Chapter 4

Interpolation/Extrapolation Techniques

4.1 Introduction

Often experimental results are available for selected conditions, but values are needed for intermediate conditions. For example, the fall speeds of raindrops are measured for specific diameters, but calculation of a rain-rate from a measured drop size distribution requires fall speeds for intermediate diameters. The estimation of such intermediate values is called *interpolation*. Another common use for interpolation is when the functional dependence is so complicated that explicit evaluation is costly in terms of computer time. In such cases, it may be more efficient to evaluate the function at selected points spanning the region of interest and then use interpolation, which can be very efficient, to determine intermediate values.

Extrapolation is the extension of such data beyond the range of the measurements. It is much more difficult, and can result in serious errors if not used and interpreted properly. For example, a high-order polynomial may provide a very good fit to a data set over its range of validity, but if higher powers than needed are included, the polynomial may diverge rapidly from smooth behaviour outside the range of the data.

In this section we will learn about:

- Interpolating within the set of known data (*Newton's* and *Lagrange's algorithms*)
- Extrapolating beyond the set of known data (*Richardson extrapolation* and *Padé approximation*)
- Best fit to and smoothing of data (*least squares* and *cubic splines*)

4.2 Interpolation

4.2.1 Newton interpolation

This is a method to construct a polynomial through all given data points. If there are n+1 (from 0 to n) data values, a unique polynomial P_n of degree n can be constructed that will pass through all the points.

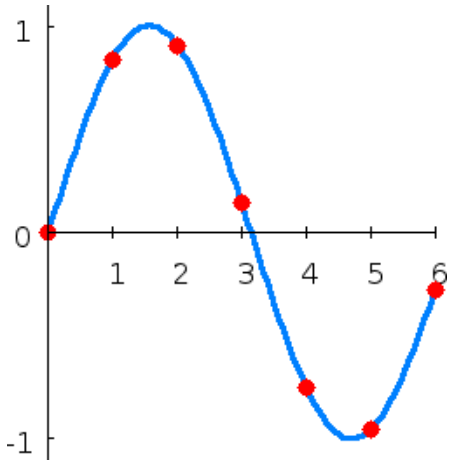

Figure 4.1: The dots denote the data points (x_k, y_k), while the curve shows the interpolation polynomial.

Given a set of n+1 data points

$$(x_0, y_0{=}f(x_0)), \ldots, (x_n, y_n{=}f(x_n)) \,, \tag{4.1}$$

where no two x_i are the same, Newton's method consists in solving the simultaneous equations that result from requiring the polynomials to pass through the data values. It is easy to define the interpolating polynomial if we construct it in the following form

$$P_n(x) = C_0 + C_1(x{-}x_0) + C_2(x{-}x_0)(x{-}x_1) + \ldots + C_n(x{-}x_0)(x{-}x_1)\cdots(x{-}x_{n-1}) \,. \tag{4.2}$$

Now using the interpolation condition $P_n(x_i) = f(x_i)$ we obtain the following triangular system of equations for the coefficients C_i

$$\begin{aligned}
f(x_0) &= C_0 \,, && (4.3)\\
f(x_1) &= C_0 + C_1(x_1{-}x_0) \,, && (4.4)\\
f(x_2) &= C_0 + C_1(x_1{-}x_0) + C_2(x_2{-}x_0)(x_2{-}x_1) \,, && (4.5)\\
&\vdots \\
f(x_n) &= C_0 + C_1(x_n{-}x_0) + C_2(x_n{-}x_0)(x_n{-}x_1) + \ldots \\
&\quad + C_n(x_n{-}x_0)(x_n{-}x_1)\cdots(x_n{-}x_{n-1}) && (4.6)
\end{aligned}$$

From this system we easily find

$$C_0 = f(x_0)\,, \tag{4.7}$$

$$C_1 = \frac{f(x_1)-f(x_0)}{x_1-x_0}\,, \tag{4.8}$$

$$C_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2-x_0}\,, \tag{4.9}$$

$$\vdots$$

The corresponding algorithm, is the so-called Newton's divided differences algorithm and is given in § 4.6.1.

The expressions on the right hand side of the equations above are usually denoted by $f[x_i]$, $f[x_i, x_{i-1}]$, $f[x_i, x_{i-1}, x_{i-2}]$, ... In this notation we can write:

$$\begin{aligned} P_n(x) &= f[x_0] + f[x_1, x_0](x-x_0) + f[x_2, x_1, x_0](x-x_0)(x-x_1) + \\ &\quad + f[x_n, x_{n-1}, \ldots, x_0](x-x_0)(x-x_1)\cdots(x-x_{n-1})\,. \end{aligned} \tag{4.10}$$

One can show that

$$f[x_k, x_{k-1}, \ldots, x_0] = \sum_{i=0}^{k} \frac{f(x_i)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_k)}\,. \tag{4.11}$$

The two previous expressions define the interpolating polynomial in *Newton's form*. This form of the interpolating polynomial is convenient if we have sequential growth of the grid, i.e. we add an $(n+2)$-nd point and increase the degree of the polynomial from n to $(n+1)$. This procedure requires the additional computation of only a single term $f[x_{n+1}, x_n, \ldots, x_0](x-x_0)(x-x_1)\cdots(x-x_n)$ in the expression for $P_n(x)$.

4.2.2 Lagrange interpolation

The Lagrange polynomials provide a convenient alternative to solving the simultaneous equations that result from requiring the polynomials to pass through the data values. Lagrange's method is similar to Newton's, and both yield the same (uniquely defined) interpolating polynomial.

The Lagrange interpolation formula is

$$f(x) = \sum_{i=1}^{N} f(x_i) P_i^L(x)\,,$$

where $f(x_i)$ are the known values of the function and $f(x)$ is the desired value of the function. The Lagrange polynomial P_i^L is the polynomial of order $N-1$ that has the value 1 when $x = x_i$ and 0 for all $x_{j\neq i}$:

$$P_i^L(x) = \frac{\prod_{j\neq i}(x-x_j)}{\prod_{j\neq i}(x_i-x_j)}\,.$$

Table 4.1: Data values for interpolating the tanh function.

