1978 Gerald, Curtis Appl Numerical Anal.

In the last relation of Eq (3.14) we have expressed the differences as forward differences to agree with our lozenge-diagram notation, using $\nabla^n = E^{-n}\Delta^n$. We shall use symbolic methods to derive derivative formulas in the next chapter.

3.8 INTERPOLATION WITH NONUNIFORMLY SPACED x-VALUES

When the x-values are not evenly spaced, our methods of writing an interpolating polynomial fail. A parallel development using the concept of divided differences* is possible, but perhaps the simplest approach is the Lagrangian polynomial. Data where the x-values are not equispaced often occur as the result of experimental observations, or when historical data is examined.

Suppose we have a table of data, of x- and $f(x)$-values:

x	$f(x)$
x_1	f_1
x_2	f_2
x_3	f_3
x_4	f_4

Here we do not assume uniform Δx, nor do we need the x-values arranged in a particular order. The x-values must all be distinct, however. Through these four data pairs we can pass a cubic. The Lagrangian form for this is:

$$P_3(x) = \frac{(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)}f_1 + \frac{(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)}f_2$$
$$+ \frac{(x-x_1)(x-x_2)(x-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)}f_3 + \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)}f_4.$$

Note that it is made up of four terms, each of which is a cubic in x; hence the sum is a cubic. The pattern of each term is to form the numerator as a product of linear factors of the form $(x - x_i)$, omitting one x_i in each term, the omitted value

* The first-order divided difference relative to x-values x_0 and x_1 is defined as $f[x_1, x_0] = [f(x_1) - f(x_0)]/(x_1 - x_0)$. The higher-order divided differences are defined in terms of the lower-order differences:

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

The Newton interpolating polynomials can be written in terms of these divided differences in the fashion of Section 3.3. When the x-values are evenly spaced, the formulas are identical.

(1978) 2nd Ed. Out 18-25

being used to form the denominator by replacing x in each of the numerator factors. In each term, we multiply by the f_i corresponding to the x_i omitted in the numerator factors. The Lagrangian polynomial for other degrees of interpolating polynomials employs this same pattern of forming a sum of polynomials all of the desired degree.

Example Interpolate for $f(2.3)$ from the table:

x	$f(x)$
1.1	10.6
1.7	15.2
3.0	20.3

With only three pairs of data, a quadratic is the highest-degree polynomial possible. It is:

$$P_2(x) = \frac{(x-1.7)(x-3.0)}{(1.1-1.7)(1.1-3.0)}(10.6) + \frac{(x-1.1)(x-3.0)}{(1.7-1.1)(1.7-3.0)}(15.2)$$
$$+ \frac{(x-1.1)(x-1.7)}{(3.0-1.1)(3.0-1.7)}(20.3).$$

At $x = 2.3$, we get $P_2(2.3) = 18.38$.

The arithmetic in this method is tedious, although electronic calculators are convenient for this type of computation. A computer program at the end of this chapter implements the method.

We develop the error term in the same way as in Section 3.6. Write $E(x)$ to exhibit the fact that the error of $P_n(x)$ is zero at the $n + 1$ values of x that are fitted exactly:

$$E(x) = f(x) - P_n(x) = (x - x_1)(x - x_2)\cdots(x - x_{n+1})g(x).$$

The auxiliary function $W(t)$ is

$$W(t) = f(t) - P_n(t) - (t - x_1)(t - x_2)\cdots(t - x_{n+1})g(x).$$

$W(t) = 0$ for $t = x_1, x_2, \ldots, x_{n+1}$, and at $t = x$, for a total of $n + 2$ zeros. Hence $W'(t)$ has n zeros, $W''(t)$ has $n - 1$ zeros, ..., $W^{(n+1)}(t)$ has one zero. Let ξ be the value of t at which $W^{(n+1)}(t) = 0$. Hence,

$$W^{(n+1)}(\xi) = 0 = f^{(n+1)}(\xi) - 0 - (n+1)!g(x),$$
$$g(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

The error is then

$$E(x) = (x - x_1)(x - x_2)\cdots(x - x_{n+1})\frac{f^{(n+1)}(\xi)}{(n+1)!}. \qquad (3.15)$$

The interval containing ξ lies between the largest and smallest of the x_i for interpolation. The error can be bracketed between a maximum and a minimum value only if we have information on the $(n + 1)$st derivative of the actual function $f(x)$.

