LECTURE NOTES
INTRODUCTION TO GEOMETRIC MODELING AND COMPUTATION

Justin E. Kerwin

March 26, 2007

1 INTRODUCTION

The field of geometric modelling is concerned with the mathematical representation of curves and surfaces necessary to define complex engineering "objects". These objects can range from machine parts built up from fairly simple shapes to complex "sculptured" surfaces found on ships, automobiles and aircraft. In the latter three examples, outer surfaces are sculptured in order to produce a streamlined, low-drag shape, but they are also complex since other design constraints must be accommodated.

Ocean engineers must deal with complex shape definition for many applications such as ships, underwater vehicles, offshore platforms, and underwater instruments. The shapes of such objects are almost never known in advance, but must evolve in an iterative way as the design process proceeds. This is illustrated in the simplified block diagram in Figure 1.

Note that it is generally necessary to create a design of an object before it can be analyzed [1]. If the analysis indicates that the design is deficient—maybe it is not strong enough, or it will sink—, the design must be changed. This generally means that shapes must be altered numerous times before a satisfactory design is obtained. In many cases, complex calculations which depend on detailed knowledge of the current shape must be carried out. Thus, the ease and accuracy of the procedure to create and modify shapes is of major importance. But in addition, once the final shape has been established, it must be communicated in some way to the manufacturing process. This sounds simple and obvious, but in the real world, problems caused by *miscommunication* of the intended shape can be very serious.

[1] This is not always the case. For example, one can determine the shape of a hydrofoil section, given it's pressure distribution.

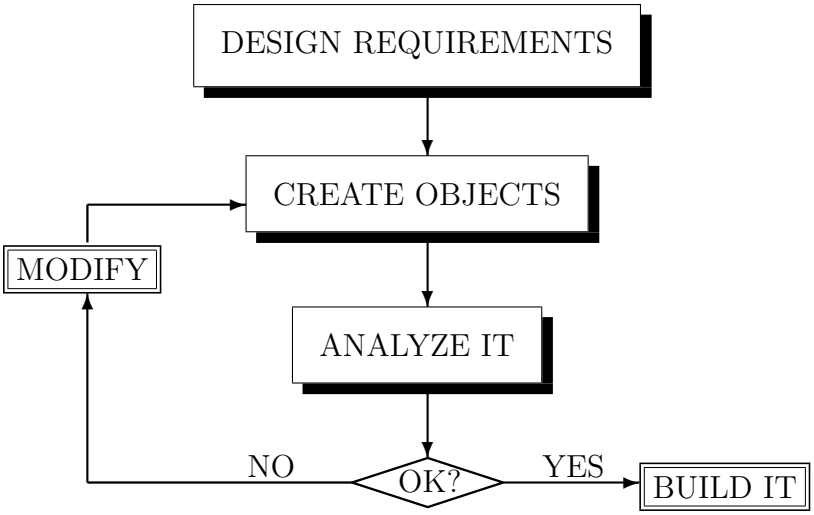


Figure 1: Simplistic flow diagram of an engineering design process.

The mathematical representation of curves and surfaces, incorporated into a computer aided design and manufacturing system with high-resolution graphics capabilities is therefore an essential ingredient in current engineering design. But this is only a fairly recent innovation— one which is evolving rapidly at the present time.

So what did engineers do in the past? A good example is the ship hull, which is of direct interest to ocean engineers, is generally formed from complex surfaces, and which has certainly been around for centuries before the invention of the computer!

1.1 How Ship Lines Were Developed in the Olden Days

Ship lines were generally developed on paper by constructing a 3-view drawing of the intersection of a sequence of orthogonal planes with the hull surface. Such a drawing is shown in Figure 2. The intersections of the hull with a sequence of horizontal planes are called **waterlines**, and one of these is frequently designated the "design waterline". This does not mean that the ship will always float exactly on this waterline, but it generally represents a condition under which the ship is to have a specified carrying capacity and performance.

The intersections of the hull with a set of vertical planes are called **buttock lines**, and the one vertical plane which forms the vertical plane of symmetry of the hull generates what is called the **profile**. The third set of planes, which are orthogonal to the first two, produce the **sections**.

Waterlines and buttocks are generally spaced in even increments of the dimensional units used for the drawing, while the stations are generally spaced in such a way that the length of the vessel is divided into either ten or twenty equal intervals.

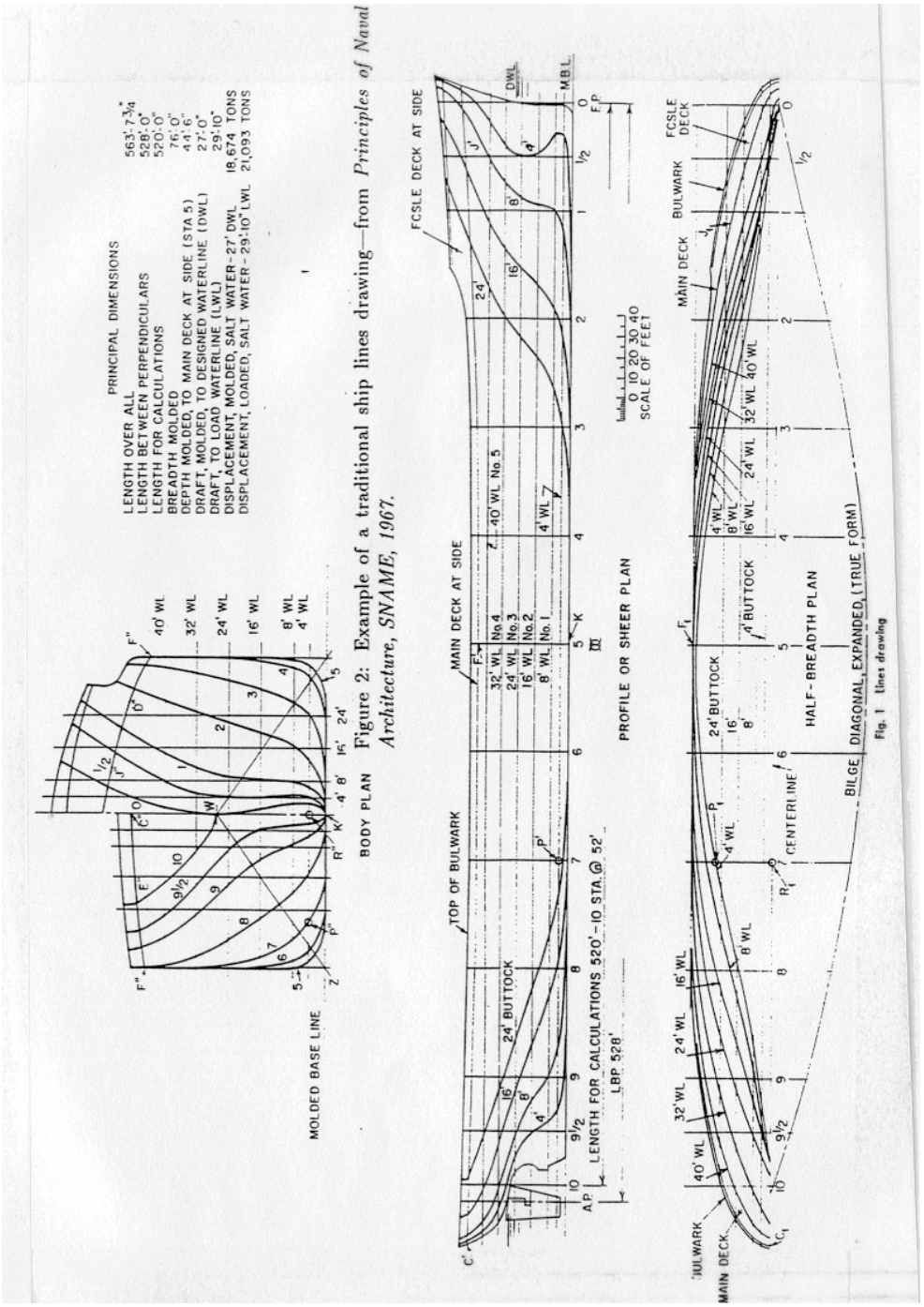


Figure 2: Example of a traditional ship lines drawing—from *Principles of Naval Architecture, SNAME, 1967.*

The length which is divided up in this way is generally not the overall length of the vessel, but the length "between perpendiculars". The forward perpendicular (FP) is generally the intersection of the profile with the design waterline, but the aft perpendicular (AP) is generally the centerline of the rudder post. Finally, the point midway between the perpendiculars is called **amidships** and is given the symbol $\otimes$. This was probably a logical choice for old style ships whose bow profile was vertical, and whose rudder post was at the aft end of the design waterline.

Drawing a good set of lines took considerable skill, since the waterlines, buttocks and sections all had to look smooth, but they also had to be consistent. If the designer wanted to get rid of a bad bump in one of the waterlines, the buttocks and sections would have to be changed to match it. This might then introduce a new bump in one or both of the other views.

It is interesting to note that this problem was avoided in an even earlier period of history where hull lines were developed by carving a wooden model. The designer did not use a solid block, but connected a series of planks with thickness corresponding to the waterline spacing. After the hull was carved, the individual planks were taken apart and traced to make a lines drawing. The three dimensional process of carving the model was, in a way, much closer to current computer graphics methods than was the generation of a three view drawing.

A variation on the model carved and taking apart method was developed by the famous yacht designer Nathaniel Herreshoff in the 1890's . He also carved a wood model, but invented a mechanism with linkages and dials to measure a large number of coordinates on the model surface. These were recorded in a little black book (a database) which was then used directly to lay out the full sized hull. No lines drawings were made.

In earlier years relatively little analysis was made of a design. The principal calculations (if any) were the hydrostatic properties of the hull which could be derived from the areas and centroids of the sections. These could be found by carefully tracing around the individual sections using mechanical devices called planimeters and integrators, or by reading coordinates from the drawing with a ruler and applying numerical integration formulas (which we will cover later in the term).

The manufacture of the hull required the development of full size patterns for the structural components. The measurements made from the lines drawing could, of course, be in full scale units, but this data would not be accurate enough to be used directly for construction. Therefore, a process called "lofting" was required, in which the lines were effectively re-drawn full size, and the reading innacuracies of the small scale drawing removed. Figure 3 is a photograph showing the lines of an ocean cruising sailboat being lofted.

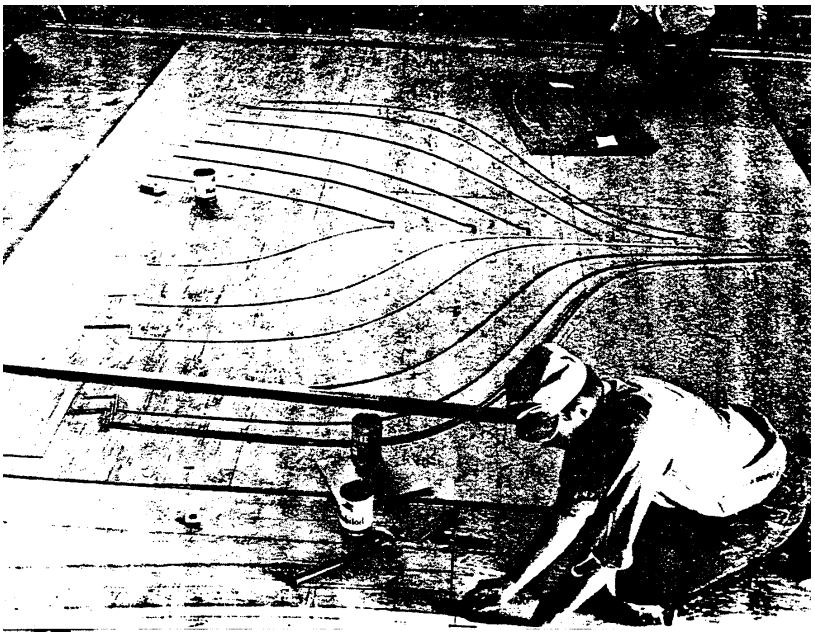


Figure 3: Lofting the sections of an ocean cruising sailboat.

