From Civil Engng Transactions, I.E. Aust., CE36, 331-337 (1994)

Interpolation and numerical differentiation in civil engineering problems

John D. Fenton
Department of Mechanical Engineering, Monash University
Clayton, Victoria, Australia 3168

Abstract

Polynomials are used as the basic means of interpolation and numerical differentiation in many areas of engineering, largely because of their simplicity. However there are dangers in the use of polynomial methods, and sometimes the most simple and obvious of methods give results which are unreliable or even catastrophically wrong for practical problems. This paper shows that a method previously published in the *Civil Engineering Transactions* is liable to be inaccurate, notably for problems in highway and hydraulic engineering. It is also computationally expensive. Another method is described which is simpler to implement, and which is more robust and powerful. It is then shown that there may be unexpected inaccuracies in such global polynomial interpolation methods, and the use of spline interpolation is recommended, with the adoption of a little-known procedure which eliminates the arbitrariness and potential inaccuracy of present widely-used spline methods.

Introduction

A fundamental problem in engineering and science is that of interpolation, the obtaining of a function which passes through data points, and then possibly differentiating the function. The writing of this paper was stimulated by Nichols (1991), who outlined current practice in the design of vertical road curves, and who suggested the use of Hermite polynomials for interpolation. The implementation suggested in that paper contained features which render it a somewhat dangerous and inefficient tool to use.

In this paper it is shown that the Hermite and similar methods lead to the solution of a matrix equation which might be badly conditioned, especially in some civil engineering applications, and that the results obtained can be very poor indeed. Also, there is no advantage to be had by using the orthogonal Hermite functions in this context.

Next, a much simpler and more robust method for polynomial interpolation is presented which is based on divided differences and Newton's interpolation formula. The method can be implemented conveniently in a spreadsheet or using the short pseudocode routines presented. No matrix solution methods are necessary.

It is shown that such methods which seek to interpolate the data globally over the region of interest can give surprisingly poor results. Even more surprisingly, increasing the level of approximation can give worse results. This can be offset by using *piecewise* polynomial approximation, such as cubic splines. However, most conventional presentations of such splines introduce unnecessary assumptions and errors at the ends. Use of the "not-a-knot" condition is described and it is asserted that it is the method which should be used in practice.

The problem of interpolation

Let $x_0, x_1, \ldots, x_n$ be $n+1$ known values of an independent variable *x*, and let $f_0, \ldots, f_n$ be associated known values such as surface elevation or discharge *etc.*. We want to find a polynomial $p_n(x)$ of degree n, which passes through each of the points: $p_n(x_i) = f_i$, for $i = 0, \ldots, n$. In some cases it is possible that derivative values (slopes) might be specified as well at some of the points.

The form advocated by Nichols (1991) was the use of Hermite polynomials, $H_0(x) = 1$, $H_1(x) = 2x$, $H_2(x) = 4x^2 - 2$, $H_3(x) = 8x^3 - 12x$, *etc.*, and to write the polynomial as

$$p_n(x) = a_0 + a_1 H_1(x) + \ldots + a_n H_n(x), \tag{1}$$

where the $a_0, \ \ldots \ , a_n$ are coefficients to be determined. If the individual Hermite polynomials are written in their polynomial form, this is exactly the same as the more familiar *power form*:

$$p_n(x) = b_0 + b_1 x + \ldots + b_n x^n, \tag{2}$$

where the $b_0, \ \ldots \ , b_n$ are different coefficients. Whereas this form is particularly convenient for differentiation and integration, other forms are usually preferred, the reasons for which will be described below.

One apparently simple and sensible approach is to assume one of the forms of equations (1) as done by Nichols, or (2), and to satisfy this equation at each of the interpolation points, which would involve the solution of $n+1$ equations in $n+1$ unknowns. For high-order interpolation this is an inefficient and time-consuming process. What is worse in practice is the fact that the set of equations obtained can be very badly conditioned, so that th e results can be of poor accuracy, and can be remarkably in error. This is particularly so for civil engineering applications, where the independent variable *x* can take on very large values, such as road chainage or talweg distance along a river. Use of these large values can render the methods which require solution of a number of equations useless for practical purposes.

