.2 shows the first few Newton polynomials: $\pi_0(x)$, $\pi_1(x)$, $\pi_2(x)$, $\pi_3(x)$, $\pi_4(x)$, and $\pi_5(x)$.



In evaluating $p(t)$ for a given numerical value of t, we naturally start with the innermost parentheses, forming successively the following quantities:

$$\begin{aligned} v_0 &= a_n \\ v_1 &= v_0(t - x_{n-1}) + a_{n-1} \\ v_2 &= v_1(t - x_{n-2}) + a_{n-2} \\ &\vdots \\ v_n &= v_{n-1}(t - x_0) + a_0 \end{aligned}$$

The quantity v_n is now $p(t)$. In the following pseudocode, a subscripted variable is *not* needed for v_i. Instead, we can write

Evaluation of Interpolation Polynomial Pseudocode

```
integer i, n;  real t, v;  real array (a_i)_{0:n}, (x_i)_{0:n}
v ← a_n
for i = n − 1 to 0 step −1
    v ← v(t − x_i) + a_i
end for
```

Here, the array $(a_i)_{0:n}$ contains the $n+1$ coefficients of the Newton form of the interpolating polynomial (3) of degree at most n, and the array $(x_i)_{0:n}$ contains the $n+1$ nodes x_i.

Calculating Coefficients a_i Using Divided Differences

We turn now to the problem of determining the coefficients $a_0, a_1, \ldots, a_n$ efficiently. Again we start with a table of values of a function f:

x	x_0	x_1	x_2	$\cdots$	x_n
$f(x)$	$f(x_0)$	$f(x_1)$	$f(x_2)$	$\cdots$	$f(x_n)$

The points $x_0, x_1, \ldots, x_n$ are assumed to be distinct, but *no* assumption is made about their positions on the real line.

Previously, we established that for each $n = 0, 1, \ldots$, there exists a unique polynomial p_n such that

Unique Polynomial p_n

- The degree of p_n is at most n.
- $p_n(x_i) = f(x_i)$ for $i = 0, 1, \ldots, n$.

It was shown that p_n can be expressed in the Newton form

$$\begin{aligned} p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots \\ + a_n(x - x_0)\cdots(x - x_{n-1}) \end{aligned}$$

A crucial observation about p_n is that the coefficients $a_0, a_1, \ldots$ do not depend on n. In other words, p_n is obtained from p_{n-1} by adding one more term, without altering the coefficients already present in p_{n-1} itself. This is because we began with the hope that p_n could be expressed in the form

$$p_n(x) = p_{n-1}(x) + a_n(x - x_0)\cdots(x - x_{n-1})$$

Adding One Term at a Time

and discovered that it was indeed possible.

A way of systematically determining the unknown coefficients $a_0, a_1, \ldots, a_n$ is to set x equal in turn to $x_0, x_1, \ldots, x_n$ in the Newton form (3) and to write down the resulting equations:

$$\begin{cases} f(x_0) = a_0 \\ f(x_1) = a_0 + a_1(x_1 - x_0) \\ f(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\ \quad \text{etc.} \end{cases} \tag{5}$$

The compact form of Equations (5) is

$$f(x_k) = \sum_{i=0}^{k} a_i \prod_{j=0}^{i-1}(x_k - x_j) \qquad (0 \leqq k \leqq n) \tag{6}$$

Equations (5) can be solved for the a_i's in turn, starting with a_0. Then we see that a_0 depends on $f(x_0)$, that a_1 depends on $f(x_0)$ and $f(x_1)$, and so on. In general, a_k depends on $f(x_0)$, $f(x_1), \ldots, f(x_k)$. In other words, a_k depends on the values of f at the nodes $x_0, x_1, \ldots, x_k$. The traditional notation is

vided Difference
der k

$$a_k = f[x_0, x_1, \ldots, x_k] \tag{7}$$

This equation defines $f[x_0, x_1, \ldots, x_k]$. The quantity $f[x_0, x_1, \ldots, x_k]$ is called the **divided difference of order k** for f. Notice also that the coefficients $a_0, a_1, \ldots, a_k$ are *uniquely* determined by System (6). Indeed, there is no possible choice for a_0 other than $a_0 = f(x_0)$. Similarly, there is now no choice for a_1 other than $[f(x_1) - a_0]/(x_1 - x_0)$ and so on. Using Equations (5), we see that the first few divided differences can be written as