1.2 Mathematical Hull Representation

The mathematical representation of the hull clearly avoids many of the problems indicated in the previous section. The shape is uniquely defined, and the coordinate at any desired point can be obtained with an accuracy suitable for full size manufacture. Massive amounts of data can be generated quickly for purposes of analysis or manufacture. A traditional looking lines drawing can be generated automatically to serve as a visualization of the hull, but this is no longer the source of definition of the hull shape. A computer generated lines drawing with more finely spaced stations is shown in Figure 4. But other forms of visualization can also be provided, including perspective views from any angle which can provide the designer with an idea of the true three dimensional shape in a way which is similar to the old carved model. An example is shown in Figure 5. However, the effect is much better when you can control the view youself on a high-resolution color monitor. You can do this using a program called *FASTSHIP* which is available on Athena.

All of this sounds good, but how does one represent a complicated three dimensional object mathematically? Simple geometries such as an ellipsoid can be defined by a single equation, but a ship as shown in Figure 2 certainly can not. It turns out that it is not easy, and that the development of good methods of mathematical hull representation is a currently active research field. A proper introduction to this field requires that we start at the beginning and look at the representation of plane curves, which we will do in section 2.

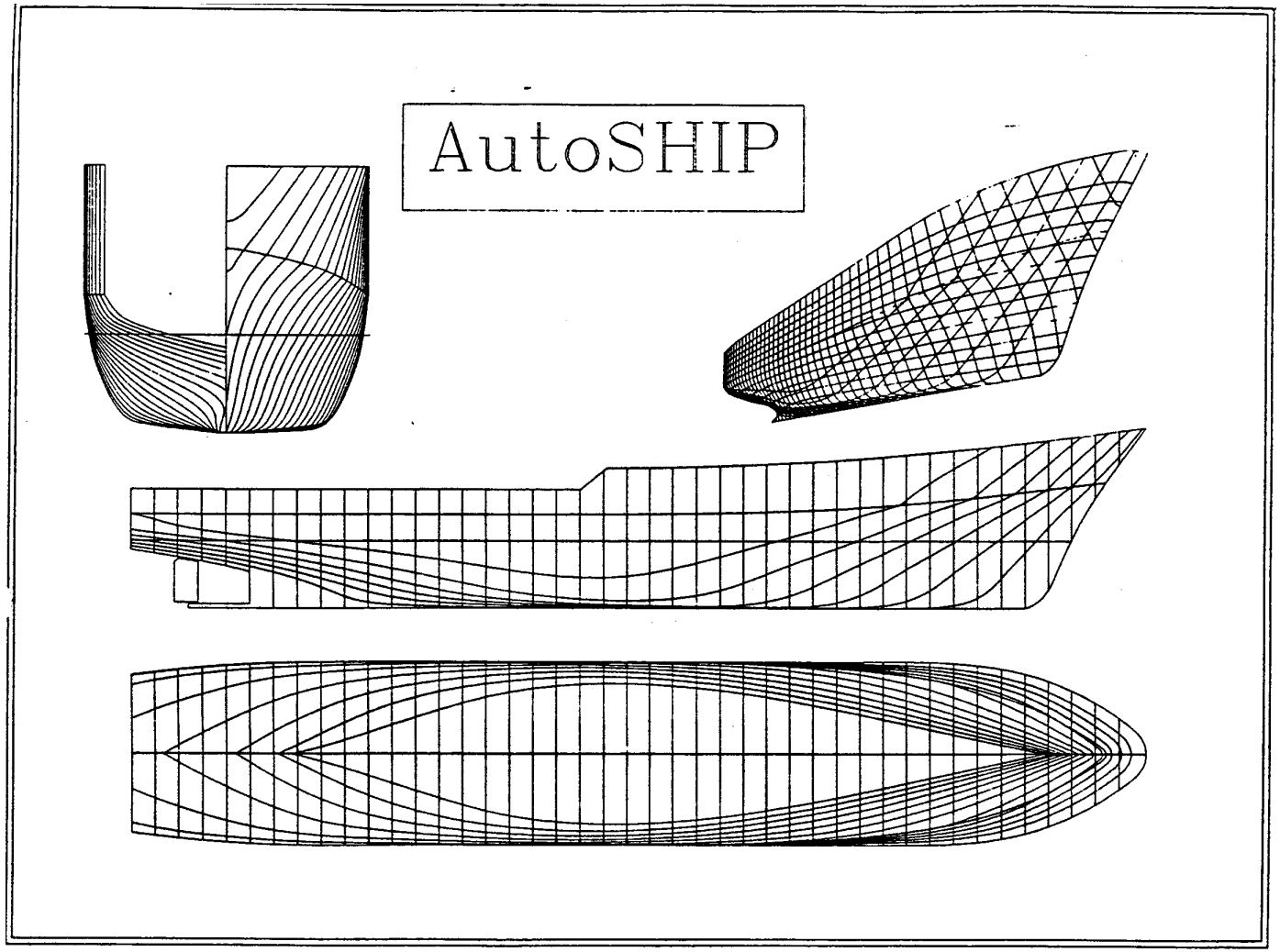


Figure 4: Example of a lines drawing generated from a mathematical fairing algorithm. From **AutoSHIP**, one of several ship hull CAD systems available for personal computers.

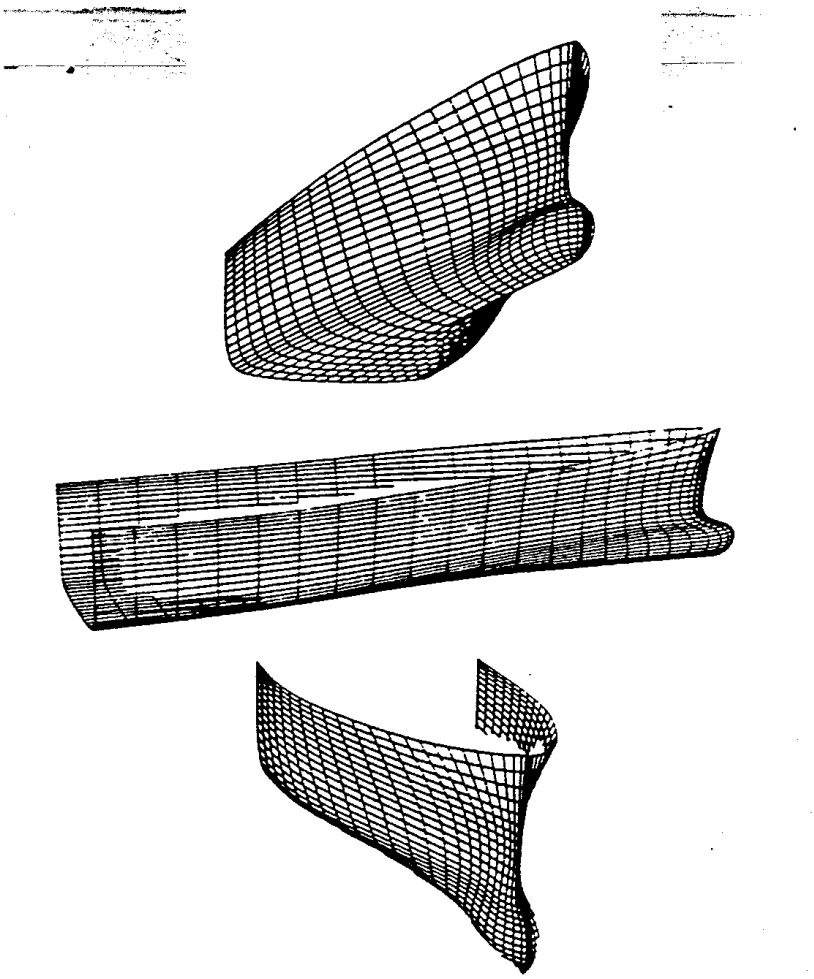


Figure 5: Perspective views of a mathematically defined hull surface with hidden lines removed. This figure is from "Some Practical problems in Mathematical Fairing", Hatton and Matida, SNAME SHAD77 Symposium,1977.

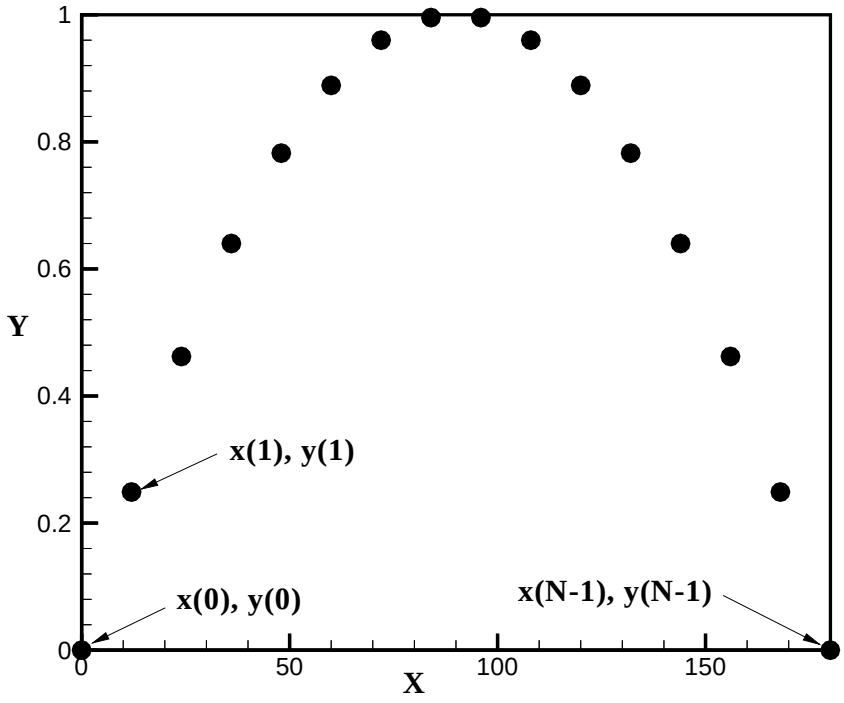


Figure 6: Notation for coordinates and base points

2 FITTING PLANE CURVES WITH POLYNOMIALS

We start out by defining a cartesian coordinate system with the x axis horizontal and the y axis vertical as shown in Figure 6. Suppose there is some curve in the xy plane, which to begin with, will have y as a single valued function of x. In other words, the curve does not double back on itself, nor does its slope ever become vertical. We would like to represent this curve mathematically, but all we know about it is what we see on the graph.[2] Perhaps we just drew it with a pencil (or a computer mouse), or that someone else generated it from some complicated formula which we are not allowed to see. Perhaps we are not given a curve at all, but simply know the value of y at a discrete set of values of x.

The simplest mathematical representation of such a curve would be a polynomial

[2]Exercise No. 1: Plot exercise using splot. Draw $y = \sin(x)$, $0 \leq x \leq \pi/2$

of degree $N-1$, which can be expressed in the form

$$y(x) = a_0 + a_1 x + a_2 x^2 + \ldots a_{i-1} x^{i-1} + \ldots a_{N-1} x^{N-1} \tag{1}$$

where the coefficients a_i and the degree $N-1$ must be determined in some way. A polynomial is simple since it requires simple mathematical operations to evaluate y at a given x, or to obtain the derivative or the integral of the function. Of course, the lower the degree $N-1$, the easier it is to evaluate.

