Parabola Volume 44, Issue 3 (2008)

Polynomial Interpolation

Bill McKee[1]

Introduction

In an earlier article in *Parabola* (Volume 42, Number 2, 2006), I showed how we could find a straight line which is drawn so as to approximately fit some data points via the process of least-squares fitting. This was then generalised to approximately fit other types of functions to data points.

This article will discuss the related problem of finding a function (in this case a polynomial) which passes **exactly** through some data points. For example, we may have temperature readings taken every hour and wish to estimate the temperature at a point of time somewhere between two of the readings. You may ask why we would choose a polynomial rather than some other function. Well, one reason is that polynomials are simple functions which are easy to understand and to evaluate. The expressions to be discussed here also form the bases for more advanced work in numerical applications of mathematics, such as the approximate evaluation of things like areas and volumes via integration.

Reminder - The Sigma Notation for Sums

If m is less than or equal to n (usually written as $m \leq n$) we use a shorthand notation to indicate sums of quantities as follows:

$$\sum_{i=m}^{n} a_i = a_m + a_{m+1} + \ldots + a_n. \qquad (1)$$

Often $m = 0$ or 1. Notice that we could have written the index of summation in (1) as j or k or l or anything except m or n which designate the first and last terms in the sum. A polynomial of degree n can be written as

$$\sum_{i=0}^{n} b_i x^i = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n.$$

The use of this sigma notation is quite standard and saves writing down long strings of symbols, as does the Pi notation for products which we will now introduce.

The Pi Notation for Products

If m is less than or equal to n we use a shorthand notation to indicate products of quantities as follows:

[1]Dr Bill McKee is a Visiting Fellow in The School of Mathematics and Statistics at the Univeristy of New South Wales.

$$\prod_{i=m}^{n} a_i = a_m \times a_{m+1} \times \ldots \times a_n.$$

Often $m = 0$ or 1.

The Problem of Polynomial Interpolation

Suppose that we have $n+1$ data points (x_i, y_i) for $i = 0, 1, \ldots, n$, where the x_i are all different. The x_i need not be equally spaced, nor need they be in increasing order. There are $n+1$ pieces of information here and we seek a polynomial $p_n(x)$ with $n+1$ coefficients which passes exactly through the $n+1$ data points. Thus we seek the $n+1$ coefficients $b_0, b_1, \ldots, b_n$ such that

$$p_n(x) = \sum_{i=0}^{n} b_i x^i = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^n$$

has the property that $p_n(x_i) = y_i$ for $i = 0, \ldots n$. Thus with $i = 0$ we require

$$b_0 + b_1 x_0 + b_2 x_0^2 + \ldots + b_n x_0^n = y_0$$

with analogous equations at each of the n other points $x_1, x_2, \ldots, x_n$. We are thus led to the problem of solving a system of $n+1$ simultaneous linear equations for the $n+1$ coefficients b_i. This can always be done but it is a messy and time-consuming process which gets even more time-consuming as n increases. There has to be a better way and there is. Read on!

The Lagrange Form

The process begins with the construction of a polynomial of degree n which is zero at all the x_i except one. Let us start with x_0. The polynomial

$$T_0(x) = \prod_{i=1}^{n} (x - x_i) = (x - x_1)(x - x_2) \ldots (x - x_n)$$

has the property that it is of degree exactly n and takes the value 0 at the points $x_1, x_2, \ldots, x_n$ (but not at x_0). If we now construct

$$L_0(x) = \frac{T_0(x)}{T_0(x_0)} = \prod_{i=1}^{n} \frac{(x - x_i)}{(x_0 - x_i)}$$

we have a polynomial of degree n which takes the value 1 at $x = x_0$ and the value 0 at all the other x_i. Hence, multiplying by y_0 we have a polynomial $y_0 L_0(x)$ of degree n which takes the required value y_0 at $x = x_0$ and is 0 at all the other points x_i for $i \neq 0$.