, a_2 Divided
ences

$$a_0 = f(x_0)$$

$$a_1 = \frac{f(x_1) - a_0}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$a_2 = \frac{f(x_2) - a_0 - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} = \frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

EXAMPLE 5 For the table:

x	1	−4	0
$f(x)$	3	13	−23

determine the quantities $f[x_0]$, $f[x_0, x_1]$, and $f[x_0, x_1, x_2]$.

Solution We write out the system of Equations (5) for this concrete case:

$$\begin{cases} 3 = a_0 \\ 13 = a_0 + a_1(-5) \\ -23 = a_0 + a_1(-1) + a_2(-1)(4) \end{cases}$$

The solution is $a_0 = 3$, $a_1 = -2$, and $a_2 = 7$. Hence, for this function, $f[1] = 3$, $f[1, -4] = -2$, and $f[1, -4, 0] = 7$. ■

With this new notation, the **Newton form of the interpolating polynomial** takes the form

n Form of
olating
omial

$$p_n(x) = \sum_{i=0}^{n} \left\{ f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1}(x - x_j) \right\} \tag{8}$$

with the usual convention that $\prod_{j=0}^{-1}(x - x_j) = 1$. Notice that the coefficient of x^n in p_n is $f[x_0, x_1, \ldots, x_n]$ because the term x^n occurs only in $\prod_{j=0}^{n-1}(x - x_j)$. It follows that if f is a polynomial of degree $\leqq n - 1$, then $f[x_0, x_1, \ldots, x_n] = 0$.

We return to the question of how to compute the required divided differences $f[x_0, x_1, \ldots, x_k]$. From System (5) or System (6), it is evident that this computation can be performed *recursively*. We simply solve Equation (6) for a_k as follows:

$$f(x_k) = a_k \prod_{j=0}^{k-1}(x_k - x_j) + \sum_{i=0}^{k-1} a_i \prod_{j=0}^{i-1}(x_k - x_j)$$

and

$$a_k = \frac{f(x_k) - \displaystyle\sum_{i=0}^{k-1} a_i \prod_{j=0}^{i-1}(x_k - x_j)}{\displaystyle\prod_{j=0}^{k-1}(x_k - x_j)}$$

Using Equation (7), we have

$$f[x_0, x_1, \ldots, x_k] = \frac{f(x_k) - \displaystyle\sum_{i=0}^{k-1} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1}(x_k - x_j)}{\displaystyle\prod_{j=0}^{k-1}(x_k - x_j)} \tag{9}$$

■ **Algorithm**

Computing the Divided Differences of f (10)

- Set $f[x_0] = f(x_0)$.
- For $k = 1, 2, \ldots, n$, compute $f[x_0, x_1, \ldots, x_k]$ by Equation (9).

EXAMPLE 6 Using Algorithm (10), write out the divided differences formulas for $f[x_0]$, $f[x_0, x_1]$, $f[x_0, x_1, x_2]$, and $f[x_0, x_1, x_2, x_3]$.

Solution

First Four Divided Differences

$$f[x_0] = f(x_0)$$

$$f[x_0, x_1] = \frac{f(x_1) - f[x_0]}{x_1 - x_0}$$

$$f[x_0, x_1, x_2] = \frac{f(x_2) - f[x_0] - f[x_0, x_1](x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}$$

$$f[x_0, x_1, x_2, x_3] = \frac{f(x_3) - f[x_0] - f[x_0, x_1](x_3 - x_0) - f[x_0, x_1, x_2](x_3 - x_0)(x_3 - x_1)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)}$$

■

Operation Count

Algorithm (10) is easily programmed and is capable of computing the divided differences $f[x_0]$, $f[x_0, x_1]$, $\ldots$, $f[x_0, x_1, \ldots, x_n]$ at the cost of $\frac{1}{2}n(3n + 1)$ additions, $(n - 1)(n - 2)$ multiplications, and n divisions excluding arithmetic operations on the indices. Now a more refined method is presented for which the pseudocode requires only three statements (!) and costs only $\frac{1}{2}n(n + 1)$ divisions and $n(n + 1)$ additions.