As an example, consider the problem given by Nichols, of a kerb profile, where the known conditions are: at a chainage of 1., the elevation is 1.00 and the slope is -1.5% , while at a chainage of 10., the elevation is 0.75 and the slope is -1.8% . (In the example matrix presented by Nichols, the last row seems to be incompatible with this, and seems to correspond to a slope of -1.8% at a chainage of 2. instead.) If the example is computed with the data given, the use of Hermite polynomials and equation (1) gives the matrix equation:

$$\begin{bmatrix} 1 & 2 & 2 & -4 \\ 1 & 20 & 398 & 7880 \\ 0 & 2 & 8 & 12 \\ 0 & 2 & 80 & 2388 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.75 \\ -0.015 \\ -0.018 \end{bmatrix} \tag{3}$$

Using computer double-precision arithmetic and a linear equation solution package this gave an accurate solution for the coefficients $a_0, \ldots, a_3$.

If, instead of chainages of 1. and 10., the elevation and slope data had been for chainages of 1001. and 1010., quite plausible in civil engineering applications, particularly in highway and river engineering, the matrix equation is obtained:

$$\begin{bmatrix} 1 & 2002 & 4008002 & 8024011996 \\ 1 & 2020 & 4080398 & 8242395880 \\ 0 & 2 & 8008 & 24048012 \\ 0 & 2 & 8080 & 24482388 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.75 \\ -0.015 \\ -0.018 \end{bmatrix}. \tag{4}$$

Using computer double-precision arithmetic with 14 significant figures this also gave an accurate solution. However, if 6 or 8 digit arithmetic is used, possibly considered reasonable for most practical calculations, the results obtained are grossly in error, as shown in Figure 1. The results would be worse if either larger chainages were used or if higher degree polynomials were used. The reasons for this inaccuracy are familiar to computational specialists. The first reason is that the matrix is, in fact, badly

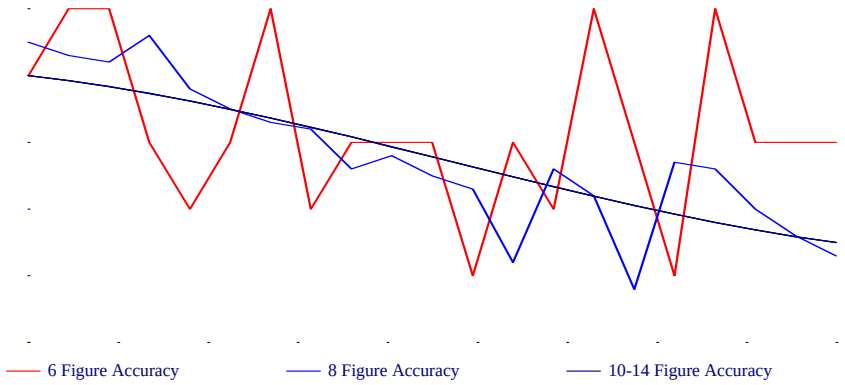

Figure 1. Kerb profile computed using Hermite polynomials with 6, 8 and 10 to 14 figure accuracy

conditioned. If the rows of the matrix in equation (4) are examined, it can be seen that in addition to the matrix elements varying by several orders of magnitude, the first two rows of the matrix and the last two rows are very similar to each other, an indicator of ill-conditioning. The second reason is that, because of the nature of the Hermite functions on the interval between 1001 and 1010, when evaluating the series in equation (1), successive contributions are large numbers oscillating in sign such that round-off error destroys the accuracy of the evaluation.

It is well-known to specialists that the use of orthogonal functions is recommended for problems of interpolation or approximation. The property of orthogonality usually means, in addition to its precise mathematical definition which is not so important her e, that the basis functions are oscillatory on the domain of orthogonality, so that the elements of matrices such as equations (3) or (4) would oscillate in sign such that no two rows would look alike, and the matrix equation can be solved highly accurately. It was this goal which would have stimulated the use of Hermite functions by Nichols. However, the interval over which Hermite polynomials are orthogonal is $[-\infty, +\infty]$. Over the interval of 1 to 10 used in equation (3), the Hermite polynomials do vary somewhat in sign, and the rows of the matrix are not too similar. Over the interval 1001 to 1010 used in the second example, however, the Hermite polynomials are dominated by their highest degree terms, they tend to look like the set of monomials 1, $2x$, $4x^2$, and $8x^3$, and over the relatively limited range they are all positive, they show no oscillatory behaviour, and over that range are certainly not in the slightest orthogonal. Hence, they seem to have no advantages over the simpler monomial formulation, equation (2). It is dangerous to use them or simple monomials for interpolation in the manner suggested above, for they can give catastrophically wrong results, which the above example suggests. In the sort of general problems encountered in interpolation in civil engineering, the fact that the Hermite polynomials are orthogonal over an infinite range plays no role and contributes no advantage.

