$f(x, y, z)$. Multidimensional interpolation is often accomplished by a sequence of one-dimensional interpolations. We discuss this in §3.6.

CITED REFERENCES AND FURTHER READING:

Abramowitz, M., and Stegun, I.A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, Volume 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §25.2.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), Chapter 2.

Acton, F.S. 1970, *Numerical Methods That Work*; 1990, corrected edition (Washington: Mathematical Association of America), Chapter 3.

Kahaner, D., Moler, C., and Nash, S. 1989, *Numerical Methods and Software* (Englewood Cliffs, NJ: Prentice Hall), Chapter 4.

Johnson, L.W., and Riess, R.D. 1982, *Numerical Analysis*, 2nd ed. (Reading, MA: Addison-Wesley), Chapter 5.

Ralston, A., and Rabinowitz, P. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), Chapter 3.

Isaacson, E., and Keller, H.B. 1966, *Analysis of Numerical Methods* (New York: Wiley), Chapter 6.

3.1 Polynomial Interpolation and Extrapolation

Through any two points there is a unique line. Through any three points, a unique quadratic. Et cetera. The interpolating polynomial of degree $N-1$ through the N points $y_1 = f(x_1), y_2 = f(x_2), \dots, y_N = f(x_N)$ is given explicitly by Lagrange's classical formula,

$$P(x) = \frac{(x-x_2)(x-x_3)...(x-x_N)}{(x_1-x_2)(x_1-x_3)...(x_1-x_N)}y_1 + \frac{(x-x_1)(x-x_3)...(x-x_N)}{(x_2-x_1)(x_2-x_3)...(x_2-x_N)}y_2 + \cdots + \frac{(x-x_1)(x-x_2)...(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)...(x_N-x_{N-1})}y_N \tag{3.1.1}$$

There are N terms, each a polynomial of degree $N-1$ and each constructed to be zero at all of the x_i except one, at which it is constructed to be y_i.

It is not terribly wrong to implement the Lagrange formula straightforwardly, but it is not terribly right either. The resulting algorithm gives no error estimate, and it is also somewhat awkward to program. A much better algorithm (for constructing the same, unique, interpolating polynomial) is *Neville's algorithm*, closely related to and sometimes confused with *Aitken's algorithm*, the latter now considered obsolete.

Let P_1 be the value at x of the unique polynomial of degree zero (i.e., a constant) passing through the point (x_1, y_1); so $P_1 = y_1$. Likewise define $P_2, P_3, \dots, P_N$. Now let P_{12} be the value at x of the unique polynomial of degree one passing through both (x_1, y_1) and (x_2, y_2). Likewise $P_{23}, P_{34}, \dots, P_{(N-1)N}$. Similarly, for higher-order polynomials, up to $P_{123...N}$, which is the value of the unique interpolating polynomial through all N points, i.e., the desired answer.

The various P's form a "tableau" with "ancestors" on the left leading to a single "descendant" at the extreme right. For example, with $N = 4$,

$$\begin{array}{lllll} x_1: & y_1 = P_1 & & & \\ & & P_{12} & & \\ x_2: & y_2 = P_2 & & P_{123} & \\ & & P_{23} & & P_{1234} \\ x_3: & y_3 = P_3 & & P_{234} & \\ & & P_{34} & & \\ x_4: & y_4 = P_4 & & & \end{array} \tag{3.1.2}$$

Neville's algorithm is a recursive way of filling in the numbers in the tableau a column at a time, from left to right. It is based on the relationship between a "daughter" P and its two "parents,"

$$P_{i(i+1)\dots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\dots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\dots(i+m)}}{x_i - x_{i+m}} \tag{3.1.3}$$

This recurrence works because the two parents already agree at points $x_{i+1} \dots x_{i+m-1}$.

An improvement on the recurrence (3.1.3) is to keep track of the small *differences* between parents and daughters, namely to define (for $m = 1, 2, \dots, N-1$),

$$\begin{aligned} C_{m,i} &\equiv P_{i\dots(i+m)} - P_{i\dots(i+m-1)} \\ D_{m,i} &\equiv P_{i\dots(i+m)} - P_{(i+1)\dots(i+m)}. \end{aligned} \tag{3.1.4}$$

Then one can easily derive from (3.1.3) the relations

$$\begin{aligned} D_{m+1,i} &= \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \\ C_{m+1,i} &= \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \end{aligned} \tag{3.1.5}$$

At each level m, the C's and D's are the corrections that make the interpolation one order higher. The final answer $P_{1\dots N}$ is equal to the sum of *any* y_i plus a set of C's and/or D's that form a path through the family tree to the rightmost daughter.

Here is a routine for polynomial interpolation or extrapolation:

```
SUBROUTINE polint(xa,ya,n,x,y,dy)
INTEGER n,NMAX
REAL dy,x,y,xa(n),ya(n)
PARAMETER (NMAX=10)          Largest anticipated value of n.
   Given arrays xa and ya, each of length n, and given a value x, this routine returns a
   value y, and an error estimate dy. If P(x) is the polynomial of degree N-1 such that
   P(xa_i) = ya_i, i = 1,...,n, then the returned value y = P(x).
INTEGER i,m,ns
REAL den,dif,dift,ho,hp,w,c(NMAX),d(NMAX)
ns=1
dif=abs(x-xa(1))
```

```
do 11 i=1,n                   Here we find the index ns of the closest table entry,
    dift=abs(x-xa(i))
    if (dift.lt.dif) then
        ns=i
        dif=dift
    endif
    c(i)=ya(i)                and initialize the tableau of c's and d's.
    d(i)=ya(i)
enddo 11
y=ya(ns)                      This is the initial approximation to y.
ns=ns-1
do 13 m=1,n-1                 For each column of the tableau,
    do 12 i=1,n-m             we loop over the current c's and d's and update them.
        ho=xa(i)-x
        hp=xa(i+m)-x
        w=c(i+1)-d(i)
        den=ho-hp
        if(den.eq.0.)pause 'failure in polint'
          This error can occur only if two input xa's are (to within roundoff) identical.
        den=w/den
        d(i)=hp*den           Here the c's and d's are updated.
        c(i)=ho*den
    enddo 12
    if (2*ns.lt.n-m)then      After each column in the tableau is completed, we decide
        dy=c(ns+1)               which correction, c or d, we want to add to our accu-
    else                         mulating value of y, i.e., which path to take through
        dy=d(ns)                 the tableau—forking up or down. We do this in such a
        ns=ns-1                  way as to take the most "straight line" route through the
    endif                        tableau to its apex, updating ns accordingly to keep track
    y=y+dy                       of where we are. This route keeps the partial approxima-
enddo 13                         tions centered (insofar as possible) on the target x. The
return                           last dy added is thus the error indication.
END
```

Quite often you will want to call `polint` with the dummy arguments `xa` and `ya` replaced by actual arrays *with offsets*. For example, the construction `call polint(xx(15),yy(15),4,x,y,dy)` performs 4-point interpolation on the tabulated values `xx(15:18)`, `yy(15:18)`. For more on this, see the end of §3.4.

CITED REFERENCES AND FURTHER READING:

Abramowitz, M., and Stegun, I.A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, Volume 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §25.2.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §2.1.

Gear, C.W. 1971, *Numerical Initial Value Problems in Ordinary Differential Equations* (Englewood Cliffs, NJ: Prentice-Hall), §6.1.

3.2 Rational Function Interpolation and Extrapolation

Some functions are not well approximated by polynomials, but *are* well approximated by rational functions, that is quotients of polynomials. We denote by $R_{i(i+1)\dots(i+m)}$ a rational function passing through the $m+1$ points