We can find a polynomial of degree $N-1$ which passes through N arbitrarily spaced distinct points $x_0, x_1, \ldots, x_{N-1}$. These points are given various names, such as **base points** or **collocation points**. We will use the term **base points** here, since it is easier to spell.

The requirement that the polynomial pass through the N base points can be expressed in the form of a set of simultaneous equations,

$$\begin{gathered}
a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_{i-1} x_0^{i-1} + \cdots + a_{N-1} x_0^{N-1} = y_0 \\
a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_{i-1} x_1^{i-1} + \cdots + a_{N-1} x_1^{N-1} = y_1 \\
\cdots \\
\cdots \\
\cdots \\
a_0 + a_1 x_{N-1} + a_2 x_{N-1}^2 + \cdots + a_{i-1} x_{N-1}^{i-1} + \cdots + a_{N-1} x_{N-1}^{N-1} = y_{N-1}
\end{gathered} \tag{2}$$

For example, suppose we are given the following three values $(x = 1, y = 1)$, $(x = 2, y = 4)$ and $(x = 3, y = 9)$. If we were particularly lacking in imagination, we might write down the following set of simultaneous equations,

$$\begin{gathered}
a_0 + a_1 + a_2 = 1 \\
a_0 + 2a_1 + 4a_2 = 4 \\
a_0 + 3a_1 + 9a_2 = 9
\end{gathered}$$

which we could then solve for the a's, with the result

$$a_0 = 0 \qquad a_1 = 0 \qquad a_2 = 1$$

Thus the desired polynomial is $y = x^2$. Continuing in our systematic way, we can evaluate y at any given x from equation 1. For example, at $x = 1.5$ we find

$$y(1.5) = 0 + 0 \times 1.5 + 1 \times 1.5^2 = 2.25$$

One could easily write a computer program which finds the coefficient of a polynomial which passes through N base points by forming the equations in 2 and using some standard subroutine for solving them. Then, the value of y at any x could be found readily from 1.

There is nothing wrong with this approach, except that it is generally not the most efficient way to do it. This is because the solution of a set of simultaneous equations requires a considerable number of multiplications and divisions, and that this number is roughly proportional to the cube of the number of equations. Thus, finding the polynomial that passes through 9 points would take 27 times as long as the simple 3 point example just given. This may still not be very long to do once on a computer, but in an interactive CAD system, this type of operation may need to be done a large number of times while the impatient designer is staring at the screen. Two other methods of finding the polynomial will therefore be described, **Lagrange Interpolation** and **Newton's Divided Difference Interpolation**. Both are commonly used, and are much faster.

2.1 LAGRANGE INTERPOLATION

Suppose we have N base points, as before, and form the following polynomial,

$$y(x) = C\big[(x - x_0)(x - x_1)\dots(x - x_{N-1})\big] \tag{3}$$

where C is an unknown constant. This is a polynomial of degree N which is zero at each of the base points. So far this is not too good, since we want a polynomial of degree $N - 1$ which has certain prescribed values at the base points. Now suppose that we modify 3 by eliminating one of the factors, say the one containing the i'th base point. Now we have a polynomial of degree $N - 1$ which is zero at all the base points except the i'th one. Finally, let us make the polynomial have a value of one at the i'th base point by replacing the constant C with the reciprocal of the value of the numerator at $x = x_i$, and call the resulting polynomial $L_i(x)$

$$L_i(x) = \frac{(x - x_0)(x - x_1)\dots[omit(x - x_i)]\dots(x - x_{N-1})}{(x_i - x_0)(x_i - x_1)\dots[omit(x_i - x_i)]\dots(x - x_{N-1})} \equiv \frac{\prod_{j=0,j\neq i}^{N-1}(x - x_j)}{\prod_{j=0,j\neq i}^{N-1}(x_i - x_j)} \tag{4}$$

Given N base points, we can form N different polynomials $L_0(x), L_1(x), \dots, L_{N-1}(x)$. Now the polynomial that we are looking for can be expressed of as a linear combination of these N different polynomials as follows,

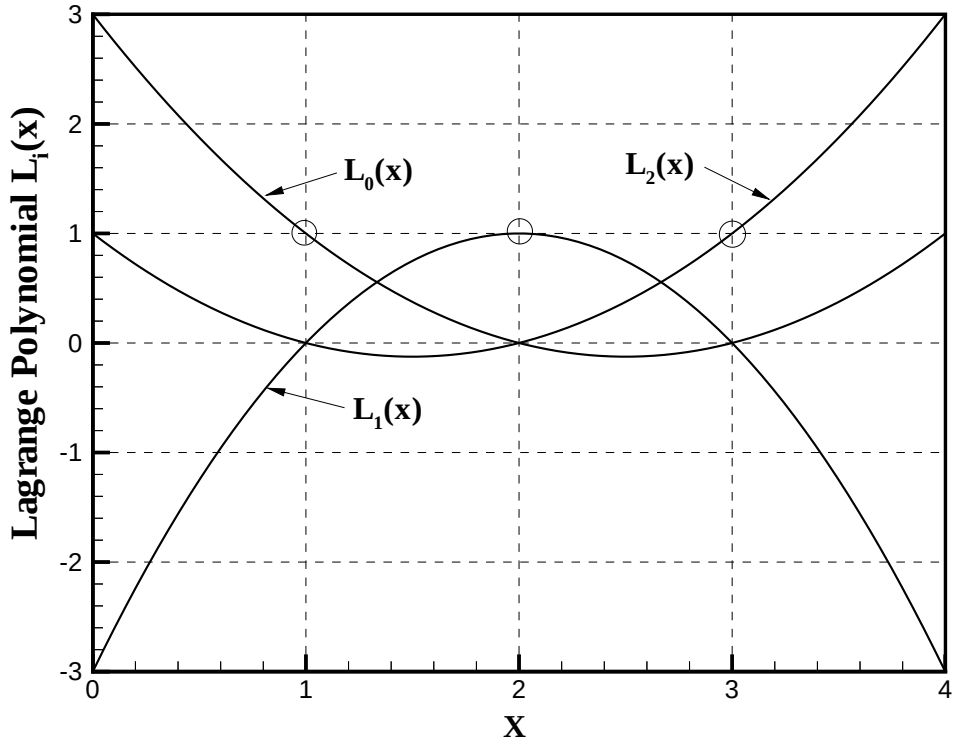


Figure 7: Lagrange interpolation polynomials for the base points $x_0 = 1, x_1 = 2, x_2 = 3$

$$y(x) = \sum_{i=0}^{N-1} y_i L_i(x) \tag{5}$$

This works because $L_i(x)$ has a value of one at $x = x_i$ and a value of zero at each of the other base points.

We can check this by doing the same example as in the preceding section, where there were three base points at $x = 1$, $x = 2$ and $x = 3$. Plots of the resulting three polynomials $L_0(x), L_1(x)$ and $L_2(x)$ are shown in Figure 7, and it is clear that they have values of zero and one in the right places. The plots were made by computing equation 4 for 101 values of x in the interval from 0 to 4. We can check this by the values of the three polynomials at a single point, say $x = 1.5$ as follows,

$$L_0(1.5) = \frac{(1.5-2)(1.5-3)}{(1-2)(1-3)} = 0.375$$

$$L_1(1.5) = \frac{(1.5-1)(1.5-3)}{(2-1)(2-3)} = 0.750$$

$$L_2(1.5) = \frac{(1.5-1)(1.5-2)}{(3-1)(3-2)} = -.125$$

We can see that these values agree with those shown in Figure 7. Now to find the value of y at $x = 1.5$ given that $y_0 = 1$, $y_1 = 4$ and $y_2 = 9$ as before, we can use equation 5, giving the result,

$$y(1.5) = 1.0 \times 0.375 + 4.0 \times 0.750 - 9 \times 0.125 = 2.25$$

We can generalize the preceding example considerably. First of all, note that the polynomials $L_i(x)$ did not depend on y_i. We can therefore find the value of y at $x = 1.5$ for any set of three values of y given at the same values of x using 5. If we scale the x axis, by a factor of ten, we can see that the values of $L_i(x)$ are unchanged. Therefore the value of y at $x = 15$ is obtained from the values of y at $x = 10$,$x = 20$, and $x = 30$ in the same way. Finally, if we add a constant to x, the interpolation polynomials are again unchanged. For example, the value of y at $x = 2.5$ is obtained from the values of y at $x = 2$, $x = 3$, and $x = 4$ with the identical formula.

In other words, in this example we have found a way to determine the value of y midway between the first two points of a set of three equally spaced points. all we have to do is multiply the first value of y by 0.375, the second value of y by 0.750 and the third value of y by -.125 and add them up. Similar weight factors can be computed for any other position relative to the given base points.

This procedure must also work in the special case when y is equal to a constant. In this case, equation 5 becomes

$$y(x) = y_0 = y_0 \sum_{i=0}^{2} L_i(x)$$

$$\sum_{i=0}^{2} L_i(x) = 1.0$$

so we see that the sum of the interpolation polynomials at any value of x must be equal to one. This can serve as a check on the calculation, or can be used to speed up the computation by setting L_{N-1} equal to 1.0 minus the sum of the others.

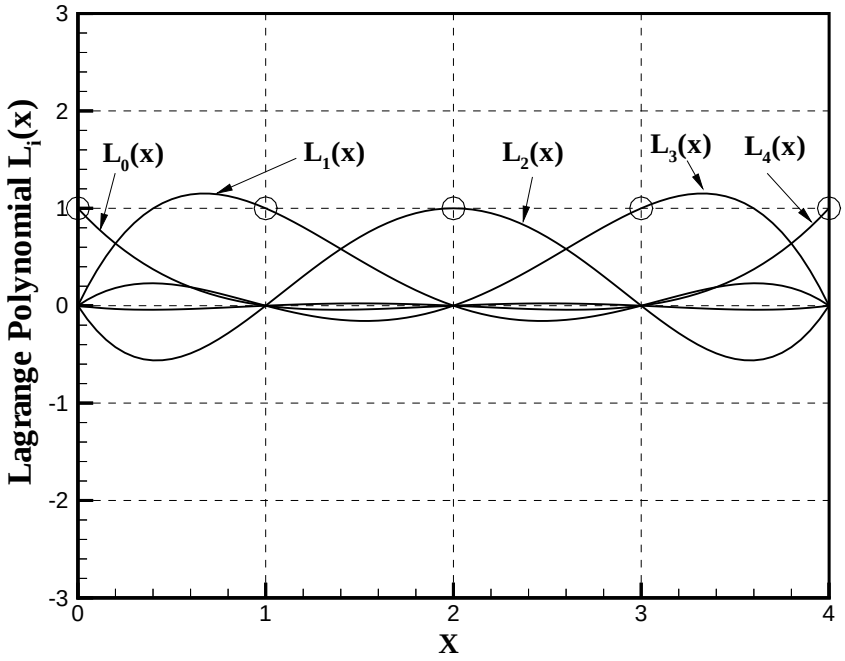


Figure 8: Lagrange interpolation polynomials of fourth degree for 5 base points $x_0 = 0, x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$

As a final comment, note from Figure 7 that the values of $L_i(x)$ fall between the values of -.125 and 1.0 in the interval between the base points, but that outside of this interval the values grow rapidly in magnitude. This is important if one tries to extrapolate values of y. The value of y at $x = 0$, for example, would be very sensitive to small errors in the values of the base points. The problem gets worse if the number of base points, (and therefore the degree of the polynomial) is increased.

extrapolatio
problem

Figure 8 shows the Lagrange interpolation polynomials of fourth degree where the base points cover the range from $(0, 4)$. Now there is no extrapolation in this range, and the magnitude of the polynomials is only slightly greater than 1.0.[3]

[3]Exercise No. 2:

1. Develop a function Lagrange().
2. Generate Figures 7 and 8 and a curve $y = x^2$.

2.2 EXAMPLES OF POLYNOMIAL FITS

The remaining figures in this section show some examples of polynomial fits. While these were all computed using the Lagrange interpolation method developed in the preceding section, the results would be the same using the direct solution of the simultaneous equations which we described first, or by Newton's divided difference method which we will develop in the next section.