It should now be clear what we have to do. We repeat the above process at each of the x_i to construct $L_i(x)$ and so form

$$p_n(x) = \sum_{i=0}^{n} y_i L_i(x) \qquad (2)$$

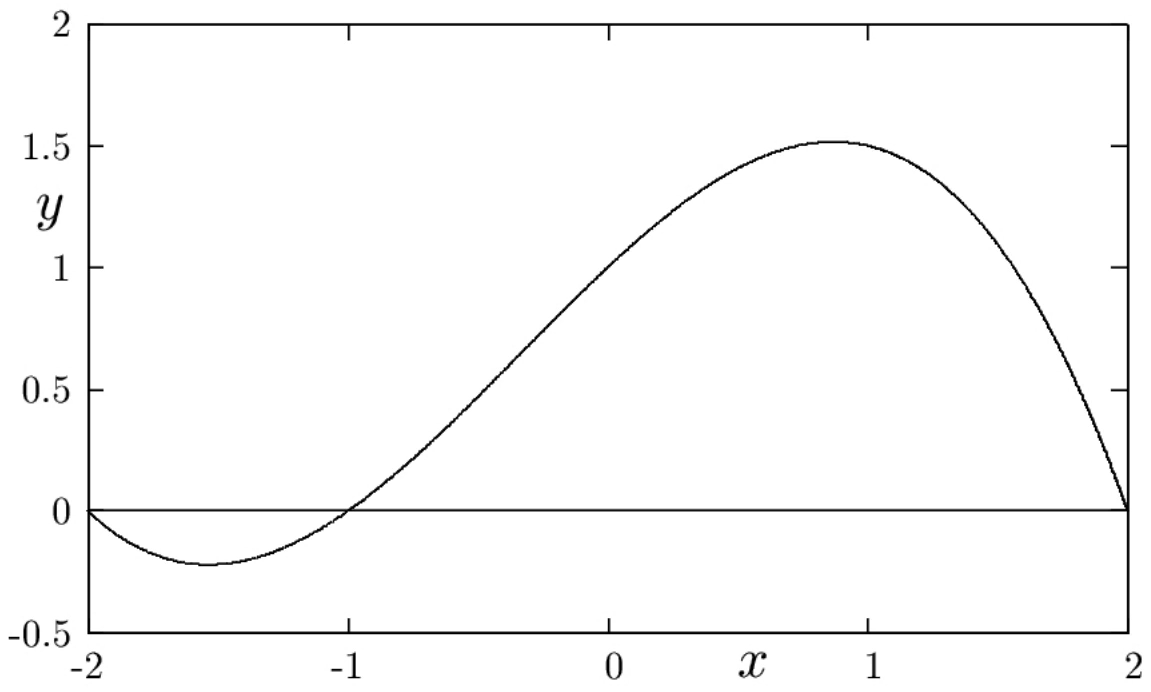

Figure 1: A Lagrange polynomial which takes the value 0 at $x = -2$, $x = -1$, and $x = 2$ and the value 1 at $x = 0$.

where

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

and $j \neq i$ means that we omit the term with $j = i$ in the product. The polynomial $p_n(x)$ has degree n or less and passes exactly through the $n + 1$ data points (x_i, y_i) for $i = 0, 1, \ldots, n$. The reason why the degree could be less than n even though the $L_i(x)$ all have degree exactly n is that cancellations could occur when we construct $p_n(x)$. For example, when $n = 2$ we have three data points and the required polynomial would in general have degree 2, but three points just might happen to lie on a straight line giving a polynomial of degree 1 not 2. Henceforth when we say that a polynomial is of degree n we will really mean that the degree is generally n but could be less than n in some special cases.

The polynomial $L_i(x)$ is sometimes called a *Lagrange Polynomial*. A typical example is shown in Figure 1. The figure (2) is referred to as the *Lagrange form of the interpolating polynomial*.

Now let us find this interpolating polynomial in a particular case. Consider the data shown in Table 1.

i	0	1	2	3
x_i	-2	-1	0	2
y_i	-17	-5	-1	7

Table 1: A typical data set

From these we find

$$\begin{aligned}
L_0(x) &= -\frac{(x+1)\,x\,(x-2)}{8} \\
L_1(x) &= \frac{(x+2)\,x\,(x-2)}{3} \\
L_2(x) &= -\frac{(x+2)(x+1)(x-2)}{4} \\
L_3(x) &= \frac{(x+2)(x+1)\,x}{24} \quad .
\end{aligned}$$

The required interpolating polynomial is thus

$$p_3(x) = -17L_0(x) - 5L_1(x) - L_2(x) + 7L_3(x) = \ldots = x^3 - x^2 + 2x - 1,$$

which is shown in Figure 2.