At the heart of the new method is the following remarkable theorem:

Theorem 2 **Recursive Property of Divided Differences**

sive Property

The divided differences obey the formula

$$f[x_0, x_1, \ldots, x_k] = \frac{f[x_1, x_2, \ldots, x_k] - f[x_0, x_1, \ldots, x_{k-1}]}{x_k - x_0} \tag{11}$$

Proof Since $f[x_0, x_1, \ldots, x_k]$ was defined to be equal to the coefficient a_k in the Newton form of the interpolating polynomial p_k of Equation (3), we can say that $f[x_0, x_1, \ldots, x_k]$ is the coefficient of x^k in the polynomial p_k of degree $\leqq k$, which interpolates f at $x_0, x_1, \ldots, x_k$. Similarly, $f[x_1, x_2, \ldots, x_k]$ is the coefficient of x^{k-1} in the polynomial q of degree $\leqq k-1$, which interpolates f at $x_1, x_2, \ldots, x_k$. Likewise, $f[x_0, x_1, \ldots, x_{k-1}]$ is the coefficient of x^{k-1} in the polynomial p_{k-1} of degree $\leqq k-1$, which interpolates f at $x_0, x_1, \ldots, x_{k-1}$. The three polynomials p_k, q, and p_{k-1} are intimately related. In fact,

$$p_k(x) = q(x) + \frac{x - x_k}{x_k - x_0}[q(x) - p_{k-1}(x)] \tag{12}$$

To establish Equation (12), observe that the right side is a polynomial of degree at most k. Evaluating it at x_i, for $1 \leqq i \leqq k-1$, results in $f(x_i)$:

$$\begin{aligned} q(x_i) + \frac{x_i - x_k}{x_k - x_0}[q(x_i) - p_{k-1}(x_i)] &= f(x_i) + \frac{x_i - x_k}{x_k - x_0}[f(x_i) - f(x_i)] \\ &= f(x_i) \end{aligned}$$

Similarly, evaluating it at x_0 and x_k gives $f(x_0)$ and $f(x_k)$, respectively. By the uniqueness of interpolating polynomials, the right side of Equation (12) must be $p_k(x)$, and Equation (12) is established.

Completing the argument to justify Equation (11), we take the coefficient of x^k on both sides of Equation (12). The result is Equation (11). Indeed, we see that $f[x_1, x_2, \ldots, x_k]$ is the coefficient of x^{k-1} in q, and $f[x_0, x_1, \ldots, x_{k-1}]$ is the coefficient of x^{k-1} in p_{k-1}. ■

Notice that $f[x_0, x_1, \ldots, x_k]$ is *not* changed if the nodes $x_0, x_1, \ldots, x_k$ are permuted. Thus, for example, we have

$$f[x_0, x_1, x_2] = f[x_1, x_2, x_0]$$

The reason is that $f[x_0, x_1, x_2]$ is the coefficient of x^2 in the quadratic polynomial interpolating f at x_0, x_1, x_2, whereas $f[x_1, x_2, x_0]$ is the coefficient of x^2 in the quadratic polynomial interpolating f at x_1, x_2, x_0. These two polynomials are, of course, the same! A formal statement in mathematical language is as follows:

Theorem 3 **Invariance Theorem**

ıce Property

The divided difference $f[x_0, x_1, \ldots, x_k]$ is invariant under all permutations of the arguments $x_0, x_1, \ldots, x_k$.

Since the variables $x_0, x_1, \ldots, x_k$ and k are arbitrary, the recursive Formula (11) can also be written as

$$f[x_i, x_{i+1}, \ldots, x_{j-1}, x_j] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_j] - f[x_i, x_{i+1}, \ldots, x_{j-1}]}{x_j - x_i} \tag{13}$$

The first three divided differences are thus

First Three Divided Differences

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

Using Formula (13), we can construct a divided-difference table for a function f. It is customary to arrange it as follows (here $n = 3$):

Divided-Difference Table

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$	$f[\,,\,,\,,\,]$
x_0	$\boxed{f[x_0]}$			
		$\boxed{f[x_0, x_1]}$		
x_1	$f[x_1]$		$\boxed{f[x_0, x_1, x_2]}$	
		$f[x_1, x_2]$		$\boxed{f[x_0, x_1, x_2, x_3]}$
x_2	$f[x_2]$		$f[x_1, x_2, x_3]$	
		$f[x_2, x_3]$		
x_3	$f[x_3]$			

In the table, the coefficients along the top diagonal are the ones needed to form the Newton form of the interpolating polynomial (3).