The first two examples show polynomial approximations to the function $y = \sin(x)$ in the interval from $0° \leq x \leq 180°$. The first one shows a second degree polynomial passing through three points, and the second one shown a fifteenth degree polynomial passing through sixteen points. Both curves go smoothly through the given points, and an increase in the degree of the polynomial increases the accuracy of the fit.

The remaining two examples show how things can go wrong. In the first case we examine polynomial approximations to a quarter of an ellipse of eccentricity 0.1, that is, $(x-1)^2 + (\frac{y}{0.1})^2 = 1$

$$\begin{aligned} x &= 1 - \cos\alpha \\ y &= 0.1 \sin\alpha \\ 0 &\leq \alpha \leq \pi/2 \end{aligned} \tag{6}$$

where the base points are equally spaced values of the parameter α [4]. Figure 11 shows a second degree polynomial approximation, which is smooth, but not nearly as accurate as the second degree approximation to the sine function.

Figure 12 shows an eighth degree approximation, which is a disaster. In fact, it is so bad that the scale of the y axis needed to be changed, as shown in figure 13.

The last example is a polynomial approximation to a curve which consists of three straight line segments. Here we might anticipate troubles since the function to be fitted has discontinuous slopes. The result is shown in figure 14.

[4] We will be looking at parametric representation of curves in more detail soon.

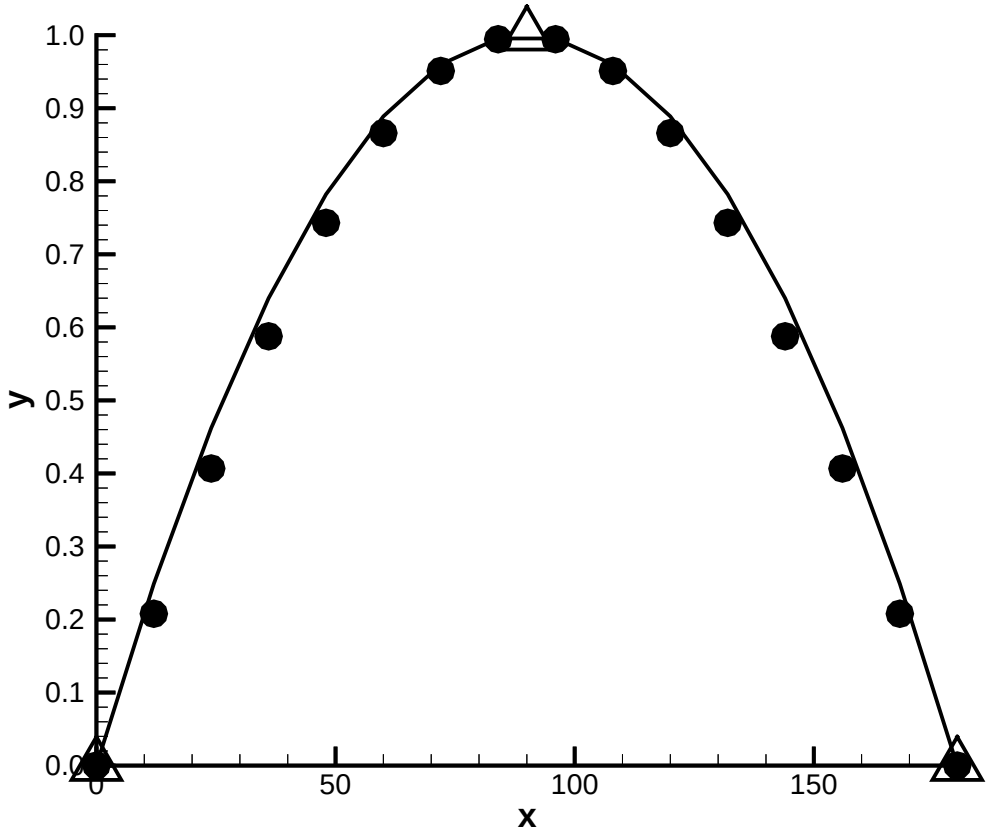


Figure 9: Second degree polynomial approximation, $y = \sum_{i=0}^{2} a_i x^i$, to $sin(x)$ using three base points at 0, 90 and 180 (identified by triangular symbols). The cicles are the true values of $sin(x)$ plotted at twelve degree intervals of x. While there is some error, the approximation is very smooth, and is as good as one could expect with only a second degree polynomial.

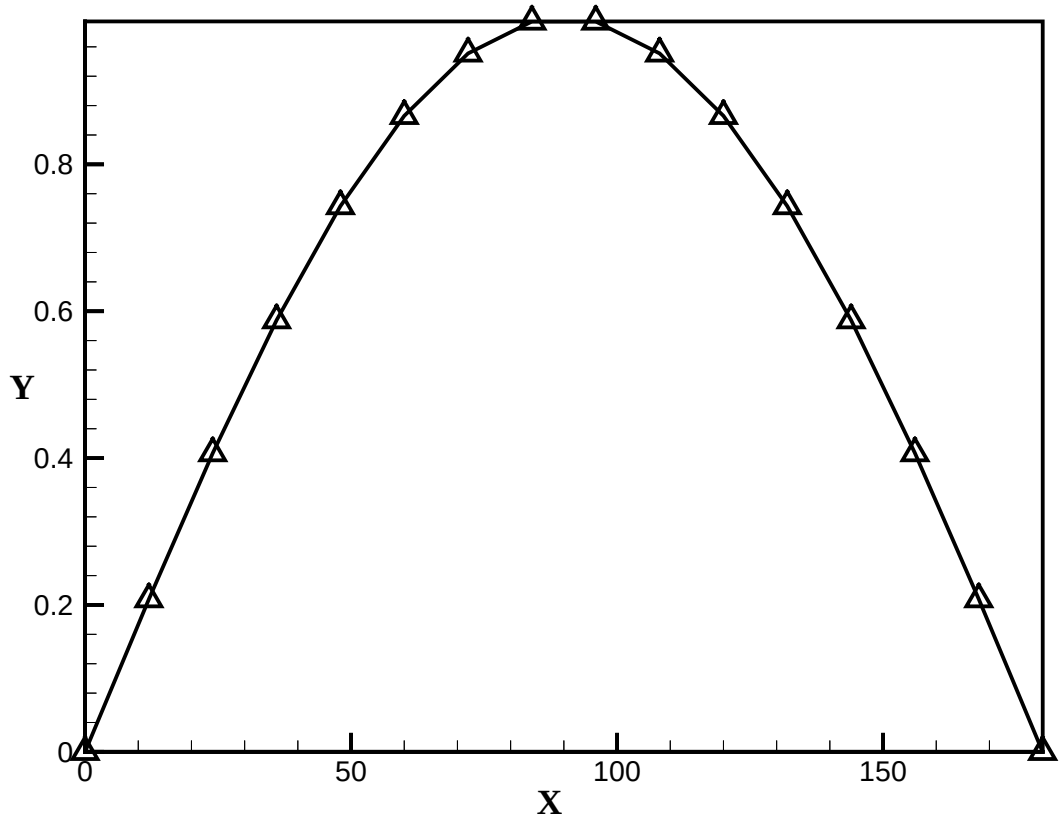


Figure 10: Fifteenth degree polynomial approximation to $sin(x)$, that is, $y = \sum_{i=0}^{15} a_i x^1$. The sixteen base points are shown as circles, so that the polynomial passes exactly through all of them. But, in addition, the fit is excellent in between. This is an example of a high degree polynomial approximation that works!

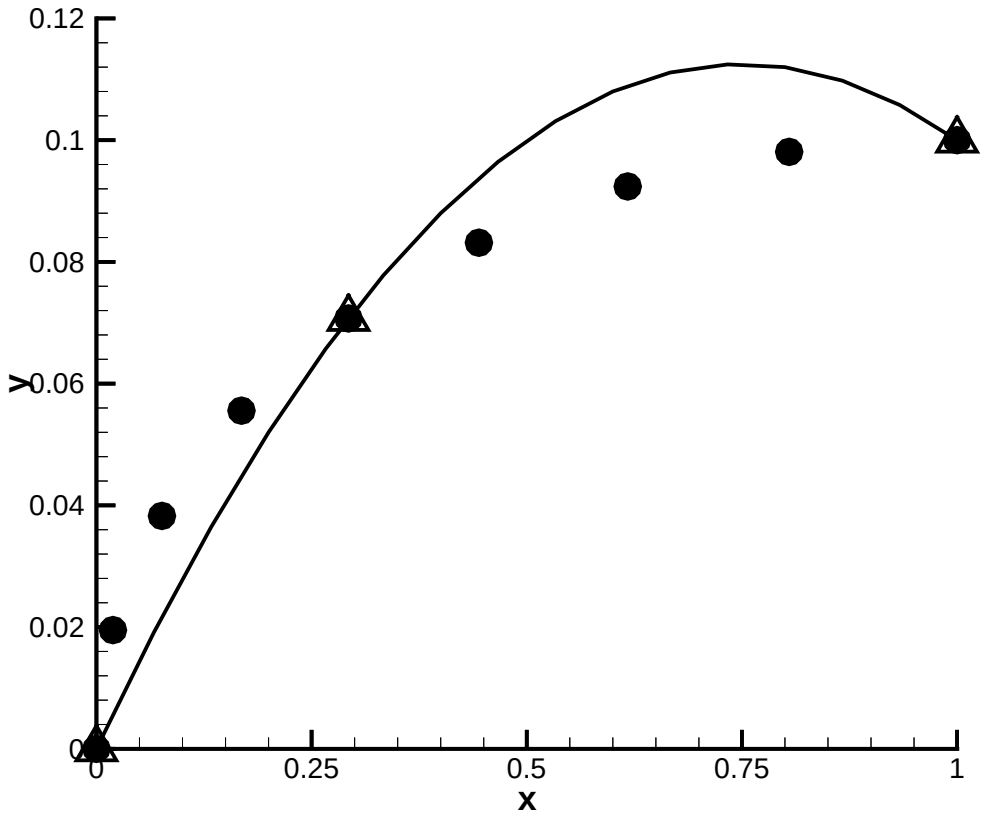


Figure 11: Second degree polynomial approximation to one quarter of an ellipse,$y = \sum_{i=0}^{2} a_i x^i$. The circles are points on the ellipse, but only the first, fifth and ninth points are used in fitting the polynomial(identified by triangular symbols). The fit is not nearly as good as in the case of the sine function.