Thus far, we have shown how to construct a polynomial $p_n(x)$ of degree n which passes exactly through the $n+1$ points (x_i, y_i) for $i = 0, \ldots, n$. An immediate question arises: might not some other procedure for constructing a polynomial with the required properties produce a different result? To investigate this question, suppose that $q_n(x)$ is a different polynomial of degree n which also has the property that $q_n(x_i) = y_i$, for $i = 0, \ldots, n$ and define $r_n(x) = p_n(x) - q_n(x)$. Then $r_n(x)$ is also a polynomial of degree n and has the property that $r_n(x_i) = y_i - y_i = 0$ for $i = 0, \ldots, n$. Since a non-trivial polynomial of degree n can have at most n zeros, it is impossible for $r_n(x)$ to have the $n+1$ zeros x_i for $i = 0, \ldots, n$. Hence, no such polynomial $q_n(x)$ can exist, i.e., $p_n(x)$ is unique. However, there are an infinite number of polynomials of degree greater than n which will pass through the given data points. For example, the polynomial

$$\mathcal{P}(x) = p_n(x) + \prod_{j=0}^{n}(x - x_j)$$

also passes exactly through the data points as does

$$\mathcal{Q}(x) = p_n(x) + g(x)\prod_{j=0}^{n}(x - x_j)$$

where $g(x)$ is *any* polynomial. In this sense, $p_n(x)$ is the 'smallest' polynomial to pass exactly through the given data points. Indeed, if we take $g(x)$ to be any continuous function we can see that there are an infinite number of *functions* which pass exactly through the given data points.

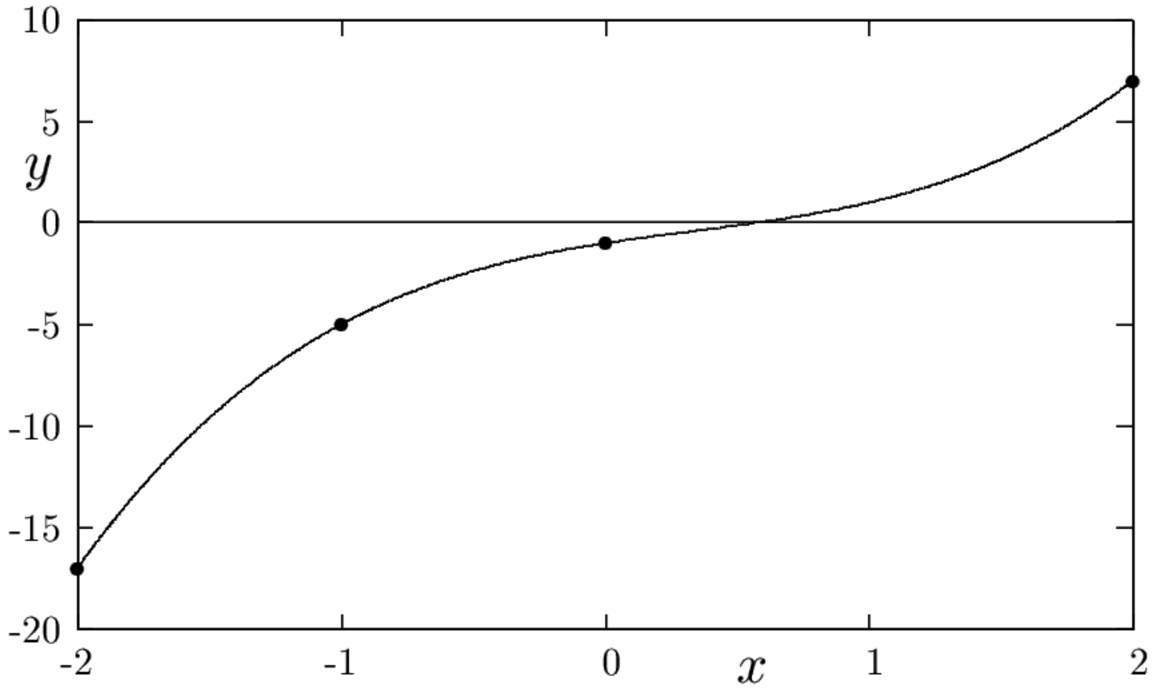

Figure 2: The interpolating polynomial of degree 3 which passes through the points $(-2, -17)$, $(-1, -5)$, $(0, -1)$ and $(2, 7)$