3.9 INVERSE INTERPOLATION

Suppose we have a table of data such as in Table 3.6, and we are required to find the x-value corresponding to a certain value of the function, say at $y = 5.0$. We have two approaches. We can consider the y's to be the independent variable (unevenly spaced) and interpolate for x with a Lagrangian polynomial. Doing so gives $x = 2.312$. This technique is straightforward, but in some instances gives poor results, the reason being that x considered as a function of y may not be well approximated by a polynomial. This may be true even though y itself behaves quite like a polynomial. (Try inverse interpolation among three or four points on the function $y = x^2$, especially for y-values outside the given range.)

Table 3.6

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$
1.6	2.3756				
		0.8926			
1.9	3.2682		0.2963		
		1.1889		0.1079	
2.2	4.4571		0.4042		0.0365
		1.5931		0.1444	
2.5	6.0502		0.5486		
		2.1417			
2.8	8.1919				

The second method is to write y as a polynomial in x and then use the methods of Chapter 1. This will generally make it necessary to multiply out the interpolating polynomial so as to express the function in the usual polynomial form. If we use the Gauss forward polynomial, with $x_0 = 2.2$, we have

$$\begin{aligned} y_s = P_3(x_s) &= 4.4571 + s(1.5931) + \frac{(s)(s-1)}{2}(0.4042) \\ &\quad + \frac{(s+1)(s)(s-1)}{6}(0.1444) \\ &\quad + \frac{(s+1)(s)(s-1)(s-2)}{24}(0.0365) \\ &= 0.00152s^4 + 0.02103s^3 + 0.2006s^2 + 1.3700s + 4.4571. \end{aligned}$$

[We have left the polynomial in terms of s to save work; after determining the value of s corresponding to $y = 5.0$, we get x from $s = (x - 2.2)/0.3$.] The polynomial of Eq. (3.16) could, of course, be determined through the Lagrangian polynomial, but the arithmetic effort is much greater. To complete the problem, we must find the root near zero of a fourth-degree polynomial in s, which is rather tedious.

The equivalent of this second technique for inverse interpolation can be accomplished more readily by the method of successive approximations. Our Gauss forward polynomial is

$$\begin{aligned} y_s &= y_0 + s\Delta y_0 + \frac{s(s-1)}{2}\Delta^2 y_{-1} + \frac{(s+1)(s)(s-1)}{6}\Delta^3 y_{-1} \\ &\quad + \frac{(s+1)(s)(s-1)(s-2)}{24}\Delta^4 y_{-2}. \end{aligned}$$

We rearrange to solve for s in the second term:

$$\begin{aligned} s = \frac{1}{\Delta y_0}\Bigg[& y_s - y_0 - s(s-1)\frac{\Delta^2 y_{-1}}{2} - (s+1)(s)(s-1)\frac{\Delta^3 y_{-1}}{6} \\ & - (s+1)(s)(s-1)(s-2)\frac{\Delta^4 y_{-2}}{24}\Bigg]. \end{aligned} \tag{3.17}$$

The method of successive approximations finds s by first neglecting all the terms in s on the right, to give

$$s_1 = \frac{1}{6.0502 - 4.4571}(5.0 - 4.4571) = 0.34.$$

The second approximation is obtained using s_1 on the right side of Eq. (3.17), including now one more term:

$$s_2 = \frac{1}{1.5931}[5.0 - 4.4571 - (0.34)(0.34 - 1)(0.2021)] = 0.369.$$

The next approximation uses s_2 on the right and picks up another term:

$$\begin{aligned} s_3 = \frac{1}{1.5931}[&5.0 - 4.4571 - (0.369)(-0.631)(0.2021) \\ &- (1.369)(0.369)(-0.631)(0.02407)] = 0.375. \end{aligned}$$

In the same fashion, $s_4 = 0.3748$, giving

$$x = 2.2 + (0.3)(-0.3748) = 2.31244.$$

(The data in Table 3.6 are for $y = \sinh x$. Substitution in the hyperbolic sine function gives $\sinh(2.31244) = 5.00001$.)