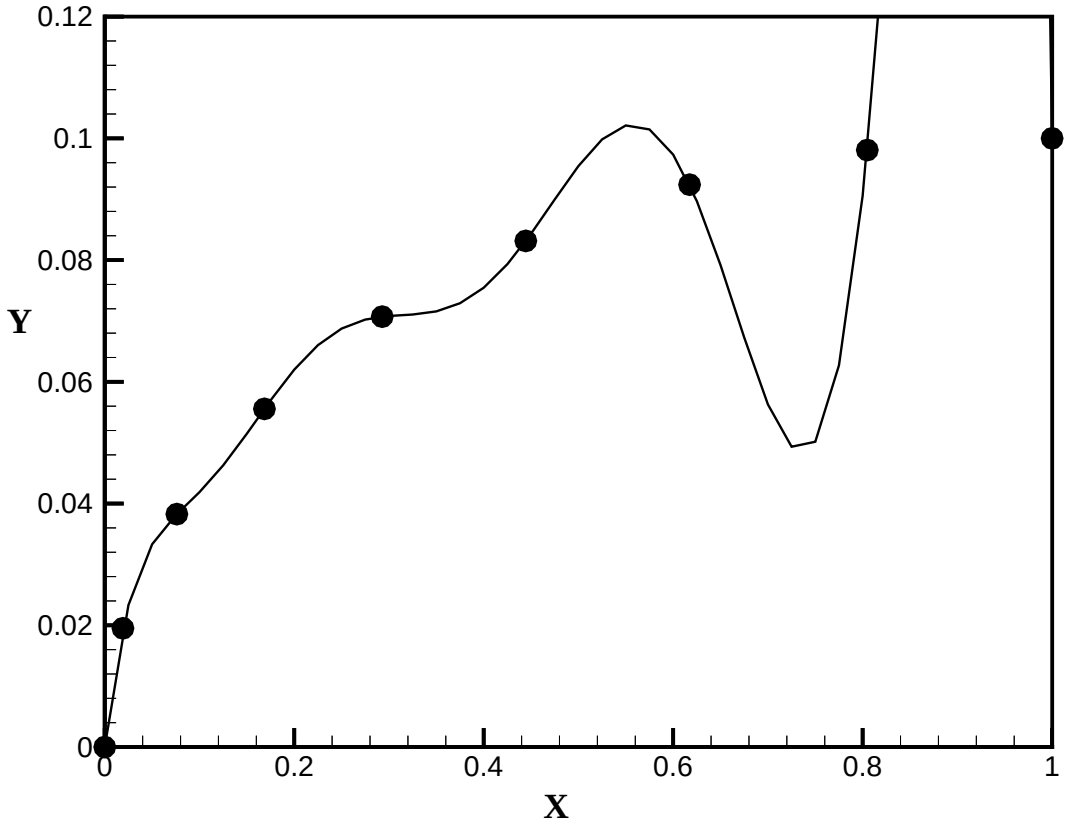


Figure 12: This is an eighth degree polynomial fit to one quarter of an ellipse. In this case, all nine points on the ellipse (shown as circles) are used as base points, and the polynomial passes through all of them, as it should. However, the fit between base points is bad, and the polynomial goes into orbit starting around $x = 0.8$. The problem is caused by the fact that the ellipse has infinite slope at $x = 0$. This causes the polynomial to go crazy at the other end of the interval, where one would hardly expect troubles.

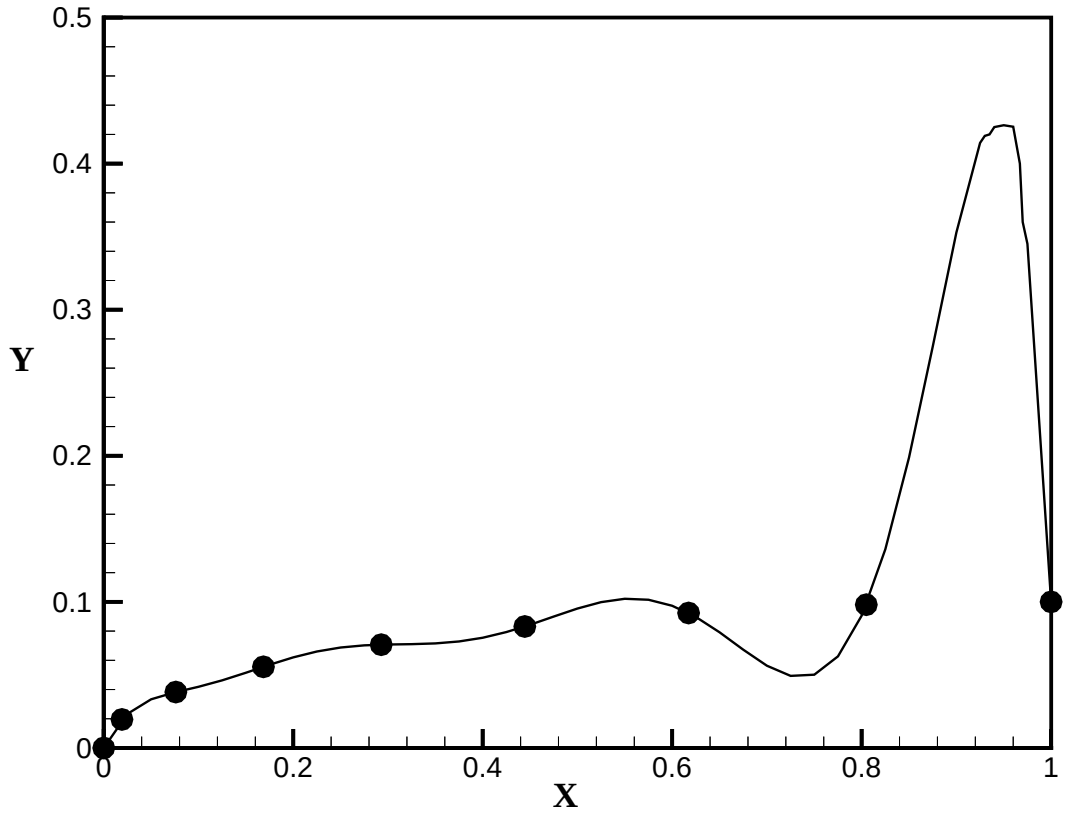


Figure 13: This is another plot of the same poynomial, with the y axis scale changed to show the entire wild oscillation of the polynomial fit. Imagine if you were performing some calculations based on this polynomial fit, and had neglected to plot it beforehand!

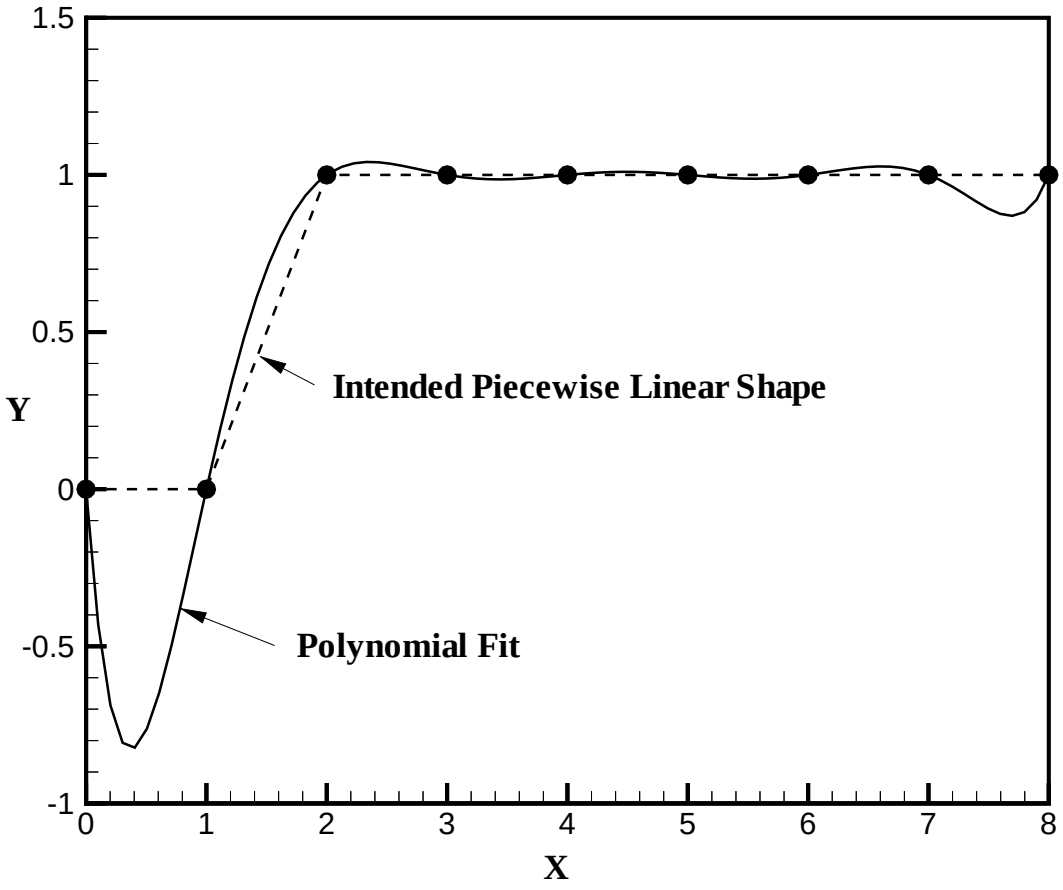


Figure 14: This is an eighth degree polynomial to a function with discontinuous slopes, as shown by the short dashed line with circular symbols. If the function to be fitted represented something which physically could not be negative (for example, the distribution of cross-sectional area of an underwater vehicle), the negative values in the polynomial approximation could cause big problems!

2.3 Newton's divided difference method

Another efficient way of finding a polynomial passing through a given set of base points is the divided difference method originally devised by Newton. Rather than deriving the general case, we will first show how the method works in the case of three base points. We will then see how the algorithm can be easily extended to any number of points.

In the case of three base points, the polynomial as defined in equation 1 would looks as follows,

$$y(x) = a_0 + a_1 x + a_2 x^2 \tag{7}$$

but this could also be expressed in factored form

$$y(x) = d_0 + d_1(x - x_0) + d_2(x - x_0)(x - x_1) \tag{8}$$

where x_0, x_1, x_2 are the base points, and d_0, d_1, d_2 are a different set of constants which must be related in some way to the a's. If we know the d's, we can evaluate the polynomial more efficiently by *nesting* as follows

Find d_0, d_1, d_2

$$y(x) = d_0 + (x - x_0)\left[d_1 + (x - x_1)d_2\right] \tag{9}$$

By setting $x = x_0$, $x = x_1$ and $x = x_2$ respectively, we obtain the following three simultaneous equations for the d's[5]

$$\begin{aligned} y_0 &= d_0 \\ y_1 &= d_0 + d_1(x_1 - x_0) \\ y_2 &= d_0 + d_1(x_2 - x_0) + d_2(x_2 - x_0)(x_2 - x_1) \end{aligned} \tag{10}$$

Since this set of equations is in triangular form, the solution can be written down directly

$$\begin{aligned} d_0 &= y_0 \\ d_1 &= \frac{y_1 - y_0}{x_1 - x_0} \\ d_2 &= \frac{(y_2 - y_0) - \frac{(y_1 - y_0)}{(x_1 - x_0)}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \end{aligned} \tag{11}$$

[5]Logic to compute $y(x)$, given d_i:
$a = d_{N-1}$;
for $(i = N - 2;\ i \geq 0; i--)$ $\{a = (x - x_i)a + d_i;\}$
$y = a$;

After some algebra[6] the expression for d_2 can be reduced to

$$d_2 = \frac{\frac{(y_2-y_1)}{(x_2-x_1)} - \frac{(y_1-y_0)}{(x_1-x_0)}}{(x_2 - x_0)}$$

While this still looks complicated, the nice thing about this method is that numerical values for the d's can be calculated from what is known as a divided difference table. The first column of the table is formed from the x coordinates of the base points, while the second column contains the y values. The third column is formed by dividing the differences in y values between the next row and the present row by the corresponding differences in x. The first element of the fourth column is the difference between the second and first elements of the preceding (third) column divided by the difference between the **third** and first x values. Similarly, the second element of the fourth column is the difference between the third and second elements of the preceding column divided by the difference between the **fourth** and second x values.