An easy and practical way of offsetting the above problems would be to use a linear transformation to shift the *x* axis such that 1001 was transformed to -1 and 1010 to +1. On this interval the Chebyshev polynomials are orthogonal and the resulting matrix using them would be very robust. The Hermite polynomials are sufficiently oscillatory on that interval th at they would be almost as good. In fact, even the set of monomials 1, x, x^2 and x^3 oscillate enough in sign on that interval that the resulting matrix would be relatively well-conditioned. However, even if the problems can be overcome in that way, the sort of naive method suggested above leading to equations (3) and (4), always gives a matrix equation, with all the possible problems of ill-conditioning, as well as the fact that the solution is computationally expensive. The number of multiplications/divisions is roughly $N^3/3$, where N is the dimension of the matrix. As well it is necessary to have some means of solving matrix equations, which may not be desirable or possible in practice. Hence, if at all possible such schemes involving the solution of simultaneous equations should be avoided.

Divided differences and Newton's interpolation formula

In this section a simple method is described which suffers from none of the disadvantages described previously. It is computationally cheap, easily programmed, involves no matrix solutions, is subject to no problems of ill-conditioning, and can be used to include easily the specifications of conditions such as gradients or higher derivatives. It is a much better way of performing polynomial interpolation.

The Newton form of the interpolating polynomial, with the coefficients written as $f[\ldots]$, which will be defined further below, is

$$\begin{aligned} p_n(x) &= f[x_0] + (x - x_0)\, f[x_0, x_1] + (x - x_0)(x - x_1)\, f[x_0, x_1, x_2] + \\ &\quad \ldots + (x - x_0)\ (\cdots)\ (x - x_{n-1})\, f[x_0, \ \ldots\ , x_n]. \end{aligned} \tag{5}$$

Note that the *k*th term is a kth degree polynomial. This form seems at first to be rather more complicated than equations (1) and (2). However, it enables the coefficients to be easily obtained and the polynomial to be evaluated easily also.

It can be shown (Conte & de Boor, 1980, p. 42) that the coefficients in this form are simply the *divided differences* of the table of function values (surface elevations in the context of Nichols (1991)):

$$f[x_i] = f(x_i), \text{for all } i,$$

and higher-order divided differences are obtained from the recursion formula

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}, \tag{6}$$

where this quantity is known as the "*k*th divided difference at x_i". For example,

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

and

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

The procedure of obtaining each *k*th divided difference at x_i is simply that of obtaining the difference between the $(k-1)$th divided differences at x_{i+1} and x_i, and dividing by the difference between x_{i+k} and x_i. This is most easily shown by the *divided difference table* shown in Table 1, where the entry at one point has been obtained from the two adjacent entries in the preceding column, and the two values of x_i given by the first and last terms in the square brackets.

x_i	$f[x_i]$	$f[,]$	$f[,,]$	$f[,,,]$
x_0	$f[x_0]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
x_1	$f[x_1]$	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	
x_2	$f[x_2]$	$f[x_2, x_3]$		
x_3	$f[x_3]$			

Table 1. Divided difference table

This procedure is trivially implemented either in the case of a spreadsheet or in a computer program, as given below in pseudocode. Note that the following algorithm returns only the divided differences $f[x_0, ...]$, that is, the top row of the table, as only these are used in the Newton interpolation formula (5).

Algorithm for calculating divided differences:

Remark. The array a_i initially has elements $f(x_i)$, for $i = 0, \ldots, n$. On exit, the a_i are the required values of $f[x_0, \ldots, x_i]$.