3.10 POLYNOMIAL INTERPOLATION IN TWO DIMENSIONS

When a function u is a polynomial function* of two variables x and y, say of degree three in x and of degree two in y, we would have

$$u = f(x, y) = a_0 + a_1x + a_2y + a_3x^2 + a_4xy + a_5y^2 + a_6x^3 + a_7x^2y + a_8xy^2 + a_9x^3y + a_{10}x^2y^2 + a_{11}x^3y^2. \quad (3.18)$$

The functional relation is seen to involve many terms. If we are concerned with four independent variables (three space dimensions plus time, say), even low-degree polynomials would be quite intractable. Except for special purposes, such as when we need an explicit representation, perhaps to permit ready differentiation at an arbitrary point, we can avoid such complications by handling each variable separately. We shall treat only this case.

Note the immediate simplification of Eq. (3.18) if we let y take on a constant value, say $y = y_1$. Combining the y factors with the coefficients, we get

$$u|_{y=c} = b_0 + b_1x + b_2x^2 + b_3x^3.$$

This will be our attack in interpolating at the point (a, b) in a table of two variables—hold one variable constant, say $y = y_1$, and the table becomes a single-variable problem. The above methods then apply to give $f(a, y_1)$. If we repeat this at various values of y, $y = y_2, y_3, \ldots, y_n$, we will get a table with x constant at the value $x = a$ and with y varying. We then interpolate at $y = b$.

Example Estimate $f(1.6, 0.33)$ from the values in Table 3.7. Use quadratic interpolation in the x-direction and cubic interpolation for y. We select one of the variables to hold constant, say x. [This choice is arbitrary since we would get the same result (except for differences due to round-off) if we had chosen to hold y constant.] We decide to interpolate for y within the three rows of the table at $x = 1.0$, 1.5, and 2.0 since the desired value at $x = 1.6$ is most nearly centered within this set. We choose y-values of 0.2, 0.3, 0.4, and 0.5 so that $y = 0.33$ is centralized. The shading in Table 3.7 shows the region of fit for our polynomials.

Table 3.7 Tabulation of a function of two variables, $u = f(x, y)$.

x \ y	0.1	0.2	0.3	0.4	0.5	0.6
0.5	0.165	0.428	0.687	0.942	1.190	1.431
1.0	0.271	0.640	1.003	1.359	1.703	2.035
1.5	0.447	0.990	1.524	2.045	2.549	3.031
2.0	0.738	1.568	2.384	3.177	3.943	4.672
2.5	1.216	2.520	3.800	5.044	6.241	7.379
3.0	2.005	4.090	6.136	8.122	10.030	11.841
3.5	3.306	6.679	9.986	13.196	16.277	19.198

* We approximate a nonpolynomial function by a polynomial that agrees with the function, just as we have done with a function of one variable.

We may either use Lagrangian interpolation or derive the interpolated values using the lozenge diagram, Fig. 3.1 after making difference tables. Let us use the later method:

	y	u	Δu	Δ²u	Δ³u
x = 1.0	0.2	0.640			
			0.363		
	0.3	1.003		−0.007	
			0.356		−0.005
	0.4	1.359		−0.012	
			0.344		
	0.5	1.703			
x = 1.5	0.2	0.990			
			0.534		
	0.3	1.524		−0.013	
			0.521		−0.004
	0.4	2.045		−0.017	
			0.504		
	0.5	2.549			
x = 2.0	0.2	1.568			
			0.816		
	0.3	2.384		−0.023	
			0.793		−0.004
	0.4	3.177		−0.027	
			0.766		
	0.5	3.943			

We need the subtables from $y = 0.2$ to $y = 0.5$ since, for a cubic interpolation, four points are required. Using any convenient path through the tables with coefficients given by the lozenge, we arrive at the results:

	x	u	Δu	Δ²u
y = 0.33	1.0	1.1108		
			0.5710	
	1.5	1.6818		0.3717
			0.9427	
	2.0	2.6245		

In the last tabulation we carry one extra decimal to guard against round-off errors. Interpolating again, we get $u = 1.8406$, which we report as $u = 1.841$.

The function tabulated in Table 3.7 is $f(x, y) = e^x \sin y + y - 0.1$, so the true value is $f(1.6, 0.33) = 1.8350$. Our error of -0.006 occurs because quadratic

interpolation for x is inadequate in view of the large second difference. In retrospect, it would have been better to use quadratic interpolation for y, since the third differences of the y-subtables are small, and let x take on a third-degree relationship.

It is instructive to observe which of the values in Table 3.7 entered into our computation. The shaded rectangle covers these values. This is the "region of fit" for the interpolating polynomial that we have used. The principle of choosing values so that the point at which the interpolating polynomial is used is centered in the region of fit obviously applies here in exact analogy to the one-way table situation. It also applies to tables of three and four variables in the same way. Of course, the labor of interpolating in such multidimensional cases soon becomes burdensome.