The only tricky part is to remember that in doing the fourth column that you divide by the difference between the x value **two** rows down from the current row and the x value in the current row. In the general case, this difference in rows increases by one for each successive column. This is difficult to state clearly in words, but will hopefully be clear from the examples to follow, or even better, from the listing of the computer code.

The resulting divided difference table looks as follows,

$$\left|\begin{array}{c|c|cc} x_0 & y_0 & \frac{y_1-y_0}{x_1-x_0} & \frac{\frac{(y_2-y_1)}{(x_2-x_1)} - \frac{(y_1-y_0)}{(x_1-x_0)}}{(x_2-x_0)} \\ x_1 & y_1 & \frac{y_2-y_1}{x_2-x_1} & \\ x_2 & y_2 & & \end{array}\right| \equiv \left|\begin{array}{c|ccc} x_0 & d_{00} & d_{10} & d_{20} \\ x_1 & d_{10} & d_{11} & \\ x_2 & d_{20} & & \end{array}\right| \qquad (12)$$

[6]

$$\begin{aligned} d_2 &= \frac{(y_2 - y_1 + y_1 - y_0) - \frac{(y_1-y_0)}{(x_1-x_0)}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\ &= \frac{\frac{y_2-y_1}{x_2-x_1}}{x_2 - x_0} + \frac{\frac{y_1-y_0}{x_1-x_0}\left[(x_1 - x_0) - (x_2 - x_0)\right]}{(x_2 - x_0)(x_2 - x_1)} \end{aligned}$$

The desired coefficients d_0, d_1 and d_2 are first elements of the second, third and fourth columns, respectively.[7]

This can be illustrated by the same simple example, $y = x^2$ used previously. The divided difference table in this case is

$$\left|\begin{array}{c|ccc} & d_0 & d_1 & d_2 \\ 1.000 & 1.0000 & 3.0000 & 1.0000 \\ 2.000 & 4.0000 & 5.0000 & \\ 3.000 & 9.0000 & & \end{array}\right| \tag{13}$$

Substituting the d's in equation 9 we find that

$$y = 1 + (x-1)[3 + (x-2)] = x^2$$

which recovers the original equation for the polynomial. Now suppose that we used the same polynomial to generate four base points at $x = 1, 2, 3, 4$. We can then pretend that we do not know the answer, and attempt to find the third degree polynomial which passes through these points. The resulting divided difference table looks as follows

$$\left|\begin{array}{c|cccc} 1.000 & 1.0000 & 3.0000 & 1.0000 & 0.0000 \\ 2.000 & 4.0000 & 5.0000 & 1.0000 & \\ 3.000 & 9.0000 & 7.0000 & & \\ 4.000 & 16.0000 & & & \end{array}\right| \tag{14}$$

As might be expected, the fourth divided difference (which is at the top of the fifth column) comes out to be zero, so that we again recover the original second degree polynomial $y = x^2$.

We will next do a less trivial case where the function to be fitted is not a polynomial, and the base points are not equally spaced. The function is $y = \ln(x)$ and the base points are at $x = 2, 4, 5, 7$. The divided difference table looks as follows

[7] Logic to compute d_i:
for $(i = 0;\ i < N; i++)\{d_{i0} = y_i\}$;
for $(j = 1;\ j < N; j++)$
for $(i = 0;\ i < N - j; i++)\ \{d_{ij} = (d_{i+1,j-1} - d_{i,j-1})/(x_{i+j} - x_i);\}$

$$\left|\begin{array}{c|cccc} 2.000 & 0.6931 & 0.3466 & -.0412 & 0.0046 \\ 4.000 & 1.3863 & 0.2231 & -.0183 & \\ 5.000 & 1.6094 & 0.1682 & & \\ 7.000 & 1.9459 & & & \end{array}\right| \tag{15}$$

The third degree polynomial approximation to the natural logarithm is then

$$\ln(x) \approx 0.6931 + (x-2)\left[0.3466 + (x-4)[-.0412 + (x-5)(0.0046)]\right]$$

Substituting $x = 6$ in the above equation, we obtain the result,

$$\ln(6.0) \approx 1.7867$$

which can be compared with the true value of 1.7918. The result is not bad, although a pocket calculator can obviously do much better![8]

[8]Exercise No. 3: Develop a function Newton().

3 IMPROVING POLYNOMIAL FITS

We saw in Section 2 that polynomial fits gave poor results for functions which had **abrupt changes in slope**, or which had **infinite slopes at the ends**. In addition, a polynomial could not represent **a curve which doubled back on itself**. We will now introduce a variety of techniques which can overcome some of these problems, namely,

a) Rotation of Coordinates
b) Piecewise Approximations
c) Parabolic Blending
d) Parametric Representation

3.1 Rotation of Coordinates

We saw in Section 2.2 that a polynomial fit to a curve which had an infinite slope was a failure. This can be avoided in some cases by a rotation of the coordinate system by an angle θ, and generating a new set of base points

$$\xi = x\cos\theta + y\sin\theta$$

$$\eta = y\cos\theta - x\sin\theta$$

which are the same points as before, but represented in the rotated coordinate system. A polynomial approximation to $\eta(\xi)$ is obtained and evaluated at a sequence of values of ξ. These points are then transformed back to the original (x, y) coordinate system.[9]

Of course, this will fail again if the angle θ causes a previously finite slope to become infinite. But in the case of a quarter ellipse, a rotation angle of 45 degrees will change the 90 degree (infinite) slope at $x = 0$ to 45 degrees, and the zero slope at $x = 1$ to -45 degrees.

Figure 15 shows the result for the eighth degree polynomial fit shown before in figure 9. The result is much better, but the oscillation between the last three base points is still unacceptable. But let's not give up yet!

The x spacing of the base points that we used was constant in the parameter α in equation (6). This puts the second base point very close to $x = 0$, where the slope is large, and leaves a large space near $x = 1$, where the oscillation occurs. On the

[9]

$$\begin{aligned} y &= \xi\cos\theta - \eta\sin\theta \\ x &= \xi\sin\theta + \eta\cos\theta \end{aligned}$$

other hand, if we generate a set of 9 base points equally spaced in x, and go through the same coordinate rotation process, we obtain a much better result, as shown in figure 16.

It is important to note that the result of a polynomial fit depends both on the choice of the number and spacing of the base points, and on the choice of coordinate system. The latter is a particularly undesirable feature since the choice of a coordinate system to represent a physical object is arbitrary.

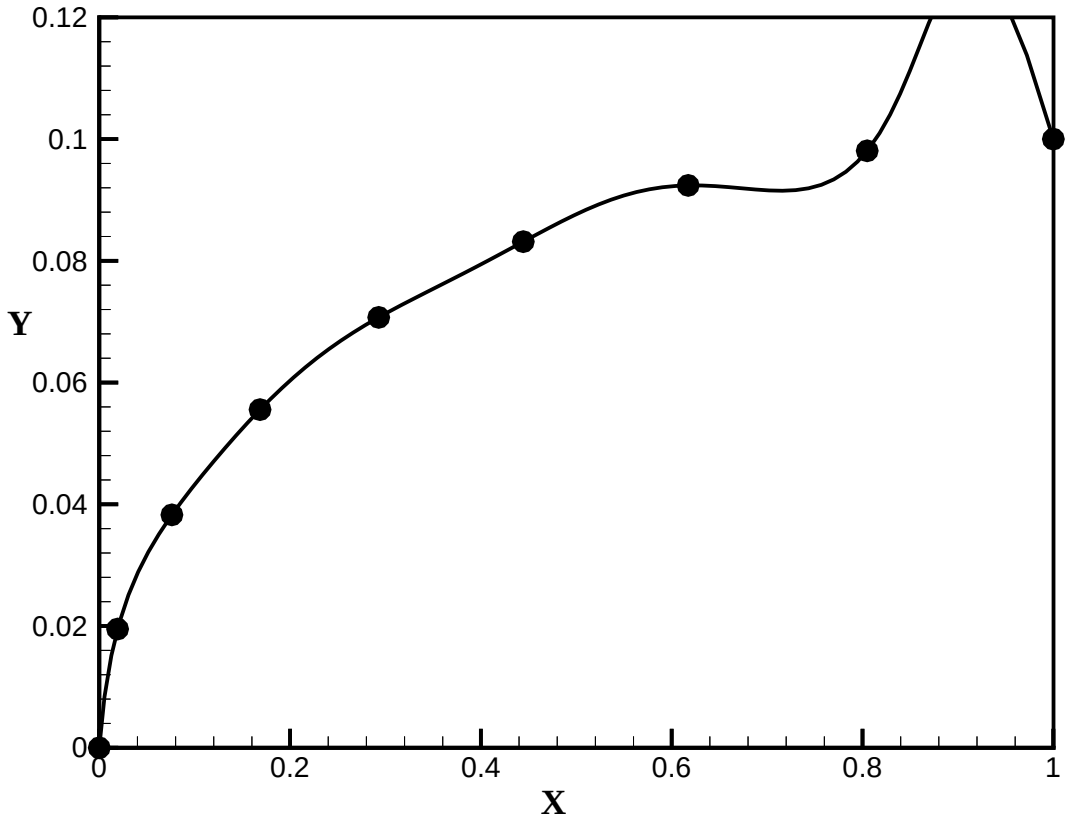


Figure 15: Eighth degree polynomial approximation to a quarter ellipse obtained by first rotating the coordinate system by 45 degrees. The magnitude of the oscillation between the last two base points is greatly reduced, but the fit is still not acceptable.($\Delta\alpha$ in (6) is constant, $\Delta\alpha = (\pi/2)/8$.)