So far, so good. Now let us suppose that we wanted to add another data point (x_{n+1}, y_{n+1}) to our data set. This could happen, for example, if the x_i and y_i were the results of experiments. We would have to start the whole process over again and the previous work would have been in vain. This would be somewhat irksome, so it would make sense to try and find a method of calculating $p_{n+1}(x)$ which made use of $p_n(x)$. This leads us to the topic of *divided differences* and the *Newton form* of the interpolating polynomial.

The Newton Form

We begin by writing $p_n(x)$ as

$$\begin{aligned} p_n(x) &= a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) \qquad (3) \\ &\quad +a_3(x-x_0)(x-x_1)(x-x_2) + \\ &\quad \ldots + a_n(x-x_0)(x-x_1)(x-x_2)\ldots(x-x_{n-1}) \qquad (4) \end{aligned}$$

which is clearly of degree n and contains $n+1$ coefficients. It is called the *Newton form* of the interpolating polynomial. Our task is now to choose these coefficients so that $p_n(x_i) = y_i$ for $i = 0, \ldots, n$. Before discussing this, let us remark that the advantage of writing $p_n(x)$ in the above Newton form is now clear : if we were to add another data point (x_{n+1}, y_{n+1}) the new interpolating polynomial would be

$$p_{n+1}(x) = p_n(x) + a_{n+1}\prod_{i=0}^{n}(x - x_i).$$

The new term is 0 at all the original x_i for $i = 0, \ldots, n$, so $p_{n+1} = p_n$ at each of these points. Hence we only have to find one new coefficient a_{n+1} rather than having to basically start again from the beginning as we had to do when using the Lagrange polynomial approach. However, we still have the problem of working out how to find the a_i.

To make a start on this, let us put $x = x_0$ in (4). Then

$$p_n(x_0) = a_0 = y_0.$$

Next, put $x = x_1$ in (4). Then

$$p_n(x_1) = a_0 + a_1(x_1 - x_0) = y_0 + a_1(x_1 - x_0) = y_1.$$

Hence

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0}.$$

Now put $x = x_2$ in (4). Then

$$p_n(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) = y_2.$$

Using the values of a_0 and a_1 found earlier and performing a few algebraic manipulations, you should be able to show from this that

$$a_2 = \frac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0}.$$

It is now clear from the form of (4) that we can find a_j for $j = 0 \ldots, n$ by putting $x = x_j$ in (4) and using the expressions for the previous coefficients. The expressions become successively more complicated and, at first sight, this process does not seem to be an improvement on the Lagrange polynomial method. However, there is an amazing simplification which occurs via the use of *divided differences* which will now be discussed.

Divided Differences

Given the $n+1$ pairs (x_i, y_i) for $i = 0, 1, \ldots, n$, we define the zeroth divided differences $y[x_i]$ by

$$y[x_i] = y_i \quad \text{for} \quad 0 \le i \le n$$

and the first divided differences $y[x_i, x_{i+1}]$ by

$$y[x_i, x_{i+1}] = \frac{y[x_{i+1}] - y[x_i]}{x_{i+1} - x_i} \quad \text{for} \quad 0 \le i \le n-1.$$

Similarly for $k = 2, \ldots, n$ we define the kth divided differences by

$$y[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{y[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - y[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$$

for $0 \le i \le n-k$. There are $n+1$ zeroth divided differences, n first divided differences, $n-1$ second divided differences and so on, finishing up at just one nth divided difference. We have already found that $a_0 = y[x_0]$, $a_1 = y[x_0, x_1]$ and $a_2 = y[x_0, x_1, x_2]$. It should not surprise you to learn that

$$a_i = y[x_0, x_1, \ldots, x_i] \quad \text{for} \quad 0 \le i \le n$$

, although we will not prove this here.