A rectangular region of fit is not the only possibility. We may change the degree of interpolation as we subtabulate the different rows or columns. Intuitively, it would seem best to use higher-degree polynomials for the rows near the interpolating point, decreasing the degree as we get farther away. The coefficient of the error term, when this is done, will be found to be minimized thereby, though for multidimensional interpolating polynomials the error term is quite complex. The region of fit will be diamond-shaped when such tapered degree functions are used.

We may adapt the Lagrangian form of interpolating polynomial to the multidimensional case also. It is perhaps easiest to employ a process similar to the above example. Holding one variable constant, we write a series of Lagrangian polynomials for interpolation at the given value of the other variable, and then combine these values in a final Lagrange form. The net result is a Lagrangian polynomial in which the function factors are replaced by Lagrangian polynomials. The resulting expression for the above example would be:

$$
\begin{aligned}
&\frac{(y-0.3)(y-0.4)(y-0.5)}{(0.2-0.3)(0.2-0.4)(0.2-0.5)} \\
&\times\left[\frac{(x-1.5)(x-2.0)}{(1.0-1.5)(1.0-2.0)}(0.640)+\frac{(x-1.0)(x-2.0)}{(1.5-1.0)(1.5-2.0)}(0.990)+\frac{(x-1.0)(x-1.5)}{(2.0-1.0)(2.0-1.5)}(1.568)\right] \\
&+\frac{(y-0.2)(y-0.4)(y-0.5)}{(0.3-0.2)(0.3-0.4)(0.3-0.5)} \\
&\times\left[\frac{(x-1.5)(x-2.0)}{(1.0-1.5)(1.0-2.0)}(1.003)+\frac{(x-1.0)(x-2.0)}{(1.5-1.0)(1.5-2.0)}(1.534)+\frac{(x-1.0)(x-1.5)}{(2.0-1.0)(2.0-1.5)}(2.384)\right] \\
&+\frac{(y-0.2)(y-0.3)(y-0.5)}{(0.4-0.2)(0.4-0.3)(0.4-0.5)} \\
&\times\left[\frac{(x-1.5)(x-2.0)}{(1.0-1.5)(1.0-2.0)}(1.359)+\frac{(x-1.0)(x-2.0)}{(1.5-1.0)(1.5-2.0)}(2.045)+\frac{(x-1.0)(x-1.5)}{(2.0-1.0)(2.0-1.5)}(3.177)\right] \\
&+\frac{(y-0.2)(y-0.3)(y-0.4)}{(0.5-0.2)(0.5-0.3)(0.5-0.4)} \\
&\times\left[\frac{(x-1.5)(x-2.0)}{(1.0-1.5)(1.0-2.0)}(1.703)+\frac{(x-1.0)(x-2.0)}{(1.5-1.0)(1.5-2.0)}(2.549)+\frac{(x-1.0)(x-1.5)}{(2.0-1.0)(2.0-1.5)}(3.943)\right].
\end{aligned}
$$

The equation is easy to write, but its evaluation by hand is laborious. If one is writing a computer program for interpolation in such multivariate situations, the Lagrangian form is recommended. There is a special advantage in that equal spacing in the table is not required. The Lagrangian form is also perhaps the most straightforward way to write out the polynomial as an explicit function.

The cubic spline (described in Chapter 10) could also be used to interpolate in multivariate tables. Here again holding all but one variable fixed so that a series of one-way table problems is solved is perhaps the best approach. The computational effort to employ spline interpolation would be very severe.

3.11 INTERPOLATION IN A COMPUTER PROGRAM

The first of the following subroutines (Program 1, Fig. 3.2) interpolates using the Lagrangian form of interpolating polynomial. The coefficients of each term are built up as the ratio of $(x - x_j)/(x_i - x_j)$ with j varying from 1 to N but omitting the factor when $i = j$. Up to a ninth-degree polynomial can be employed. The x, y-pairs can be in any order, but the number must be just one more than the degree of the interpolating polynomial. In other words, a polynomial that fits exactly through all N points is used.