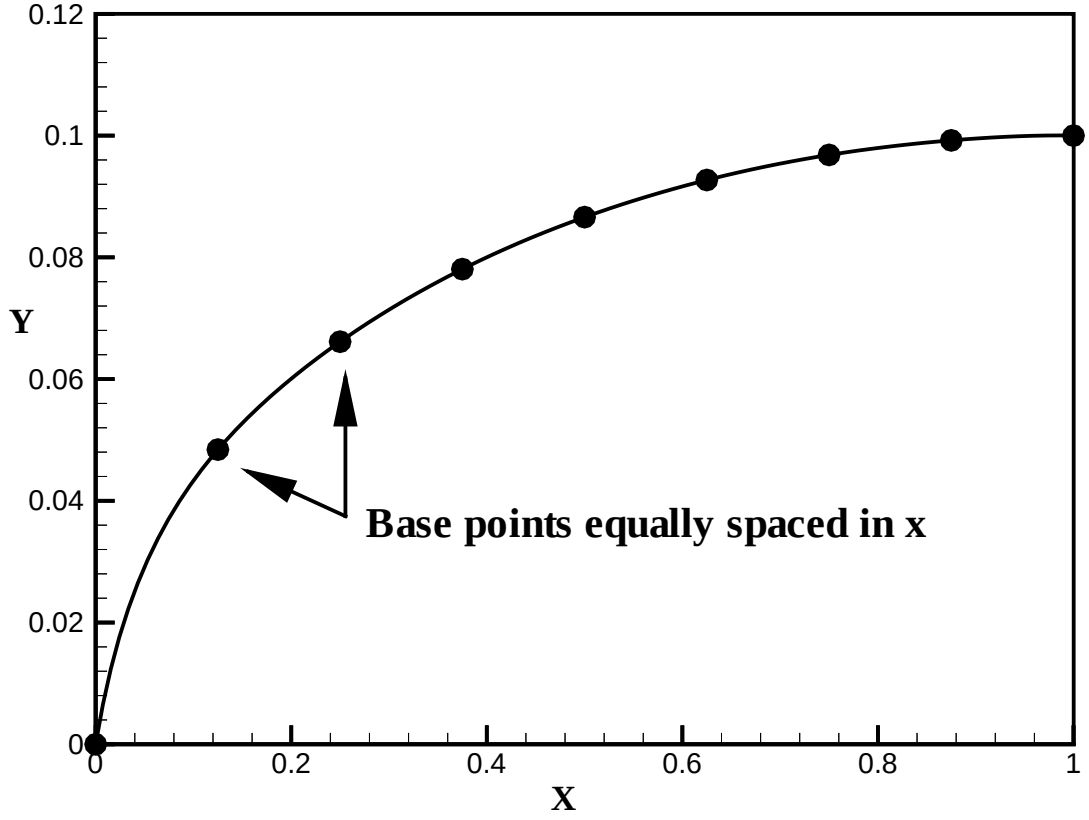


Figure 16: Eighth degree polynomial fit to a quarter ellipse using rotated coordinates as before, but where the base points are spaced equally in x. The fit is now quite smooth, and might be acceptable. However, note that the slope at $x = 0$ is not infinite. If this were to be joined to the bottom quarter of an ellipse, a discontinuous slope would exist at this point.($\Delta x = 1.0/8$.)

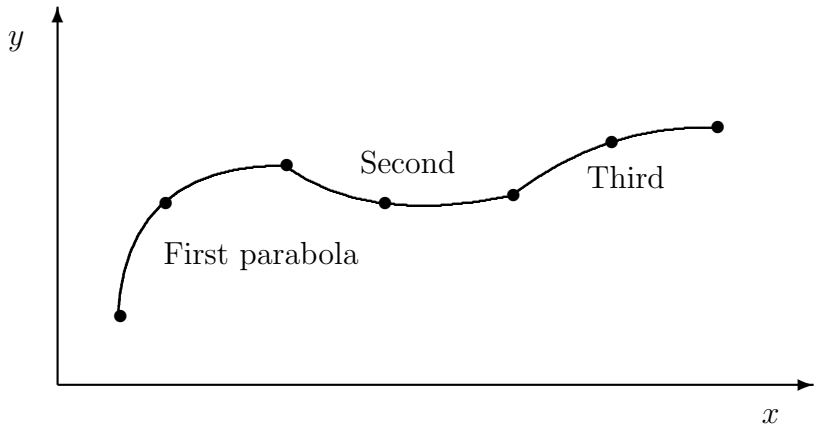


Figure 17: Piecewise approximation to a curve by a sequence of second degree polynomials, each passing through three points.

3.2 Piecewise Approximations

If we want to fit a curve through a large number of data points, it is not obvious that the degree of the polynomial should increase with increasing number of points. First of all, it is more work to find a 20'th degree polynomial than a 3'rd degree one, but also, the higher degree polynomial might be extremely unpredictable. An alternative is to fit lower degree polynomials through smaller groups of the data, and to piece them together. The ultimate piecewise fit is to use a first degree polynomial through pairs of two points, or in other words, to connect the points with straight lines.

While this might not be the most accurate way to fit a curve, it is always the safest! For example, a piecewise linear approximation to a quarter ellipse with nine points would have been much better than the 8'th degree polynomial illustrated in Figure 12.

Better results might be obtained by approximating a curve with a sequence of 2'nd degree polynomials passing through groups of three points, as shown in figure 17 below

This will generally be more accurate than a piecewise linear approximation, but the individual parabolas will not necessarily have matching slopes at the points where one joins to the next. The discontinuity in slope, however, will generally be less that would be the case for the piecewise linear approximation.

An algorithm for a piecewise polynomial approximation is as follows. We are given a sequence of base points $x_0, y_0, x_1, y_1, \ldots, x_{N-1}, y_{N-1}$ as before, and want to find y at a given x. The first step is to search through the table of base points to find the M points which are closest to x, where the degree of the polynomial is $M-1$. Either Lagrange interpolation or Newton's method can then be used to find y.

The next step in refinement is to try to make the individual polynomials more continuous at their boundaries. For example, a polynomial of degree 2 (parabola) is uniquely determined by three base points. But suppose, instead, that we fit a parabola through two points, and require as a third condition that the slope of this parabola at the boundary match the slope of the parabola in the next interval. Going one step further, suppose we fit a cubic through two points, and require that the first and second derivatives match the corresponding values of the adjacent cubics. We would then have a curve with continuous slope and curvature, which would then appear to the eye to be very smooth.

The latter is called a cubic spline, and represents physically the shape assumed by an elastic beam supported by a set of point loads along its length. Such a beam will have continuous first and second derivatives everywhere, but will have a discontinuous third derivative at each of the point loads. This is in contrast to the single polynomial passing through all the base points, which has all derivatives continuous. This property of elastic beams was used in drawing ship lines by using a real wood or plastic beam called a **SPLINE** supported by lead weights called **DUCKS**. The shape of the curve could be adjusted by moving the ducks around, and when the desired shape was obtained, the curve was transferred onto paper by running a pencil (or India ink pen if you were very skilled) along the spline.

The mathematical equivalent of the physical spline has become an extremely important tool in computer aided representation of curves, and we will derive their properties in Section 5.

3.3 Parabolic Blending

Another way to handle the discontinuities at the boundaries resulting from piecewise polynomial fits, and to provide a way of fitting curves which is independent of the coordinate system is called parabolic blending. This was developed by the Ford Motor Company around 1968, and is presented in detail in Rogers and Adams, " Mathematical Elements of Computer Graphics". We will only describe the general idea here, since the algebra gets fairly complicated. Shown in the sketch below is a curve represented by a set of base points numbered one through seven.

Suppose we want to evaluate the curve between points 4 and 5. We could put a polynomial through points 3,4 and 5 or through points 4,5 and 6. If we chose one or the other, we would have the problem of discontinuity in slope when we moved on to the next interval. Another way around this problem, is to let the curve between 4 and 5 be a blend of the two parabolas. We start at point 4 with the parabola through 3,4 and 5. When we reach midway between 4 and 5 we use half of this parabola and half of the parabola through 4,5 and 6. Finally, when we reach 5, we use all of the latter parabola.

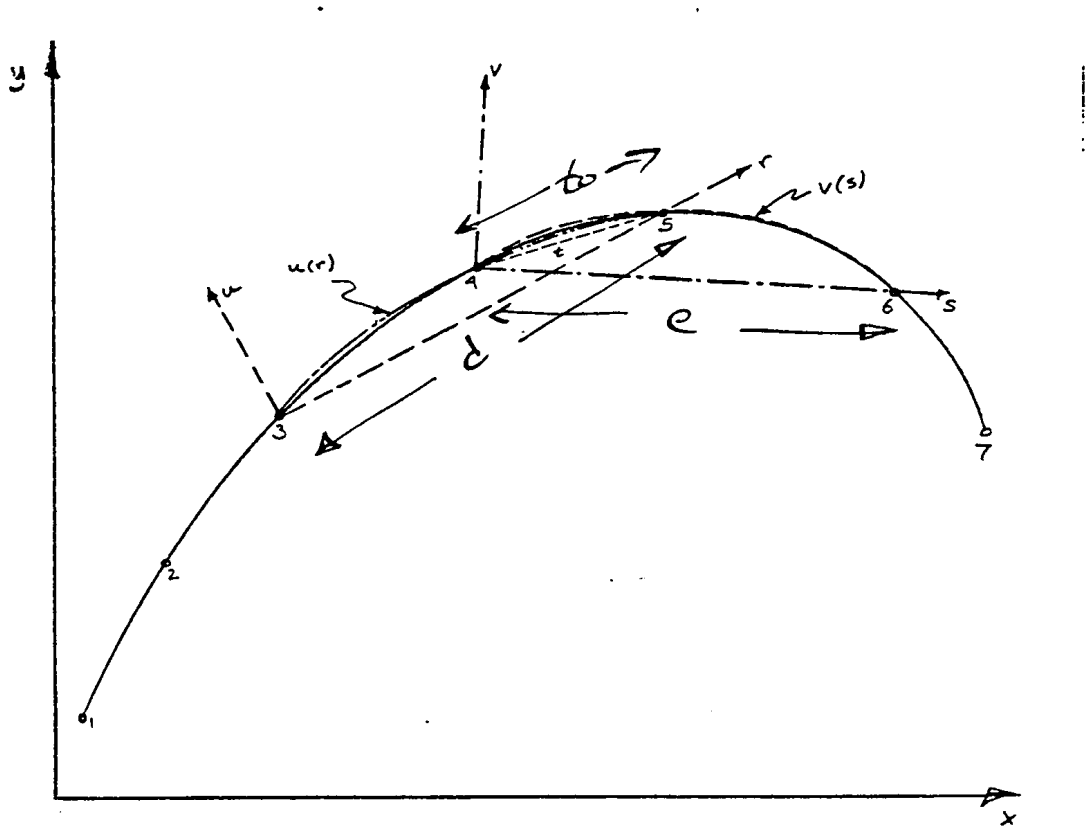


Figure 18: Definition of the various local coordinate systems required by the parabolic blending method of fitting a curve.

To get a little more specific, define a local (r,u) coordinate system, with the r axis passing through 3 and 5 as shown in the figure. A parabola in this coordinate system which passes through 3,4 and 5 can be written

$$u(r) = \alpha r(d - r)$$

where α is a constant determined so that the parabola goes through 4. Similarly,

$$v(s) = \beta s(e - s)$$

is the parabola in the next interval in terms of its local coordinates.

Finally, if t is the local coordinate going from 4 and 5, with $t = 0$ at point 4 and $t = t_0$ at point 5, the final blended curve can be expressed in the form,

$$C(t) = (1 - \frac{t}{t_0})u(r) + (\frac{t}{t_0})v(s)$$

Thus when $t << t_0$ $C(t) \approx u(r)$ and when $t \approx t_0$ $C(t) \approx v(s)$.

Rogers and Adams explain that this procedure is the mathematical equivalent to an artist sketching a curve by repeated pencil strokes, each one going over the earlier part of the curve, but extending it further. With an interactive computer system, the curve can be generated as the user is inputting points with a mouse or a light pen, only lagging behind one interval. A single polynomial fit, on the other hand, cannot be generated until all the points have been supplied.

Why did we start the explanation with point 4? Obviously a different starting up procedure must be used initially, since the first interval has no information to the left of it. This, and all the algebraic details of all the necessary coordinate transformations will not be covered here.