$x_0 = -1.5$	$x_1 = -0.75$	$x_2 = 0.0$	$x_3 = 0.75$	$x_4 = 1.5$
$f(x_0) = -14.1014$	$f(x_1) = -0.931596$	$f(x_2) = 0.0$	$f(x_3) = 0.931596$	$f(x_4) = 14.1014$

This is a particularly convenient way to interpolate among tabulated values with polynomials when data points are not evenly spaced.

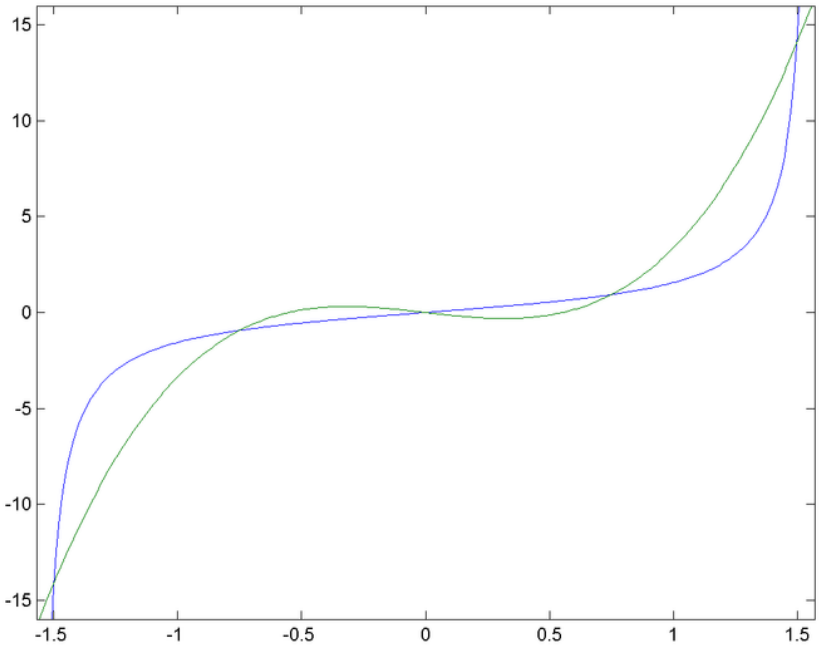

Figure 4.2: The tangent function and its interpolant of order 4 (Lagrange interpolation)

It is usually preferable to search for the nearest value in the table and then use the lowest-order interpolation consistent with the functional form of the data. High-order polynomials that match many entries in the table simultaneously can introduce undesirable rapid fluctuations between tabulated values. *If used for extrapolation with a high-order polynomial, this method may give serious errors.*

Example

We wish to interpolate $f(x) = \tan(x)$ at the points shown in Table 4.1.

The basic Lagrange polynomials are

$$P_0^L(x) = \frac{x-x_1}{x_0-x_1}\cdot\frac{x-x_2}{x_0-x_2}\cdot\frac{x-x_3}{x_0-x_3}\cdot\frac{x-x_4}{x_0-x_4} = \frac{1}{243}x(2x-3)(4x-3)(4x+3)\,, \tag{4.12}$$

$$P_1^L(x) = \frac{x-x_0}{x_1-x_0}\cdot\frac{x-x_2}{x_1-x_2}\cdot\frac{x-x_3}{x_1-x_3}\cdot\frac{x-x_4}{x_1-x_4} = \frac{-8}{243}x(2x-3)(2x+3)(4x-3)\,, \tag{4.13}$$

$$P_2^L(x) = \frac{x-x_0}{x_2-x_0}\cdot\frac{x-x_1}{x_2-x_1}\cdot\frac{x-x_3}{x_2-x_3}\cdot\frac{x-x_4}{x_2-x_4} = \frac{1}{243}(243-540x^2+192x^4)\,, \tag{4.14}$$

$$P_3^L(x) = \frac{x-x_0}{x_3-x_0}\cdot\frac{x-x_1}{x_3-x_1}\cdot\frac{x-x_2}{x_3-x_2}\cdot\frac{x-x_4}{x_3-x_4} = \frac{-8}{243}x(2x-3)(2x+3)(4x+3)\,, \tag{4.15}$$

$$P_4^L(x) = \frac{x-x_0}{x_4-x_0}\cdot\frac{x-x_1}{x_4-x_1}\cdot\frac{x-x_2}{x_4-x_2}\cdot\frac{x-x_3}{x_4-x_3} = \frac{1}{243}x(2x+3)(4x-3)(4x+3)\,. \tag{4.16}$$

Thus the interpolating polynomial is

$$\begin{aligned} f(x) &= \frac{1}{243}\Big[f(x_0)\,x(2x-3)(4x-3)(4x+3) - 8f(x_1)\,x(2x-3)(2x+3)(4x-3) \\ &\qquad + f(x_2)\,(243-540x^2+192x^4) - 8f(x_3)\,x(2x-3)(2x+3)(4x+3) \\ &\qquad + f(x_4)\,x(2x+3)(4x-3)(4x+3)\Big] \qquad (4.17) \\ &= -1.47748x + 4.83456x^3\,. \qquad (4.18) \end{aligned}$$

4.2.3 Runge's phenomenon

When a function f is approximated on an interval $[a,b]$ by means of an interpolating polynomial P, the discrepancy between f and P will (theoretically) be zero at each node of interpolation. A natural expectation is that the function f will be well approximated at intermediate points and that, as the number of nodes increases, this agreement will become better and better. Historically, it was a big surprise that this assumption is not generally true.

Runge's phenomenon is a problem which occurs when using polynomial interpolation with polynomials of high degree. It was discovered by Carl David Tolmé Runge when exploring the behaviour of errors when using polynomial interpolation to approximate certain smooth and simple functions.

A specific example of this remarkable phenomenon is provided by the *Runge function*:

$$f(x) = \frac{1}{1+x^2}\,, \tag{4.19}$$

on the interval $[-5,5]$.