Program 2 (Fig. 3.3) employs the Newton–Gregory interpolating polynomial, again of degree up to nine, to interpolate in a function tabulated with uniform x-spacing. The differences are computed as they are needed, and a single array is employed both to compute and to store them, for maximum economy of memory use. The value of x_0 is automatically chosen to put the point of interpolation as near the center of the domain as possible.

```
      SUBROUTINE LAGINT (X, Y, N, XINT, YOUT)
C  THIS SUBROUTINE PERFORMS LAGRANGIAN INTERPOLATION WITHIN A SET
C  OF (X, Y) PAIRS TO GIVE THE Y VALUE CORRESPONDING TO XINT.  THE
C  DEGREE OF THE INTERPOLATING POLYNOMIAL IS ONE LESS THAN THE NUMBER OF
C  POINTS SUPPLIED
C  PARAMETERS ARE -
C    X      ARRAY OF VALUES OF THE INDEPENDENT VARIABLE
C    Y      ARRAY OF FUNCTION VALUES CORRESPONDING TO X
C    N      NUMBER OF POINTS
C    XINT   THE X-VALUE FOR WHICH ESTIMATE OF Y IS DESIRED
C    YOUT   THE Y-VALUE RETURNED TO CALLER
      DIMENSION X(N), Y(N)
      YOUT = 0.0
      DO 20 I = 1,N
        TERM = Y(I)
        DO 10 J = 1,N
          IF (I .EQ. J)  GO TO 10
          TERM = TERM*(XINT - X(J))/(X(I) - X(J))
10      CONTINUE
        YOUT = YOUT + TERM
20    CONTINUE
      RETURN
      END
```

Figure 3.2 Program 1.

```
DATA tst(keep=x y yinterp);
SET row; SET col; set mtrx2;
ARRAY xx{20} row1-row20;  * row values;
ARRAY yy{21} col1-col21;  * column values;
ARRAY nm(&ydim. ) xn1-xn&ydim. ; ARRAY dm(&ydim. ) xd1-xd&ydim. ;
ARRAY mtx(20,21) mm1-mm420;  * known response values in matrix form;

frst_x=&frstx. ; frst_y= &frsty. ;

DO xrow = 0+frst_x to %EVAL(&xdim. -1)+frst_x;  * xdim dimensions in x ;
x=xx{xrow};

y= &yds.;
  DO ii = 1 to &ydim. ;
    nm{ii}=1; dm{ii}=1;
     DO jj = 1 to &ydim. ;
      IF (jj NE ii) THEN
        DO; nm{ii} = nm{ii}*(y                  - yy{jj+(frst_y-1)});
            dm{ii} = dm{ii}*(yy{ii+(frst_y-1)} - yy{jj+(frst_y-1)});
        END;
     END;
  END;

  yprd=0;
  DO ii = 1 TO &ydim. ;
    yprd = yprd + (nm{ii}/dm{ii}) * mtx{xrow,ii+frst_y-1};
  END;
 yinterp = yprd;
 OUTPUT;
END;

proc sort; BY x y;

PROC PRINT data=tst NOobs; VAR x y yinterp; run;
```

```
                16        17        18
               470       480       490
16 3000  -1.089 -1.062 -1.036
17 3500  -1.031 -1.004 -0.978
18 4000  -0.975 -0.949 -0.922
```

```
PROC TRANSPOSE DATA=tst OUT=tr_x(DROP=_name_) prefix=_x; VAR x;
PROC TRANSPOSE DATA=tst OUT=tr_y(drop=_name_) prefix=_y; VAR yinterp;

PROC PRINT DATA=tr_x NOobs; run;
PROC PRINT DATA=tr_y NOobs; RUN;

DATA one; SET tr_x; SET tr_y;
ARRAY xx(&xdim.) _x1-_x&xdim. ;
ARRAY yy(&xdim. ) _y1-_y&xdim. ;
ARRAY xnm(&xdim.) xn1-xn&xdim. ; ARRAY xdm(&xdim. ) xd1-xd&xdim. ;
y= &yds. ;
x= &xds. ;

DO i = 1 to &xdim. ;
  xnm{i}=1; xdm{i}=1;

 DO j = 1 to &xdim. ;
   IF (j ne i) THEN
     DO; xnm{i} = xnm{i}*(x      - xx{j});
         xdm{i} = xdm{i}*(xx{i} - xx{j});
     END;
 END;

END;

estimate=0;
Do ii = 1 TO &xdim. ;
  estimate = estimate + (xnm{ii} / xdm{ii}) * yy {ii};
END;
OUTPUT;
RUN;

PROC PRINT data=one NOobs; var x y estimate; RUN;
```