1. Do step 3 for $i = 1, \ldots, n$
2. Do step 3 for $j = n,\ n-1,\ \ldots,\ i$
3. $a_j := (a_j - a_{j-1})/(x_j - x_{j-i})$

The computational cost of this is small. The number of divisions is $n\,(n+1)\,/2$, compared with roughly $n^3/3$ multiplications/divisions for matrix methods. If large numbers of points were to be used, this difference would be important. A greater advantage, however, is that all the coefficients can be found using three lines of computer program and no specialised software.

Interpolation using gradient information

In the formulation of the previous section no gradient information was included. The method is easily modified to account for the specification of any derivative at any of the interpolation points as well. Consider the divided difference for the case where two of the x_i are the same. If f is continuously differentiable, then

$$f[x_i, x_i] = \lim_{x_{i+1} \to x_i} \frac{f\,[x_{i+1}] - f\,[x_i]}{x_{i+1} - x_i} = f\,'(x_i) \tag{7}$$

that is, the divided difference for two equal values of *x* is simply the derivative at that point.

Considering higher degrees, it can be shown (Conte and de Boor, 1980, p.64) that if f has k continuous derivatives, then in the case when all of the $x_i, \ \ldots\ , x_{i+k}$ are equal, we have the generalisation of (7):

$$f[x_i, \ \ldots\ , x_{i+k}] = \frac{1}{k!} f^{\,(k)}(x_i) \quad \text{for } \ x_i = \ldots = x_{i+k}. \tag{8}$$

With this expression and equation (6), the formula for the divided difference when none of the arguments are repeated, interpolation of a rather more general nature can be performed. The Newton form (5) for $p_n(x)$ can be used irrespective of whether (6) or (8) is used to obtain the divided differences.

The kerb example used above will be used here to show how the method works. As $x_0 = x_1$ and $x_2 = x_3$ we calculate $f[x_0, x_1] = f\,'(x_0) = -0.015$ and $f[x_2, x_3] = f\,'(x_2) = -0.018$, but with all other divided differences calculated in the usual manner from (6). The divided difference table is shown in Table 2, where the bold face entries are those where gradient information has been inserted and the italic face entries are those where the conventional divided difference has been obtained. Six-figure accuracy was used for the Table as presented. Compared with the solution of a system of simultaneous equations, the computational cost is trivial.

i	x_i	$f[x_i]$	$f[,]$	$f[,,]$	$f[,,,]$
0	1001	1	-0.015	-0.00142	0.000278
1	1001	1	-0.027778	0.001086	
2	1010	0.75	-0.018		
3	1010	0.75			

Table 2. Divided difference table with gradient information

With this, the interpolating polynomial (5) becomes

$$p_n(x) = 1. - 0.015\,(x - 1001) - 0.001420\,(x - 1001)^2 + 0.000278\,(x - 1001)^2(x - 1010)$$

Results are as shown in Figure 2. In comparison with Figure 1 it can be seen that the results even with the 6-figure accuracy as shown in the table are adequate for practical purposes.

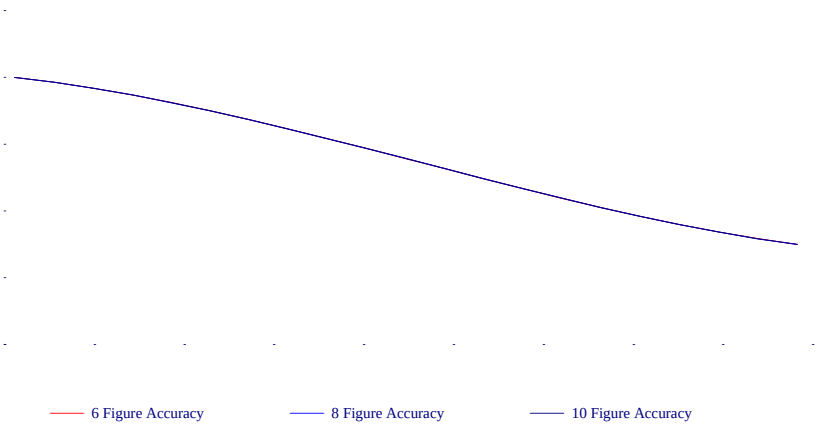

Figure 2. The kerb profile of Figure 1 as computed using Divided Differences and Newton's Interpolating Polynomial