Let P_n be the polynomial that interpolates this function at n+1 equally spaced points on the interval $[-5,5]$, including the endpoints.

Equally spaced nodes

To use the Newton method, we select 11 equally spaced nodes (i.e. we fit a polynomial of degree 10).

In the interval $[a, b]$, the set of n equidistant nodes is given by

$$x_i = a + (i-1)\frac{b-a}{n-1} \qquad 1 \le i \le n \; . \tag{4.20}$$

As shown in Figure 4.3, the resulting curve assumes negative values, which of course, $f(x)$ does not have! More specifically, the resulting interpolation oscillates toward the end of the interval. Adding more equally spaced nodes – and thereby obtaining a higher-degree polynomial – only makes matters worse with wilder oscillations.

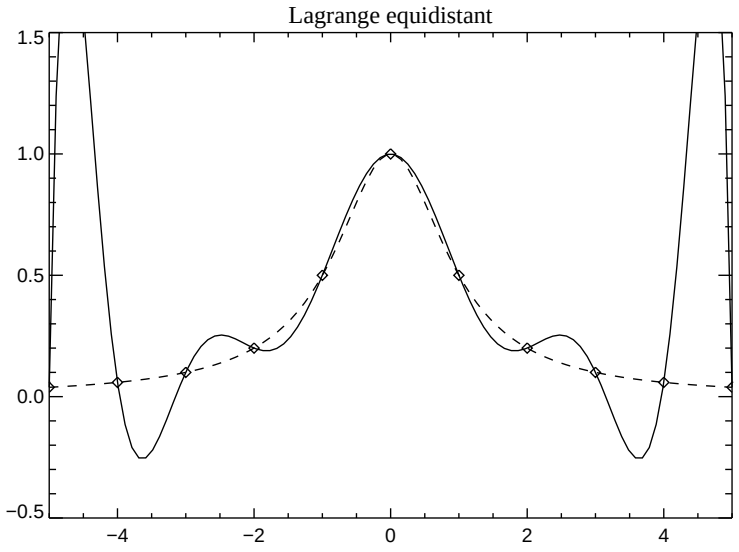

Figure 4.3: Polynomial interpolation of the Runge function (dashed), using equidistant nodes.

Thus, contrary to intuition, equally distributed nodes are a very poor choice in interpolation.

Nevertheless, polynomial interpolation is possible for any reasonable function. This is the essence of Weierstrass's interpolation theorem (stated in 1885):

> "... every continuous function in an interval (a, b) can be represented in that interval to any desired accuracy by a polynomial."

So far, our conclusion from Runge's phenomenon is merely that polynomial interpolation through all given points does often not yield that representation.

Interpolation strategies that work better include using the non-equidistant *Chebyshev nodes* (see below), or *Hermite interpolation*, where the first derivative of the function $f'(x_i)$ is matched as well as the function value $f(x_i)$.

Chebyshev nodes

In the example above, polynomial interpolation at equally spaced points yields a polynomial oscillating above and below the true function. This oscillation is reduced (and convergence of the interpolation polynomials achieved) by choosing interpolation points at Chebyshev nodes.

In the interval $[a, b]$ a set of *Chebyshev nodes* will be defined as:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{i-1}{n-1}\pi\right) \qquad 1 \le i \le n\,. \tag{4.21}$$

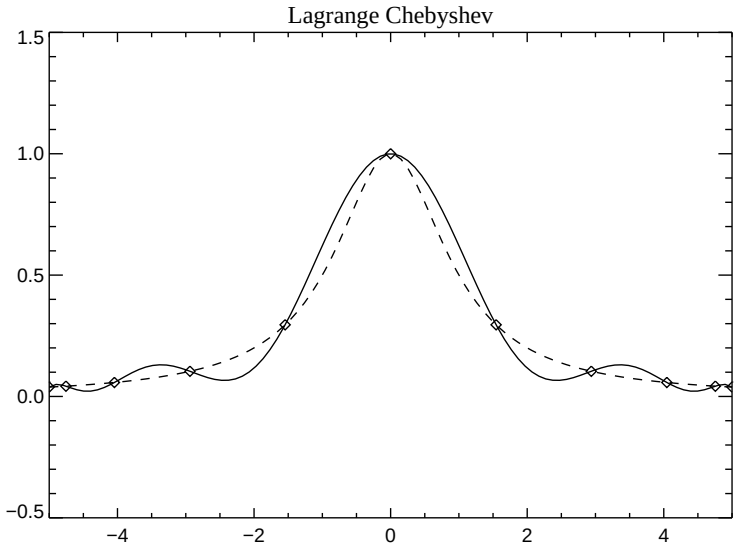

Figure 4.4: Polynomial interpolation of the Runge function using Chebyshev nodes.

Using Chebyshev nodes for interpolating the Runge function, we see (Fig. 4.4) that the oscillations are still there, but they don't grow with increasing number of points, but rather decrease in amplitude. You will find (see the exercises for this chapter) that the resulting polynomial curve needs to fit a 9-th order polynomial before the curve starts to appear significantly smoother.

4.2.4 Cubic spline interpolation

A problem with polynomial interpolation is that higher-order polynomials sometimes produce undesirable fluctuations when the polynomials are forced to fit the data exactly (see the *Runge phenomenon* in the previous section). Spline interpolation provides a technique for obtaining a smoother interpolation and provides a solution to the problem of Runge's phenomenon.

To obtain smoother interpolants, one uses *splines* which are piecewise polynomial functions[1] with continuity requirements imposed on their derivatives at the nodes.

The most common example is the *cubic spline* where $k = 3$. A cubic spline $s_i(x) = P_3(x)$ is constructed for each interval $[x_{i-1}, x_i]$ between data points by determining the four polynomial coefficients, (a, b, c, d).

The two first requirements are that the endpoints of the polynomial match the data:

$$s(x_{i-1}) = f(x_{i-1}) \qquad \text{and} \qquad s(x_i) = f(x_i) \; .$$

The two other constraints arise from the requirement that the first and second derivatives be the same as in adjoining intervals (these constraints are shared with the nearby data intervals, so only provide two constraints). It is conventional to specify that the second derivatives vanish at the endpoints of the data set ("natural splines"). This then specifies a set of simultaneous equations to be solved for the interpolating function. Computer routines are readily available to perform these interpolations.