The process of calculating the divided differences is easy to implement in a computer program via an i loop inside a k loop. For written presentations the process is generally set out via a **divided difference table** as shown in Table 2 for $n = 3$.

x_0	$y[x_0]$			
		$y[x_0, x_1]$		
x_1	$y[x_1]$		$y[x_0, x_1, x_2]$	
		$y[x_1, x_2]$		$y[x_0, x_1, x_2, x_3]$
x_2	$y[x_2]$		$y[x_1, x_2, x_3]$	
		$y[x_2, x_3]$		
x_3	$y[x_3]$			

Table 2: The divided difference table for four data points

The required coefficients a_j are then given by the entries at the top of each column except the first which shows the x values.

-2	-17			
		12		
-1	-5		-4	
		4		1
0	-1		0	
		4		
2	7			

Table 3: The divided difference table for the data in Table 1

Let us now apply this method to the example considered in Table 1. The divided difference table for this example is readily found and shown in Table 3.

Hence the required interpolating polynomial is

$$p_3(x) = -17 + 12(x+2) - 4(x+2)(x+1) + (x+2)(x+1)x$$

which may be expanded out to give $x^3 - x^2 + 2x - 1$, the same polynomial as we found earlier using the Lagrange form. Now suppose we added the extra point $(x_4, y_4) = (1, 13)$. You should verify that $a_4 = y[x_0, x_1, x_2, x_3, x_4] = -2$ and hence that

$$p_4(x) = p_3(x) - 2(x+2)(x+1)x(x-2) = \ldots = -1 + 10x + 7x^2 - x^3 - 2x^4.$$

Discussion and some Applications

Thus far, we have shown how to construct the unique polynomial of degree n which passes exactly through $n+1$ points. Why would we want to do this? One common circumstance would be if the y_i values were the values of some function f, that is, $y_i = f(x_i)$, and we wished to estimate f at some value of x which was not one of the x_i and was less than the smallest of the x_i or larger than the largest of the x_i. This process is generally called *extrapolation*. If the x value lies between the smallest and the largest of the x_i it is called *interpolation*. One often sees considerable use made of interpolation or extrapolation in the media, either using this method or some other. Much of it is highly questionable.

You may find this hard to believe but when I was in high school some five decades ago nobody had calculators. To find values of functions like logarithms and trigonometric functions, we had books of tables. To estimate the values of these functions at points which were not tabulated we interpolated linearly between adjacent tabulated values. Multiplication was performed by adding logarithms (to base 10, not base e) and division by subtraction of logarithms. The introduction of slide rules (analogue devices which added or subtracted numbers marked on a logarithmic scale to perform multiplications or divisions) simplified matters and replaced tables for most purposes, but the introduction of cheap calculators revolutionised things and made life much easier for everybody.

We will now give some applications of polynomial interpolation.

a) Rootfinding

A common circumstance in the practical application of mathematics to areas like science and engineering is to find roots of an equation. You will all be familiar with the formula for finding the roots of a quadratic equation, for example, the roots of a polynomial of degree 2. There is a more complicated expression for the roots of a polynomial of degree 3 and an horrendously complicated expression for the roots of a polynomial of degree 4. It can be shown that there is no general expression for the roots of a general polynomial of degree 5 or greater. For these, numerical methods must be used, as they must, to find the roots of an equation like $\cos x - x = 0$. You may be familiar with *Newton's method* for finding a root of the equation $f(x) = 0$. This starts from some initial estimate x_0 of the root and constructs the tangent to the curve $y = f(x)$ at the point $(x_0, f(x_0))$, and then finds where that tangent line cuts the x-axis. This is taken as the next estimate x_1 of the root. The process is then repeated using x_1 instead of x_0 to give x_2, and so on. Newton's method requires us to be able to find the derivative of $f(x)$. For simple cases like $f(x) = \cos x - x$ this is easy but in cases which arise in practical applications it might be difficult or even impossible. For example, the values of $f(x)$ might be known only as the results of some computer program which outputs $f(x)$ after we have input x. Hence there is a need to devise methods which do not require the evaluation of derivatives. One such method is *inverse interpolation* which will now be outlined.