Evaluation of the interpolating polynomial

In addition to the disadvantages described previously, the forms of the interpolating polynomial shown in equations (1) and (2) are inefficient to evaluate. It is more efficient to use *nested multiplication* to evaluate them or the Newton form. To evaluate the latter, for convenience we denote the divided differences $f\,[x_0, \ldots, x_k]$ by a_k for all *k*. Then the Newton form equation (5) can be written in nested form, requiring *n* multiplications:

$$p_n(x) = (\cdots(a_n(x - x_{n-1}) + a_{n-1})(x - x_{n-2}) + \cdots a_2)(x - x_1) + a_1)(x - x_0) + a_0, \qquad (9)$$

giving the simple algorithm:

Algorithm for nested multiplication evaluation of polynomial:

Remark: Given points and associated divided differences $a_0, \ldots, a_n$ we wish to evaluate p_n at some point x_{-1}. We calculate the sequence $b_n, \ldots, b_0$:

1. $b_n := a_n$
2. Do step 3 for $k = n - 1,\ n - 2, \ldots,\ 0$
3. $\qquad b_k := a_k + (x_{-1} - x_k)b_{k+1}$
4. $p_n(x_{-1}) := b_0$

At the end of the calculation, the result is stored in b_0, however the other b_k give us some useful information which we can use to evaluate any derivative of the interpolating polynomial, as in the Section below. The curves in Figure 2 were obtained using this nested form.

Differentiation and conversion to power form

From step 3 of the algorithm for nested multiplication,

$$b_{k+1} = \frac{b_k - a_k}{x_{-1} - x_k}$$

thus, since $b_0 = p_n(x_{-1})$ and as $a_k = f[x_0, \ldots, x_k]$, the b_k can be written as

$$b_k = p_n[x_{-1}, \ldots, x_{k-1}],$$

the divided difference of the interpolatory polynomial p_n for all *k*, based on the points $x_{-1}, \ldots, x_{k-1}$. Hence we can write, by analogy with (9) (or (5) in expanded form)

$$p_n(x) = b_0 + (x - x_{-1})b_1 + (x - x_{-1})(x - x_0)b_2 + \ldots + (x - x_{-1})\,(\cdots)\,(x - x_{n-2})b_n, \qquad (10)$$

and the b_k are the divided differences for the same interpolating polynomial as the a_k, but based on the points $x_{-1}, x_0, \ldots, x_{n-1}$ instead of the original $x_0, \ldots, x_n$.

The algorithm provides a scheme for filling in an additional line on the divided difference table, working from right to left, based on the new point x_{-1}. This provides the most efficient means of giving the ordinary power form by shifting to a new set of centres, and repeated $n - 1$ times. The algorithm is particularly handy for obtaining derivatives of p_n, since $p_n^{(j)}(x)/j!$ is the$(j + 1)$st coefficient in any Newton form for p_n in which the first $j + 1$ interpolation points are *all x*, from equation (8).

In the case where the procedure is repeated several times, to bring all centres to x_{-1} to obtain the polynomial in ordinary power form and/or to obtain the derivatives at that point, we dispense with the symbols b_k, so that in computations the location a_k is overwritten with the value of "b_k" as calculated in step 3 of the nested multiplication algorithm. At the same time the $(x_0, \ldots, x_k)$ are overwritten with the values of $(x_{-1}, \ldots, x_{k-1})$ each time.

Algorithm for conversion to power form and differentiation:

Remark: With known $a_0, \ldots, a_n$, and the point x_{-1}, at which the derivatives may be required or the power form of $p_n(x)$ written:

1. Do step 4 for $j = 1, \ldots, n$
2. Do step 4 for $k = n - 1, n - 2, \ldots, j - 1$
3. $a_k := a_k + (x_{-1} - x_k)a_{k+1}$
4. $x_k := x_{k-1}$

Remark: The *k*th derivative at x_{-1} is now $p_n^{(k)}(x_{-1}) = a_k\, k!$. The nested algorithm above now evaluates the polynomial in the form $p_n(x) = a_0 + a_1\,(x - x_{-1}) + \ldots + a_n\,(x - x_{-1})^n$.

The algorithm may be repeated for any number of different x_{-1}, each time simply using the preceding values of x_k and the a_k.

Performing this algorithm for the above example, with coefficients rounded to six decimal places and performing arithmetic to that same accuracy gave for the calculated gradients at the end points, -0.015000 (exact to this accuracy) and -0.018042 (*cf.* -0.018) , showing that the rounding of the coefficients can lead to inaccuracies in the derivatives. In practice one would work to higher accuracy.