Consider the case of n+1 nodes, i.e. n intervals between nodes. If we define the spline function on each interval as $S_i(x)$ we can write:

$$S_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3 \; , \tag{4.22}$$

for $x_{i-1} < x < x_i$, combining these functions in such a way that the resulting approximation is continuous and has 2 continuous derivatives (no visible kinks).

The 4 steps/conditions involved in the technique are:

i) $S_i(x_{i-1}) = y_{i-1}$, which yields n equations, for $i = 1$ to n,

$$a_i = y_{i-1} \tag{4.23}$$

ii) $S_i(x_i) = y_i$, which yields n equations, for $i = 1$ to n,

$$a_i + b_i \nabla x_i + c_i \nabla x_i^2 + d_i \nabla x_i^3 = y_i \; , \tag{4.24}$$

where $\nabla x_i \equiv x_i - x_{i-1}$.

iii) $S'_k(x_i) = S'_{i+1}(x_i)$, which yields $n-1$ equations, for $i = 1$ to $n-1$,

$$b_i + 2c_i \nabla x_i + 3d_i \nabla x_i^2 = b_{i+1} \tag{4.25}$$

iv) $S''_k(x_i) = S''_{i+1}(x_i)$, which yields $n-1$ equations, for $i = 1$ to $n-1$,

$$2c_i + 6d_i \nabla x_i = 2c_{i+1} \tag{4.26}$$

[1] In the piecewise interpolation technique we divide the interval $I = [a, b]$ into a set of m subintervals. We then use polynomial interpolation of order, say, k in each of the subintervals, usually with $k \ll m$. For example, piecewise linear interpolation ($k = 1$) is obtained when the interpolation points are connected by straight lines.

Eq. (4.22) states that $S(x)$ consists of piecewise cubic polynomials.

Properties i and ii state that the piecewise cubics interpolate the given set of data points.

Property iii requires that the piecewise cubics represent a smooth continuous function.

Property iv states that the second derivative of the resulting function is also continuous.

Hence, we have $4n$ unknowns (the vectors of coefficients a, b, c and d) and $(4n-2)$ equations. By adding two additional equations, we can uniquely solve for the unknown vectors.

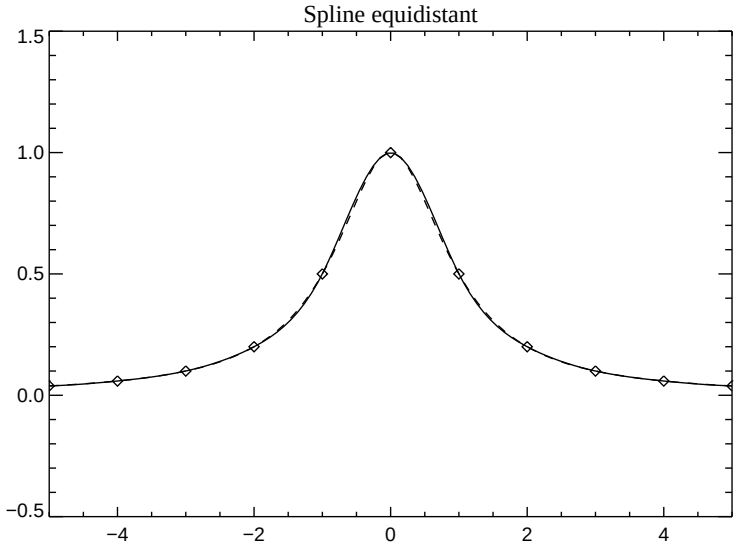

Figure 4.5: Cubic spline fit to the Runge function, using the same points as in Fig. 4.3.

Going back the Runge function we studied in Sec. 4.2.3, let us apply cubic spline interpolation to it. As can clearly be seen in Figure 4.5 even with only 8 nodes the cubic spline does a much better job at interpolating than the direct interpolating polynomial methods.

4.3 Extrapolation

4.3.1 Richardson extrapolation

An important problem that arises in many scientific and engineering applications is that of approximating limits of infinite sequences which in most instances converge very slowly. Thus, to approximate limits with reasonable accuracy, it is necessary to compute a large number of terms, and this is in general costly. These limits can be approximated economically and with high accuracy by applying suitable extrapolation (or convergence acceleration) methods to a small number of terms.

In many problems, such as numerical integration or differentiation, approximate value for some quantity is computed based on some step size. Ideally, we would like to obtain the

limiting value as the step size approaches zero, but we cannot take arbitrarily small step sizes because of excessive cost or round-off error. Based on values for nonzero step sizes, however, we may be able to estimate what value would be for a step size of zero.

Let $A(h)$ denote value obtained with step size h. If we compute the value of A for some nonzero step sizes, and if we know the theoretical behaviour of $A(h)$ as $h \to 0$, then we can extrapolate from known values to obtain an approximation for $A(0)$. This whole procedure can be summarized as follows:

Assume $A(h)$ is an approximation of A with a step size h and the error is $O(h^2)$,

$$A(h) = A + a_2 h^2 + a_4 h^4 + \dots \ . \tag{4.27}$$

– note that $A(h = 0)$ is what we seek. Then we can half the step size and obtain

$$A(h/2) = A + a_2 h^2/4 + \dots \ . \tag{4.28}$$

From these two expressions, we can eliminate the h^2 term and combine $A(h)$ and $A(h/2)$ to obtain a better approximation for A,

$$A = \frac{4A(h/2) - A(h)}{3} + O(h^4) \ . \tag{4.29}$$

The above is often generalized as:

$$A(h) = a_0 + a_1 \, h^p + O(h^r) \tag{4.30}$$

(which implies $p = 2$ and $r = 4$ for the case above) giving us

$$A(0) = a_0 = \frac{2^p A(h/2) - A(h)}{2^p - 1} + a_2' h^r + \dots \tag{4.31}$$

The point here is that we have reduced the error from order h^p in the original approximation to order h^r in the new approximation. If $A(h)$ is known for several values of h, for example h, $h/2$, $h/4$, $h/8$, ..., the extrapolation process can be repeated to obtain still more accurate approximations.