We begin by choosing two approximations, x_0 and x_1 to a root of the equation $f(x) = 0$. For example, these could be obtained by inspecting the graph of $y = f(x)$. We then calculate $y_0 = f(x_0)$ and $y_1 = f(x_1)$. The trick is to regard x as a function of y and to construct the interpolating polynomial of degree 1 in y which passes through (y_0, x_0) and (y_1, x_1). This is, of course, just a straight line. Putting $y = 0$ in this expression gives our next estimate x_2 of the root. We now construct the interpolating polynomial of degree 2 through (y_0, x_0), (y_1, x_1) and (y_2, x_2) and set $y = 0$ in that expression to give us our next estimate x_3 of the root, and so on. At each stage, we are adding one more data point and the Newton form of the interpolating polynomial using divided differences is ideal for this purpose.

To illustrate this process, let us consider the equation

$$\cos x = x.$$

If we draw the graphs of $y = \cos x$ and $y = x$ we can see that there is only one root located at approximately $x = 0.75$, remembering that x is in radians, not degrees. So let us define $f(x) = \cos x - x$ and start with $x_0 = 0.7$ and $x_1 = 0.8$. The process of inverse interpolation described above then gives rise to Table 4. All the numerical results presented in this article were obtained using software which uses about 15 decimal digits in its calculations. To save space, only the first 7 significant decimal digits are shown in the results.

The notation used in Table 4 is that $0.4520267E - 13$ means $0.4520267 \times 10^{-13}$. Since we are looking for the point where f is exactly zero, it is clear that the method is converging quite nicely to the true root. As with all such rootfinding methods, care is needed in the choice of the starting values or the method may not converge at all or converge to the wrong root if the equation has more than one root. In particular, if the

n	x_n	$f(x_n)$
0	$0.7000000E+00$	$+0.6484219E-01$
1	$0.8000000E+00$	$-0.1032933E+00$
2	$0.7385654E+00$	$+0.8696646E-03$
3	$0.7390853E+00$	$-0.3376796E-06$
4	$0.7390851E+00$	$-0.4520267E-13$

Table 4: The application of inverse interpolation to find the solution to the equation $f(x) = 0$ where $f(x) = \cos x - x$

function f is continuous and monotone increasing or monotone decreasing at and near a root, the method will converge to this root provided the two initial estimates x_0 and x_1 are sufficiently close to the root.

b) Numerical Integration

Another common problem which arises in practice is the calculation of definite integrals of the form

$$I = \int_a^b f(x)\ dx.$$

For simple functions f we may be able to find the primitive function (also called the indefinite integral) F such that $F' = f$ and so find $I = F(b) - F(a)$. In other cases, even seemingly innocuous ones like $x^{-1} \sin x$, no such primitive is available and we must use approximate methods. One approach is to evaluate f at several points between a and b, construct the interpolating polynomial which passes through those points and then integrate that polynomial. We can think of this as using the polynomial as an approximation to the original function on the interval $[a, b]$.

One of the simplest such methods is the *trapezoidal rule* which approximates f on $[a, b]$ by a straight line drawn from $(a, f(a))$ to $(b, f(b))$. You should be able to show that integrating this, or just simply adding the areas of a rectangle and a triangle, gives the approximation $I \approx T$ where

$$T = \tfrac{1}{2}\,(b-a)\,[f(a) + f(b)]\,.$$

It should not surprise you to learn that this formula is exact if f is a polynomial of degree 0 or 1, that is a straight line, but not in general exact for any other function. The trapezoidal rule is commonly implemented in the form of the *composite trapezoidal rule* which divides the interval $[a, b]$ up into N sub-intervals of equal length, applies the trapezoidal rule to each of these sub-intervals and adds up all the results. Thus, we define $x_i = a + ih$ for $i = 0, \ldots, N$, where $h = (b-a)/N$. If $f_i = f(x_i)$, this gives the approximation $I \approx T_N$ where

$$T_N = \tfrac{1}{2}h\,[\,(f_0 + f_1) + (f_1 + f_2) + \ldots + (f_{N-2} + f_{N-1}) + (f_{N-1} + f_N)\,]\,.$$