The error of global polynomial approximation

It can be shown (Conte & de Boor, 1980, Section 2.5) that the error of interpolating a function $f(x)$ by an *n*th degree polynomial is

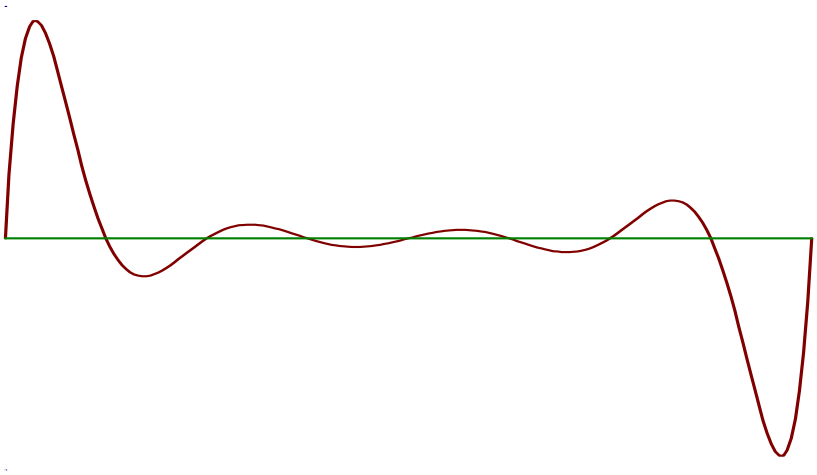

Figure 3. Approximate distribution of interpolation error for equally-spaced values of x_k.

$$f(x) - p_n(x) = \frac{(x - x_0)\dots(x - x_n)}{(n+1)!} f^{(n+1)}(\xi(x)), \tag{11}$$

where the superscript $(n+1)$ indicates the $(n+1)$th derivative, and where $\xi(x)$ is some point, dependent on *x* itself, in the interval of interpolation. We do not know, *a priori*, how that function varies, so in the absence of any other information about the distribution of the error of the interpolation, it varies like the first term in the equation: $(x-x_0)\cdots(x-x_n)$, a polynomial of degree $n+1$. For equispaced points, this function has local maxima increasing towards the ends, increasing with *n*, the degree of the polynomial. For 9 equally-spaced points the function is shown in Figure 3, in which it can be seen that the error will be greatest near the ends. This shows how polynomial interpolation with equally-spaced node points can be inaccurate near the ends of the interpolation interval (and to be avoided totally if outside the interval, as with extrapolation). If it is possible to choose the positions of the interpolation points x_k, they can be placed closer towards the ends and the error can be reduced, especially using points determined by Chebyshev polynomials. In many engineering problems, the points are fixed, that is not possible, and will not be considered here.

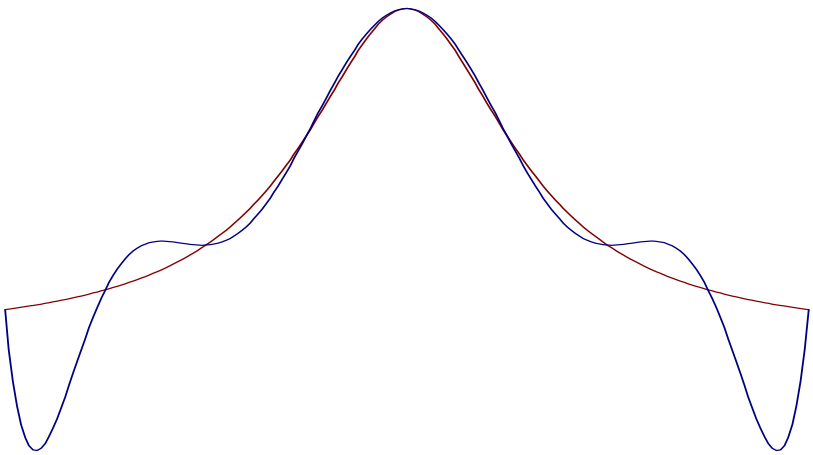

Figure 4. Comparison of interpolation of the function $1/(1+x^2)$ by an 8th degree polynomial over 9 equally-spaced points.