Let us use Richardson extrapolation to improve the accuracy of a finite-difference approximation to derivative of the function $\sin x$ at $x = 1$.

Using the first-order accurate forward difference formula, we have

$$A(h) = a_0 + a_1 h + O(h^2) \ , \tag{4.32}$$

so $p = 1$ and $r = 2$ in this instance.

Using step sizes of $h = 0.5$ and $h/2 = 0.25$, we obtain

$$A(h) = \frac{\sin 1.5 - \sin 1}{0.5} = 0.312048 \ , \tag{4.33}$$

and

$$A(h/2) = \frac{\sin 1.25 - \sin 1}{0.25} = 0.43005 \,. \tag{4.34}$$

The [Richardson] extrapolated value is the given by

$$A(0) = a_0 \approx \frac{2\,A(h/2) - A(h)}{2 - 1} = 0.548061 \,. \tag{4.35}$$

For comparison, the correctly rounded result is given by $\cos 1 = 0.540302$.

Figure 4.6 shows the function $A(h)$ vs h and the steps described above.

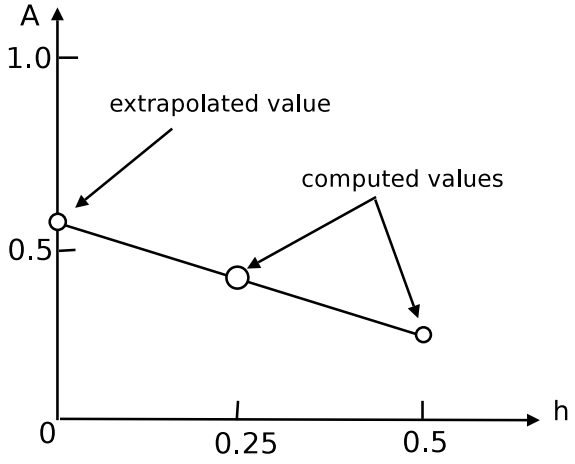

Figure 4.6: Richardson Extrapolation for $\sin x$ at $x = 0.0$

Some remarks

- The extrapolated value, though improved, is still only an approximation, not the exact solution, and its accuracy is still limited by the step size and arithmetic precision used.
- If $F(h)$ is known for several values of h, then extrapolation process can be repeated to produce still more accurate approximations, up to limitations imposed by finite-precision arithmetic.
- Consider the result of a numerical calculation as a function of an adjustable parameter (usually the step size). The function can then be fitted and evaluated at $h = 0$ to yield very accurate results. Press et al. (1992) describe this process as turning lead into gold. Richardson extrapolation is one of the key ideas used in the popular and robust Bulirsch–Stoer algorithm for solving ordinary differential equations.

4.3.2 Romberg integration

Continued Richardson extrapolations using the composite trapezoid rule with successively halved step sizes is called Romberg integration. It is capable of producing very high accuracy (up to the limit imposed by arithmetic precision) for very smooth integrands.

It is often implemented in automatic (though nonadaptive) fashion, with extrapolations continuing until the change in successive values falls below specified error tolerance.

The error in this approximation has the form

$$\int_a^b f(x)dx = T(h) + a_1h^2 + a_2h^4 + a_3h^6 + \dots \ , \tag{4.36}$$

where $T(h)$ is the approximate value for the integral obtained with the trapezoidal rule for step size h. Now suppose we computed a sequence of composite trapezoidal approximations at step sizes $h_0 = h, h_1 = h/2, h_2 = h/4, \dots$. denoting these approximations by $T_{0,0}, T_{1,0}, T_{2,0}, \dots$, we can extrapolate to get the new sequence of approximations

$$T_{k,1} = \frac{4T_{k,0} - T_{k-1,0}}{3} \ . \tag{4.37}$$

Extrapolating again we get

$$T_{k,2} = \frac{16T_{k,1} - T_{k-1,1}}{15} \ . \tag{4.38}$$

The process can be repeated giving

$$T_{k,j} = \frac{4^jT_{k,j-1} - T_{k-1,j-1}}{4^j - 1} \ , \tag{4.39}$$

and forming the array of approximations

$$\begin{array}{ccccccc} T_{0,0} & & & & & & \\ T_{1,0} & T_{1,1} & & & & & \\ T_{2,0} & T_{2,1} & T_{2,2} & & & & \\ T_{3,0} & T_{3,1} & T_{3,2} & T_{3,3} & & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$$

The first column of this array represents the initial trapezoidal approximations, the second column the first extrapolation, etc. Each approximation in the array is computed from the approximations immediately to its left and upper left.

Let us illustrate the *Romberg method* by evaluating

$$\int_0^{\pi/2} \sin x \, dx, \tag{4.40}$$

If we use the composite trapezoid rule, we have

$$A(h) = a_0 + a_1h^2 + O(h^4) \ , \tag{4.41}$$

so $p = 2$ and $r = 4$ in this instance.

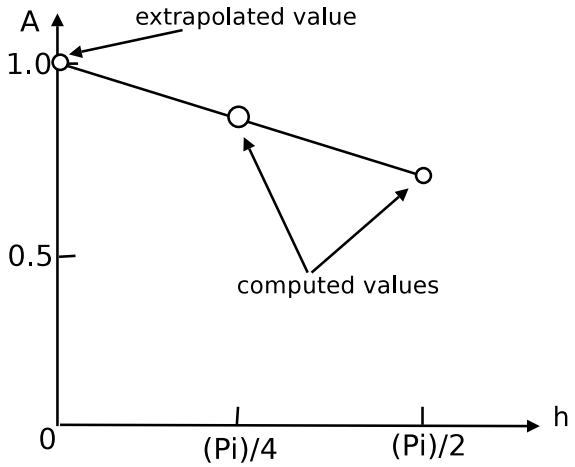

Figure 4.7: Romberg–Richardson Extrapolation (see Sec. 4.3.2)

With $h = \pi/2$, we have

$$A(h) = A(\pi/2) = 0.785398\ , \tag{4.42}$$
$$A(h/2) = A(\pi/4) = 0.948059 \tag{4.43}$$
$$\tag{4.44}$$

The extrapolated value is then given by

$$A(0) = a_0 \approx \frac{4A(h/2) - A(h)}{3} = 1.002280\ , \tag{4.45}$$

which is substantially more accurate than values previously computed (the exact answer is of course 1). See Fig. 4.7 for an illustration.