EXAMPLE 7 Construct a divided-difference diagram for the function f given in the following table, and write out the Newton form of the interpolating polynomial.

x	1	$\frac{3}{2}$	0	2
$f(x)$	3	$\frac{13}{4}$	3	$\frac{5}{3}$

Solution The first entry is

$$f[x_0, x_1] = \frac{\left(\frac{13}{4} - 3\right)}{\left(\frac{3}{2} - 1\right)} = \frac{1}{2}$$

After completion of column 3, the first entry in column 4 is

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{\frac{1}{6} - \frac{1}{2}}{0 - 1} = \frac{1}{3}$$

The complete diagram is

Sample Divided-Difference Table

x	$f[\,]$	$f[\,,\,]$	$f[\,,\,,\,]$	$f[\,,\,,\,,\,]$
1	$\boxed{3}$			
		$\boxed{\frac{1}{2}}$		
$\frac{3}{2}$	$\frac{13}{4}$		$\boxed{\frac{1}{3}}$	
		$\frac{1}{6}$		$\boxed{-2}$
0	3		$-\frac{5}{3}$	
		$-\frac{2}{3}$		
2	$\frac{5}{3}$			

Thus, we obtain

$$p_3(x) = 3 + \tfrac{1}{2}(x-1) + \tfrac{1}{3}(x-1)\left(x-\tfrac{3}{2}\right) - 2(x-1)\left(x-\tfrac{3}{2}\right)x$$ ■

Algorithms and Pseudocode

Turning next to algorithms, we suppose that a table for f is given at points $x_0, x_1, \ldots, x_n$ and that all the divided differences $a_{ij} \equiv f[x_i, x_{i+1}, \ldots, x_j]$ are to be computed. The following pseudocode accomplishes this:

```
integer i, j, n;  real array (a_ij)_{0:n×0:n}, (x_i)_{0:n}
for i = 0 to n
    a_i0 ← f(x_i)
end for
for j = 1 to n
    for i = 0 to n − j
        a_ij ← (a_{i+1,j−1} − a_{i,j−1})/(x_{i+j} − x_i)
    end for
end for
```

Observe that the coefficients of the interpolating polynomial (3) are stored in the first row of the array $(a_{ij})_{0:n\times 0:n}$.

If the divided differences are being computed for use only in constructing the Newton form of the interpolation polynomial

$$p_n(x) = \sum_{i=0}^{n} a_i \prod_{j=0}^{i-1}(x - x_j)$$

where $a_i = f[x_0, x_1, \ldots, x_i]$, there is *no* need to store all of them. Only $f[x_0]$, $f[x_0, x_1]$, $\ldots$, $f[x_0, x_1, \ldots, x_n]$ need to be stored.

When a one-dimensional array $(a_i)_{0:n}$ is used, the divided differences can be overwritten each time from the last storage location backward so that, finally, only the desired coefficients remain. In this case, the amount of computing is the same as in the preceding case, but the storage requirements are less. (Why?) Here is a pseudocode to do this:

```
integer i, j, n;  real array (a_i)_{0:n}, (x_i)_{0:n}
for i = 0 to n
    a_i ← f(x_i)
end for
for j = 1 to n
    for i = n to j step −1
        a_i ← (a_i − a_{i−1})/(x_i − x_{i−j})
    end for
end for
```

This algorithm is more intricate, and the reader is invited to verify it—say, in the case $n = 3$.

For the numerical experiments suggested in the computer problems, the following two procedures should be satisfactory. The first is called *Coef*. It requires as input the number n and tabular values in the arrays (x_i) and (y_i). Remember that the number of points in