The interior points double up and we readily see that

$$T_N = h \sum_{i=1}^{N-1} f_i + \tfrac{1}{2} h \, [\, f_0 + f_N \,] \, .$$

The next step up is the well-known *Simpson's rule* which approximates f on $[a, b]$ by the interpolating parabola though $(a, f(a))$, $(c, f(c))$ and $(b, f(b))$ where $c = (a+b)/2$ is the mid-point of the interval. Integrating this parabola gives the approximation $I \approx S$ where

$$S = \tfrac{1}{6} \, (b - a) \, [f(a) + 4f(c) + f(b)] \, .$$

Simpson's rule is exact for all polynomials of degree 2 or less. Somewhat surprisingly, it is also exact for all polynomials of degree 3. As with the trapezoidal rule, Simpson's rule is usually implemented as the composite Simpson's rule. This divides $[a, b]$ into an **even** number $N = 2M$ of subintervals at points $x_i = a + ih$ for $i = 0, \ldots 2M$ where $h = (b - a)/2M$. Then applying Simpson's rule to each group of two intervals (x_{2j-2}, x_{2j-1}) and (x_{2j-1}, x_{2j}) for $j = 0, \ldots M$ to give $I \approx S_{2M}$ where

$$S_{2M} = \tfrac{1}{3} \, h \sum_{j=1}^{M} [\, f_{2j-2} + 4f_{2j-1} + f_{2j} \,] \, .$$

Again, there is some doubling up and we readily find

$$S_{2M} = \tfrac{1}{3} \, h \left\{ f_0 + 2 \sum_{j=1}^{M-1} f_{2j} + 4 \sum_{j=1}^{M} f_{2j-1} + f_{2M} \right\} \quad .$$

There are a great number of such integration rules based upon alternative numbers and placement of the points. For example, we need not insist that the endpoints be included, nor that the points be equally spaced. Since this is an article on interpolation not numerical integration, we will not pursue this matter any further except to conclude with an example.

It is always a good idea to test methods or programs on things for which we already know the correct answer in order to fix simple errors or bugs in our mathematics or programming. We can then have some confidence in applying these techniques to problems for which we do not know the answers. Thus we will apply the trapezoidal rule to

$$I = \int_0^1 \frac{dx}{1 + x^2} = \pi/4 = 0.785398163397448 \ldots \tag{5}$$

Table 5 shows the error E_N (defined as approximate value - exact value) for this integral for various values of N together with the error multiplied by N^2 when the composite trapezoidal rule is applied to (5).

This table strongly indicates that the error decreases as N increases and that the error is approximately proportional to $1/N^2$, that is, to h^2. This is no accident. It can be shown that for a wide class of well-behaved functions (specifically those for which

N	E_N	N^2E_N
2	-0.103981634E-01	-0.415926536E-01
4	-0.260404575E-02	-0.416647320E-01
6	-0.115739678E-02	-0.416662841E-01
8	-0.651039775E-03	-0.416665456E-01
10	-0.416666171E-03	-0.416666171E-01

Table 5: The error for various values of N when the composite trapezoidal rule is applied to the integral given by equation (5).

f and its first two derivatives are continuous for $a \leq x \leq b$) the error in the composite trapezoidal rule is approximately proportional to $1/N^2$. For the composite Simpson's rule the error is approximately proportional to $1/N^4$ provided f and its first four derivatives are continuous on the interval of integration. The composite Simpson's rule consequently gives much better accuracy for a comparable amount of work than does the composite trapezoidal rule. For example, using the composite Simpson's rule on the same integral with $N = 2M = 6$ gives an error of roughly -2×10^{-5} compared with approximately -1×10^{-3} for the composite trapezoidal rule.

Concluding Remarks

In this article, we have shown how to construct the unique polynomial of degree n which passes through $n+1$ data points and discussed two of the myriad of applications of this procedure. Now, a polynomial of degree n may have up to $n-1$ maxima and minima and thus be quite 'wiggly'. One consequence of this when we use polynomial interpolation to estimate the values of a function at intermediate points is that the polynomial may exhibit large fluctuations and hence incur large errors away from the tabulated points. I intend to discuss ways of dealing with this difficulty in a later article.