A Mathematica algorithm for Romberg integration can be found in § 6.3.2.

4.3.3 Padé extrapolation

A Padé approximation is a rational function, viz. a ratio of two polynomials, which agrees to the highest possible order with a known polynomial of order M:

$$f(x) = \sum_{k=0}^{M} c_k x^k \simeq \frac{\sum_{i=0}^{n} a_i x^i}{\sum_{i=0}^{m} b_i x^i}\ . \tag{4.46}$$

One may think of the coefficients c_k as representing a power series expansion of any general function.

In the rational function, one has to set a scale, usually by defining $b_0 = 0$. This leaves $m+n+1$ unknowns, the coefficients a_i and b_i, for which it is unproblematic to solve: the expression is multiplied with the denominator of the rational function, giving on both sides of the

equation polynomials containing the unknown coefficients; one equates all terms with the same power of x to obtain the solution.

Padé approximations are useful for representing unknown functions with possible poles, i.e. with denominators tending towards zero.

Consider a partial Taylor sum

$$T_{N+M}(z) = \sum_{n=0}^{N+M} a_n z^n \,, \tag{4.47}$$

which is a polynomial of degree N+M. Write this in a rational form,

$$P_M^N(z) = \frac{\sum_{n=0}^{N} A_n z^n}{\sum_{m=0}^{M} B_m z^m} \,, \tag{4.48}$$

which is called the $[N, M]$ *Padé approximant*. Here the coefficients are determined from the Taylor series coefficients as follows:

> We set $B_0 = 1$, and determine the N+M+1 coefficients $A_0, A_1, \ldots A_N$ and $B_1, B_2, \ldots, B_M$ by requiring that when the rational function above be expanded in Taylor series about $z = 0$ the first N+M+1 coefficients match those of the original Taylor expansion.

For example, consider the exponential function

$$e^z = 1 + z + \frac{z^2}{2} + \ldots \tag{4.49}$$

The $[1, 1]$ Padé approximant of this is of the form

$$P_1^1(z) = \frac{A_0 + A_1 z}{1 + B_1 z} \,. \tag{4.50}$$

Multiplying the equation

$$\frac{A_0 + A_1 z}{1 + B_1 z} = 1 + z + \frac{z^2}{2} + O(z^3) \tag{4.51}$$

by $(1 + B_1 z)$, we get

$$A_0 + A_1 z + 0\, z^2 = 1 + (B_1{+}1)z + \Big(B_1{+}\frac{1}{2}\Big)z^2 + O(z^3) \,. \tag{4.52}$$

Comparing coefficients, we see that

$$A_0 = 1, \quad B_1 = -\frac{1}{2}, \quad A_1 = \frac{1}{2} \,, \tag{4.53}$$

and thus the $[1, 1]$ Padé is

$$P_1^1(z) = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z} \,. \tag{4.54}$$

Table 4.2: Comparison of partial Taylor series with successive padé approximants for the exponential function, evaluated at $z = 1$. Note that precisely the same data is incorporated in T_{N+M} and in P_M^N.

$T_{N+M}(1)$	$P_M^N(1)$	Relative error of Padé
$T_3(1) = 2.667$	$P_2^1(1) = 2.667$	-1.9%
$T_4(1) = 2.708$	$P_2^2(1) = 2.71429$	-0.15%
$T_5(1) = 2.717$	$P_3^2(1) = 2.71875$	+0.017%
$T_6(1) = 2.71806$	$P_3^3(1) = 2.71831$	+0.00103%
$T_7(1) = 2.71825$	$P_4^3(1) = 2.71827957$	-0.000083%

How good is this? For example at $z = 1$,

$$P_1^1(1) = 3 \,, \tag{4.55}$$

which is 10% larger than the exact answer $e = 2.718281828\ldots$, and is not quite as good as the result obtained from the first 3 terms in the Taylor series,

$$1 + z + \frac{1}{2}z^2\bigg|_{z=1} = 2.5 \,, \tag{4.56}$$

about 8% low.

However in higher orders, Padé approximants rapidly outstrip Taylor approximants as shown in Table 4.2, where we compare the numerical accuracy of P_N^M with T_{N+M}. This comes at a price, the Padé approximation, being a rational expression, has poles, which are not present in the original function. Thus, e^z is an entire function, while the $[1,1]$ Padé approximant of this function has a pole at $z = 2$.

4.4 Data fitting and smoothing

4.4.1 Least-square fitting

The least-square (LSQ) method computes a set of coefficients to the specified function that minimize the square of the difference between the original data and the predicting function. In other words, it minimizes the square of the error between the original data and the values predicted by the approximation.

Suppose that the data points are $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. The fitting curve $f(x)$ has the deviation (error) d from each data point, i.e., $d_1 = y_1 - f(x_1), d_2 = y_2 - f(x_2), \ldots, d_n = y_n - f(x_n)$. According to the method of least squares, the best fitting curve has the property that:

$$S_{lsq} = d_1^2 + d_2^2 + \ldots + d_n^2 = \sum_{i=1}^{n} d_i^2 = \sum_{i=1}^{n} [y_i - f(x_i)]^2 = \text{a minimum} \,. \tag{4.57}$$

Linear least-square method

The simplest function to fit is a linear function $f(x) = ax + b$. By minimizing the square sum of errors, the unknown parameters a and b (the *regression coefficients*) can be determined.

