```
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 $R_{i(i+1)\ldots(i+m)}$ a rational function passing through the $m+1$ points

$(x_i, y_i) \ldots (x_{i+m}, y_{i+m})$. More explicitly, suppose

$$R_{i(i+1)\ldots(i+m)} = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu} \tag{3.2.1}$$

Since there are $\mu + \nu + 1$ unknown p's and q's (q_0 being arbitrary), we must have

$$m + 1 = \mu + \nu + 1 \tag{3.2.2}$$

In specifying a rational function interpolating function, you must give the desired order of both the numerator and the denominator.

Rational functions are sometimes superior to polynomials, roughly speaking, because of their ability to model functions with poles, that is, zeros of the denominator of equation (3.2.1). These poles might occur for real values of x, if the function to be interpolated itself has poles. More often, the function $f(x)$ is finite for all finite *real* x, but has an analytic continuation with poles in the complex x-plane. Such poles can themselves ruin a polynomial approximation, even one restricted to real values of x, just as they can ruin the convergence of an infinite power series in x. If you draw a circle in the complex plane around your m tabulated points, then you should not expect polynomial interpolation to be good unless the nearest pole is rather far outside the circle. A rational function approximation, by contrast, will stay "good" as long as it has enough powers of x in its denominator to account for (cancel) any nearby poles.

For the interpolation problem, a rational function is constructed so as to go through a chosen set of tabulated functional values. However, we should also mention in passing that rational function approximations can be used in analytic work. One sometimes constructs a rational function approximation by the criterion that the rational function of equation (3.2.1) itself have a power series expansion that agrees with the first $m + 1$ terms of the power series expansion of the desired function $f(x)$. This is called $Padé$ *approximation*, and is discussed in §5.12.

Bulirsch and Stoer found an algorithm of the Neville type which performs rational function extrapolation on tabulated data. A tableau like that of equation (3.1.2) is constructed column by column, leading to a result and an error estimate. The Bulirsch-Stoer algorithm produces the so-called *diagonal* rational function, with the degrees of numerator and denominator equal (if m is even) or with the degree of the denominator larger by one (if m is odd, cf. equation 3.2.2 above). For the derivation of the algorithm, refer to [1]. The algorithm is summarized by a recurrence relation exactly analogous to equation (3.1.3) for polynomial approximation:

$$R_{i(i+1)\ldots(i+m)} = R_{(i+1)\ldots(i+m)} + \frac{R_{(i+1)\ldots(i+m)} - R_{i\ldots(i+m-1)}}{\left(\frac{x - x_i}{x - x_{i+m}}\right)\left(1 - \frac{R_{(i+1)\ldots(i+m)} - R_{i\ldots(i+m-1)}}{R_{(i+1)\ldots(i+m)} - R_{(i+1)\ldots(i+m-1)}}\right) - 1} \tag{3.2.3}$$

This recurrence generates the rational functions through $m + 1$ points from the ones through m and (the term $R_{(i+1)\ldots(i+m-1)}$ in equation 3.2.3) $m - 1$ points. It is started with

$$R_i = y_i \tag{3.2.4}$$

and with

$$R \equiv [R_{i(i+1)\ldots(i+m)} \quad \text{with} \quad m = -1] = 0 \tag{3.2.5}$$

Now, exactly as in equations (3.1.4) and (3.1.5) above, we can convert the recurrence (3.2.3) to one involving only the small differences

$$\begin{aligned} C_{m,i} &\equiv R_{i\ldots(i+m)} - R_{i\ldots(i+m-1)} \\ D_{m,i} &\equiv R_{i\ldots(i+m)} - R_{(i+1)\ldots(i+m)} \end{aligned} \tag{3.2.6}$$

Note that these satisfy the relation

$$C_{m+1,i} - D_{m+1,i} = C_{m,i+1} - D_{m,i} \tag{3.2.7}$$

which is useful in proving the recurrences

$$\begin{aligned} D_{m+1,i} &= \frac{C_{m,i+1}(C_{m,i+1} - D_{m,i})}{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i} - C_{m,i+1}} \\ C_{m+1,i} &= \frac{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i}(C_{m,i+1} - D_{m,i})}{\left(\frac{x-x_i}{x-x_{i+m+1}}\right) D_{m,i} - C_{m,i+1}} \end{aligned} \tag{3.2.8}$$

This recurrence is implemented in the following subroutine, whose use is analogous in every way to `polint` in §3.1.

```
SUBROUTINE ratint(xa,ya,n,x,y,dy)
INTEGER n,NMAX
REAL dy,x,y,xa(n),ya(n),TINY
PARAMETER (NMAX=10,TINY=1.e-25)      Largest expected value of n, and a small number.
    Given arrays xa and ya, each of length n, and given a value of x, this routine returns a
    value of y and an accuracy estimate dy. The value returned is that of the diagonal rational
    function, evaluated at x, which passes through the n points (xa_i, ya_i), i = 1...n.
INTEGER i,m,ns
REAL dd,h,hh,t,w,c(NMAX),d(NMAX)
ns=1
hh=abs(x-xa(1))
do 11 i=1,n
    h=abs(x-xa(i))
    if (h.eq.0.)then
        y=ya(i)
        dy=0.0
        return
    else if (h.lt.hh) then
        ns=i
        hh=h
    endif
    c(i)=ya(i)
    d(i)=ya(i)+TINY                  The TINY part is needed to prevent a rare zero-over-
enddo 11                                 zero condition.
y=ya(ns)
ns=ns-1
do 13 m=1,n-1
    do 12 i=1,n-m
```

```
        w=c(i+1)-d(i)
        h=xa(i+m)-x                h will never be zero, since this was tested in the ini-
        t=(xa(i)-x)*d(i)/h             tializing loop.
        dd=t-c(i+1)
        if(dd.eq.0.)pause 'failure in ratint'
          This error condition indicates that the interpolating function has a pole at the re-
          quested value of x.
        dd=w/dd
        d(i)=c(i+1)*dd
        c(i)=t*dd
    enddo 12
    if (2*ns.lt.n-m)then
        dy=c(ns+1)
    else
        dy=d(ns)
        ns=ns-1
    endif
    y=y+dy
enddo 13
return
END
```

CITED REFERENCES AND FURTHER READING:

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §2.2. [1]

Gear, C.W. 1971, *Numerical Initial Value Problems in Ordinary Differential Equations* (Englewood Cliffs, NJ: Prentice-Hall), §6.2.

Cuyt, A., and Wuytack, L. 1987, *Nonlinear Methods in Numerical Analysis* (Amsterdam: North-Holland), Chapter 3.

3.3 Cubic Spline Interpolation

Given a tabulated function $y_i = y(x_i),\ i = 1...N$, focus attention on one particular interval, between x_j and x_{j+1}. Linear interpolation in that interval gives the interpolation formula

$$y = Ay_j + By_{j+1} \tag{3.3.1}$$

where

$$A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} \qquad B \equiv 1 - A = \frac{x - x_j}{x_{j+1} - x_j} \tag{3.3.2}$$

Equations (3.3.1) and (3.3.2) are a special case of the general Lagrange interpolation formula (3.1.1).

Since it is (piecewise) linear, equation (3.3.1) has zero second derivative in the interior of each interval, and an undefined, or infinite, second derivative at the abscissas x_j. The goal of cubic spline interpolation is to get an interpolation formula that is smooth in the first derivative, and continuous in the second derivative, both within an interval and at its boundaries.

Suppose, contrary to fact, that in addition to the tabulated values of y_i, we also have tabulated values for the function's second derivatives, y'', that is, a set