Lest this be thought a piece of undue mathematical pessimism, consider the function $f(x) = 1/(1+x^2)$

over the interval $[-5, +5]$ interpolated by an 8th degree polynomial over nine equally-spaced points shown in Figure 4, which is actually a famous example in numerical analysis (Runge's example) to illustrate the point that *global* polynomial approximation may not very accurate. Once again, a sensible but naive approach would be to include more equally-spaced points and to use a higher-degree interpolating polynomial. The fact that this just makes matters worse is dramatically demonstrated by the results of Figure 5, for a 30-degree polynomial interpolation!

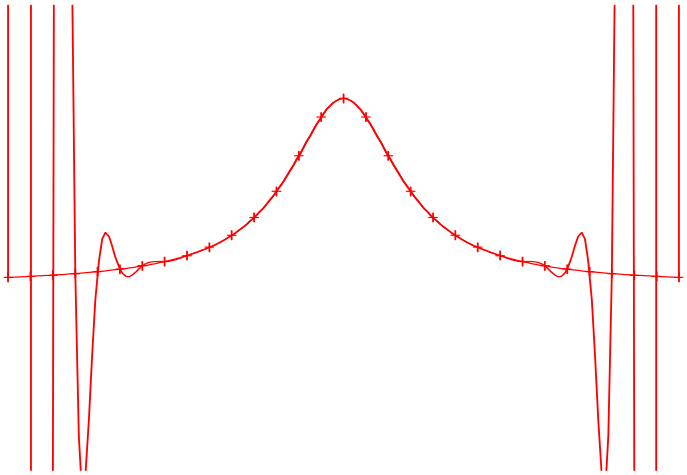

Figure 5. Comparison of interpolation of the function $1/(1 + x^2)$ by a 30th degree polynomial over 31 equally-spaced points.

Piecewise polynomial interpolation - cubic splines

Global polynomial approximation and piecewise-continuous interpolation differ in important respects. The above examples have shown that the former may become increasingly poor as the number of points increases. If the data to be interpolated is rapidly varying somewhere, the polynomial interpolant is apt to be poor over a finite range. There are no such difficulties in piecewise interpolation. Also, as the number of interpolation points *n* increases, the amount of computation for global approximation goes up like n^2 (or n^3 if the naive methods of Section 2 are used), whereas for piecewise interpolation it increases like n.

The most familiar method for piecewise interpolation is to use cubic splines, which is described in many books (for example, Conte and de Boor, 1980, and de Boor, 1978) and included in software packages. In almost all of those, with the exception of de Boor (1978) the method suffers from a major disadvantage, and it is the intention here to describe that and to recommend that de Boor's "not-a-knot" condition be used.

The physical interpretation and the name of cubic splines is familiar to civil engineers, for it comes from a draughtsperson's flexible strip or "spline" which can be used to fair smooth curves between points. If the strip is held in position at various points by pins, then between any two of those pins there are no lateral forces acting so that the shear force in the strip is constant, the bending moment varies at most linearly, and hence by beam theory (for sufficiently small deflections) the strip takes on a cubic variation between the two points. As the variation of moment is different between other points, other cubics will apply there. However, because the shear force and bending moment are continuous across each pin, then the first and second derivatives are continuous across the pins, or interpolation points. With four unknown coefficients for the cubic in between each pair of points, and the requirement that each of the two cubics, to left and right of each interpolation point, must interpolate at that point and must have the

same first and second derivatives, almost enough equations are obtained for all the four coefficients of each cubic. It is necessary, however, to specify two more conditions. This may be by specifying the slope at the two end points, as in Conte and de Boor (1980, Section 6.7), or by arbitrarily specifying that the second derivative at both the end points is zero. This "moment-free" end condition gives the so-called "natural spline" approximation, which is the method usually adopted. In general, however, there is no particular reason at all why the second derivative of the interpolating spline should be zero at the ends.