Because the least-squares fitting process minimizes the summed square of the residuals, the coefficients are determined by differentiating S_{lsq} with respect to each parameter, and setting the result equal to zero. That is,

$$\partial S_{lsq}/\partial a = 0 \quad \text{and} \quad \partial S_{lsq}/\partial b = 0 \, . \tag{4.58}$$

The minimization leads to

$$a = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2} \tag{4.59}$$

and

$$b = \frac{1}{n} \left(\sum_{i=1}^{n} y_i - a \sum_{i=1}^{n} x_i \right) . \tag{4.60}$$

As you can see, estimating the coefficients a and b requires only a few simple calculations. Extending this example to a higher-degree polynomial is straightforward although a bit tedious. All that is required is an additional normal equation for each linear term added to the model.

Example

Apply the above to find the equation $y = mx + c$ of the LSQ line that best fits the following data

$$x_1 = 5/2, \; x_2 = 3, \; x_3 = 3/2, \; x_4 = 1 \, , \tag{4.61}$$

$$y_1 = 2, \; y_2 = 9/2, \; y_3 = 3, \; y_4 = 1. \tag{4.62}$$

The answer should be $m = 6/5$ and $c = 9/40$.

Example

As another example, let us apply the LSQ method to the population growth in the United States. Table 4.3 lists the total population of the U.S., as determined by the U.S. census, for the years 1900 to 2000.

Figure 4.8 shows the simple LSQ fit which as you might guess is probably off from the exact answer (the best estimate for the 2010 population is indicated by a cross). We will see that *cubic spline smoothing* does a much better job.

Table 4.3: Total population of the United States in millions of people.

year	millions
1900	75.995
1910	91.972
1920	105.711
1930	123.203
1940	131.669
1950	150.697
1960	179.323
1970	203.212
1980	226.505
1990	249.633
2000	281.422

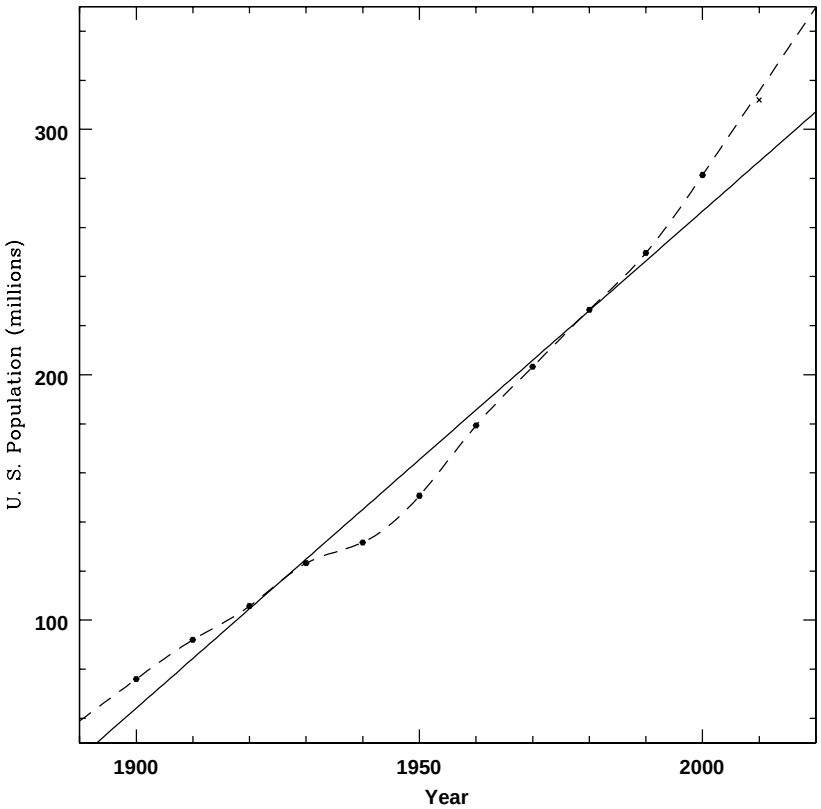

Figure 4.8: Fit to the U.S. population data. *Solid line*: LSQ fit. *Dashed line*: cubic spline interpolation (not a fit). The cross indicates an estimate for the 2010 population.

Non-linear least-square fitting

The linear least-squares method does not only work for fitting a linear functions $a_0 + a_1 x$, but rather for any function that is linear in the coefficients $a_0, a_1, \ldots$. Thus, polynomials of arbitrary orders $a_0 + a_1 c + a_2 x^2 + \ldots$ can easily be fitted – all that's involved is solving a linear system of equations.

Non-linear least-square fitting, on the other hand, is about fitting functions that are non-linear in $a_0, a_1, \ldots$, like e.g. $a_0 e^{a_1 x}$. In this case, a non-linear function needs to be minimized, and there are no general (fool-proof) methods to find a global (absolute) minimum of an arbitrary nonlinear function.

We will not go into any details of this method in the present course.

4.4.2 Cubic spline smoothing

In cubic spline smoothing we relax the requirement that the spline pass exactly through the points and demand only that the spline and its first and second derivatives be continuous.

So continuity still leaves n+3 degrees of freedom. They are determined by balancing two opposing criteria:

- The spline must come reasonably close to the data.
- The spline must have low curvature.

We use a chi-square (χ^2) measure to quantify how close the spline $S(x_i)$ comes to the data:

$$\chi^2 = \sum_{i=0}^{n} \frac{[S(x_i) - y_i]^2}{\sigma_i^2} \,. \tag{4.63}$$

The numerator of each term measures the discrepancy between the spline and the data point, $S(x_i) - y_i$. The denominator provides a weight. The higher the uncertainty σ_i the farther we are allowed to miss the data point. If the spline is a good representation of the data, then deviations are only caused by statistical fluctuations, and the average value of each term in χ^2 is expected to be about one. For reasonably large number of nodes, n, the value of χ^2 is then about $n \pm \sqrt{n}$.

To quantify the "curvature", we integrate the square of the second derivative, giving

$$\int |S''(x)|^2 dx \tag{4.64}$$

These constraints are contradictory. Notice that to make the "curvature" zero, its smallest possible value, the spline would have to have zero curvature – i.e., a straight line. But that would probably give us a high value of χ^2. On the other hand, we can make χ^2 equal to zero by having the spline interpolate the points exactly, but that would give a high curvature.

Putting these two constraints together, we require that the cubic spline minimize

$$W = \rho\chi^2 + \int |S''(x)|^2 \, dx \, . \tag{4.65}$$

The constant ρ determines the trade-off between the two contradictory criteria. Putting $\rho = 0$ removes the constraint on χ^2 altogether and allows the spline to become a straight line. Putting ρ very large puts high emphasis on minimizing χ^2, forcing the spline to interpolate without regard to curvature.

Minimizing subject to the continuity requirements leads again to a tridiagonal system that is easily solved to give the coefficients of the cubics.

Making ρ too small allows for a high value of χ^2. Making it too large forces an unreasonably small value of χ^2. Usually we have to experiment with the choice. The best value results in χ^2 in the range $n \pm \sqrt{n}$.

We should consider "smoothing" to be a poor substitute for fitting data to a decent model function that has a theoretical basis. When we fit experimental data to a model function, we are actually testing our understanding of nature. When we smooth in order to fit to an arbitrary cubic spline, we are merely parameterizing an observation, but not learning something more fundamental.

Let us apply *cubic spline smoothing* to the population growth in the United States: The task is to model the population growth and predict the population in the year 2010.

The spline is shown in Fig. 4.8 as a solid line (the dashed line shows the normal cubic spline) which gives a value of about 312.691 millions. Given the simplicity of the method, spline smoothing does a remarkably good job.

Fitting a 3rd-order polynomial, gives 312.691379 – was ρ used too large? *(Rachid to check)*

4.5 The χ^2 method

The Ξ^2 test is undoubtedly the most important and most used member of the nonparametric family of statistical tests. The Ξ^2 test is employed to test the difference between an *actual sample and another hypothetical or previously established* distribution such as that which may be expected due to chance or probability. Chi Square can also be used to test differences between two or more actual samples. **Is the difference between the samples caused by chances or by an underlying relationship?**.

Null Hypothesis

```
We will always have a null hypothesis which states
 that the observed distribution is not significantly different
  from the expected distribution
  [and of course use words relevant to that particular problem].
```

The test statistic is[2]

$$\Xi^2 = \sum \frac{(f_{\text{exp.}} - f_{\text{obs.}})^2}{f_{\text{exp.}}} , \tag{4.66}$$

where $f_{\text{exp.}}$ and $f_{\text{obs.}}$ are the **expected** and **observed** frequencies per category. How to find these values and work out the problems will hopefully become clear when working the examples below.

4.5.1 Reduced Ξ^2 statistic

What is the reduced chi-square error (Ξ^2/ν) and why should it be equal to 1.0 for a good fit?

The problem with Ξ^2 as written is that its value depends on the number of points used! Simply duplicating all of your data points will most likely double the value of Ξ^2 !

- The method of least squares is built on the hypothesis that the optimum description of a set of data is one which minimizes the weightedsum of squares of deviations, Δy, between the data, y_i, and the fitting function f.
- The sum of squares of deviations is characterized by the Òestimated variance of the fitÓ, s^2, which is an estimate of the variance of the parent distribution, σ^2.
- The ratio of s^2/σ^2can be estimated by Ξ^2/ν, where $\nu = Np1$, N is the number of observations and p is the number of fitting parameters. Ξ^2/ν is called the **reduced chi-square statistic**.
- If the fitting function accurately predicts the means of the parent distribution, then the estimated variance, s^2, should agree well with the variance of the parent distribution, σ^2, and their ratio should be close to one.
- This explains the origin of the rule of thumb for chi-square fitting that states that a Ògood fitÓ is achieved when the reduced chi-square equals one.

4.6 Appendix

4.6.1 Newton's divided differences algorithm

This algorithm requires as input the number of points n in the table and the tabular values in the x and $f(x)$ arrays. The subroutine then computes the coefficients required in the

[2]You may be wondering why we want the sum of squares?
There are two ways to answer this: the very simple-minded explanation is that, by using the squares, you don't get cancellation between deviations, that would otherwise give you a small deviation in cases where your model is not really good. Another way o answering this comes from the fact that in many cases, you assume your distribution of errors to be gaussian, and then the sqared deviation shows itself in the exponent of the probability distribution function (PDF).

Newton interpolating polynomial, storing them in the array `c`:

Newton divided diffs

```
do j=1,n
    c(j) = f(x(j))
enddo
for k=1,n-1
    for j=n,k+1,-1
        c(j) = (c(j) - c(j-1)) / (x(j) - x(j-k))
    end
end
```

4.6.2 Vandermonde matrix

A Vandermonde matrix of order n is of the form

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix} \tag{4.67}$$

The pattern observed in the columns of the *Vandermonde matrix* will aid in the construction of a function that will automatically create this matrix for us:

Vandermonde module

```
function vandermonde(x0) result(V)
    real, dimension(:), intent(in)   :: x
    real, dimension(size(x),size(x)) :: V
    integer                          :: i, n
    do i = 1, n
        V(:,i) = x**(n-1)
    enddo
endfunction vandermonde
```

For example run the following and check your output:

Vandermonde sample

```
program vandermonde_drv
    use vandermonde_mod
    real, dimension(6)                          :: xdata
    real, dimension(size(xdata),size(xdata)) :: V
    integer                                     :: i, j
```

```
    xdata = (/ 10, 20, 30, 40, 50, 60 /)
    V = vandermonde(xdata)
    do i = 1, size(V,1)
        print "(6f12.0)", (V(i,j),j=1,size(V,2))
    enddo
endprogram vandermonde_drv
```

4.6.3 Lab exercise

Using Chebyshev nodes, perform a Lagrange interpolation on the Runge function ($f(x) = 1/(1+x^2)$ in the interval $[-5, 5]$) with

(i) 6 nodes, and

(ii) 10 nodes.

Compare your results (generate a figure) to what is found when using polynomial interpolation with similar number of nodes (see Fig. 4.3).