A superior means of interpolation is to use the "not-a-knot" condition at the first and last *interior* points, where it is required that in addition to the first and second derivatives agreeing to left and right of the interpolation point, that also the third derivatives agree, to give the two extra equations necessary. The physical significance of this is simply that a single cubic interpolates over the first two intervals and another over the last two intervals. No arbitrary assumptions have been introduced, and it can be shown that the error is much less than for natural splines. In fact it can be shown that if the end conditions supplied are exact, the error of spline interpolation is $5/384 \times \left| f^{\,iv}(\xi) \right| (\max_i \ h_i)^4$ where ξ is some value of x, and the symbol $\max_i h_i$ denotes the maximum spacing between interpolation points. If the end conditions provided are the "moment free" conditions when they are not actually the case, the errors near the ends are proportional to $|f\,''(\xi)|\,(h_i)^2$, which is considerably larger. If the "not-a-knot" conditions are used, the errors near the ends have a coefficient larger than 5/384 shown above, but importantly, they are still fourth order. It is a source of amazement to the author that the "not-a-knot" conditions have not been adopted as the standard means of implementing spline interpolation if no gradient information is known.

The manner in which the fluctuations of the interpolating polynomial are held down has made cubic splines popular as a means of interpolating and obtaining derivatives numerically. Whichever additional two equations are supplied, the resulting system of equations is tri-diagonal and diagonally dominant so that Gaussian elimination can be used to solve the equations with a single sweep through to eliminate terms just beneath the diagonal and a single sweep back to eliminate those above the diagonal. FORTRAN programs for all these operations are given in de Boor (1978), and are available as shareware.

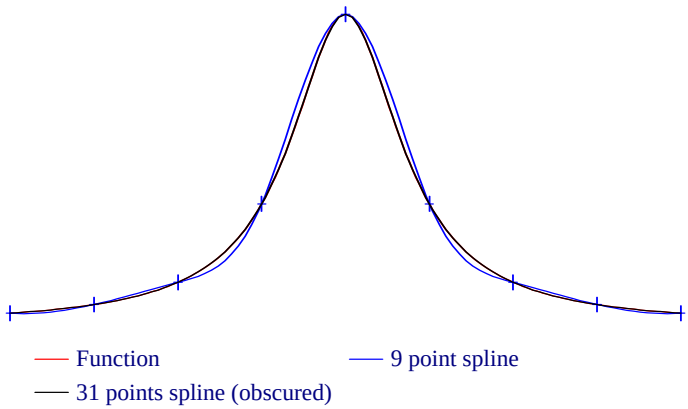

Figure 6. Interpolation of the function $1/(1+x^2)$ by a cubic spline over 9 and 31 equally-spaced points

Figure 6 shows interpolation of Runge's example using cubic splines with 9 and 31 points, the same numbers as for Figures 4 and 5. With 9 points some differences between the function and the interpolant are clear, but very much less than for global polynomial approximation. With 31 points the interpolant everywhere is obscured by the function, even near the sharp crest. The superiority over global polynomial methods is obvious. Nevertheless, the global polynomial methods described above are simple in concept and in application, and for relatively low-degree interpolation might generally find favour.

Conclusions

It has been shown that the method recommended in a previous paper, of simple substitution into the interpolating function at each of the points to be interpolated leads to the solution of a matrix equation, which might be badly conditioned, especially in so me civil engineering applications, and that the results obtained can be very poor indeed. Also, there is no advantage to be had from using orthogonal functions in that context.

A much simpler, more accurate and more robust method using divided differences and Newton's interpolating polynomial has been presented and examined. It can be implemented conveniently in a spreadsheet or using the short pseudocode routines which are presented. The amount of programming is trivial, and the methods can be used to obtain derivatives as well.

It has been shown that the methods considered all aim to interpolate the data globally over the region of interest, but that for some problems such global methods can give very poor results, and that surprisingly, increasing the level of approximation can give worse results. This can be offset by using piecewise polynomial approximation, such as cubic splines. However, most conventional presentations of such splines introduce unnecessary assumptions and errors at the ends. Use of the "not-a-knot" condition is described and it is asserted that it is the method which should be used in all problems unless end derivatives or curvatures are known.

References

Conte, S.D., and de Boor, C. (1980) *Elementary Numerical Analysis* (Third Edn.), McGraw-Hill Kogakusha, Tokyo.

de Boor, C. (1978) *A Practical Guide to Splines*, Springer, Berlin.

Nichols, J.M. (1991) The solution of road gradings using orthogonal Hermite polynomials, *Civil Engineering Trans. I.E. Aust.* **CE33**, 67-71.