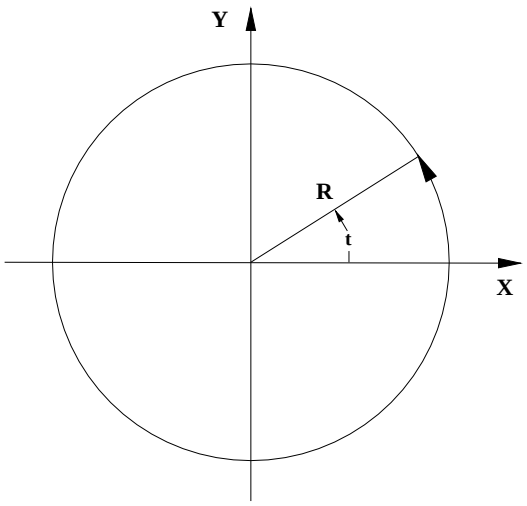


Figure 19: Parametric representation of the equation of a circle

3.4 Parametric Representation of Plane Curves

Up to now, we have only considered representing curves in the form $y = f(x)$. This has a number of disadvantages. First, the result depends on the choice of the coordinate system, which is not natural if the objective is to represent a physical object. Second, we cannot represent a function by a polynomial in x if it's derivative becomes infinite, or if it doubles back on itself (the latter implies that the derivative is infinite at least at one point).

A way to get around these difficulties is to use a parametric representation of the curve. In this case the curve is defined by two equations,

$$x = x(t) \qquad y = y(t) \tag{16}$$

where t is a *parameter*, which we would like to be related to the curve in a way which is independent of the coordinate system used. For example, suppose we wanted to describe a circle of radius R. As shown in figure 19, the parameter could be the angular coordinate of a point on the circle, i.e. $t = \theta$. Then, the equation of the circle, put in the form of equation 16 would be,

$$x = R\cos(t) \qquad y = R\sin(t) \tag{17}$$

Instead of the angular coordinate θ, we could let the parameter be the arc length along the circle, $t = R\theta$. The equation of the circle would then be

$$x = R\cos(t/R) \qquad y = R\sin(t/R) \tag{18}$$

This suggests a general procedure in which any curve is represented in terms of its arc length. The functional dependence on arc length could be a polynomial, i.e.

$$x(t) = \sum_{n=0}^{N-1} a_n t^n \qquad y(t) = \sum_{n=0}^{N-1} b_n t^n \tag{19}$$

where the a's and b's are now two sets of polynomial coefficients to be determined, as before, by requiring that the curve pass through a set of N base points. The only problem (in addition to the doubling of the amount of work that needs to be done) is that we do not necessarily know the value of arc length corresponding to each of the given base points.

However, we can do the next best thing, which is to assign parametric values to the base points which are the arc lengths along a polygon connecting the points. As shown in figure 20, the value of the parameter, t, at the n'th base point is

$$t(0) = 0 \qquad t_n = t_{n-1} + \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2} \quad n = 1, \ldots N-1 \tag{20}$$

The specific steps required to obtain a parametric polynomial are as follows:

1. Find t_n for each of the N base points.
2. Fit a polynomial of degree $N-1$ through (t_0, x_0), $(t_1, x_1) \ldots (t_{N-1}, x_{N-1})$. Note that x is the *dependent* variable in this case. Any method can be used to obtain the polynomial, a good choice being Newton's divided difference method.
3. Evaluate $x(t)$ at a large number of equally spaced values of t.
4. Fit a polynomial of degree $N-1$ through (t_0, y_0), $(t_1, y_1) \ldots (t_{N-1}, y_{N-1})$. Now y is the dependent variable.
5. Evaluate $y(t)$ at the same values of t as was just done for $x(t)$.
6. We now have $y(x)$ (or $x(y)$) .

Three examples of polynomial fits obtained with this procedure are shown in the following figures. One disadvantage of a parametric representation of a curve, is that y is not known as an explicit function of x. This means that if we want to find y at a given x, we must first find the value of t that gives the desired x value. This problem

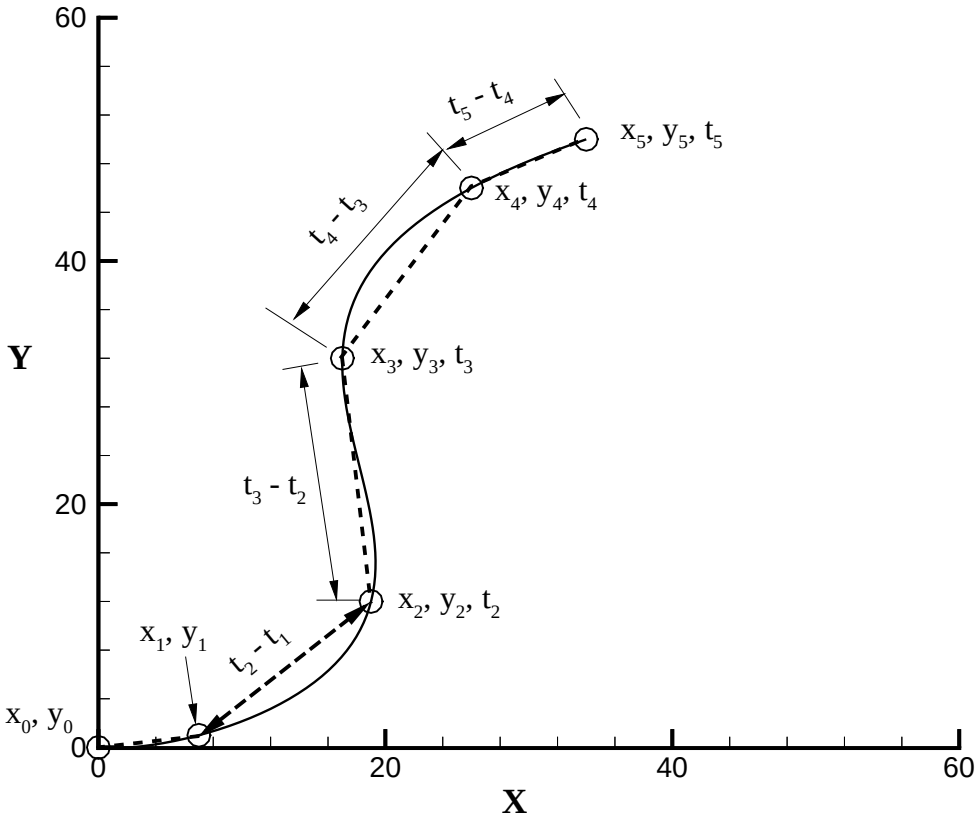


Figure 20: Representation of a "bulbous bow" ship section in parametric form where $y(t)$ and $x(t)$ are fifth degree polynomials passing through six base points. Note how the curve can double back on itself without difficulty.

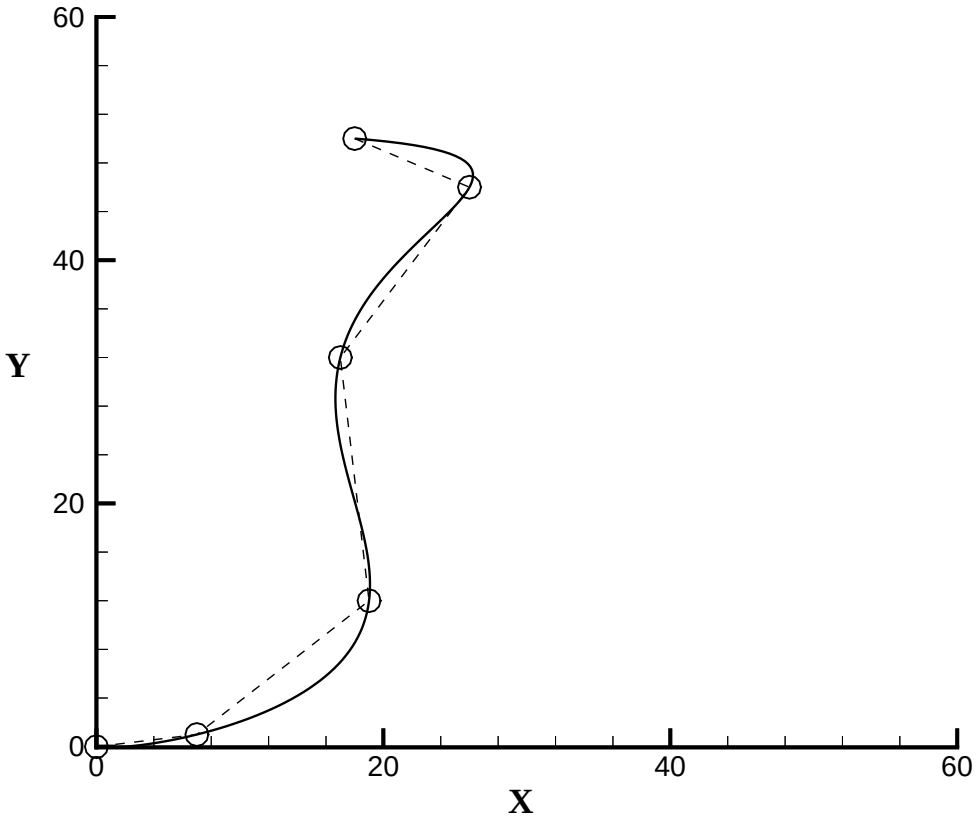


Figure 21: This is the same data as in the preceding figure, except that the last point has been moved in to show off the fact that the curve is very flexible!

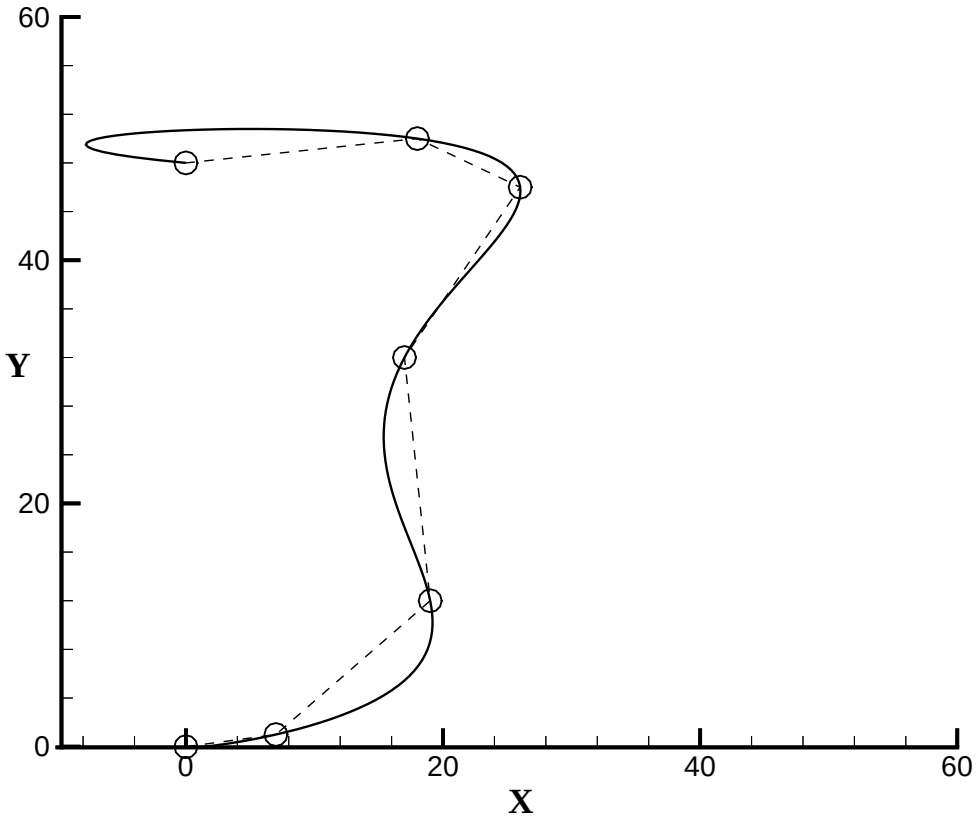


Figure 22: This is an even more extreme case where a seventh point has been added. Note the extra loop at the top, which may not be what the designer intended.

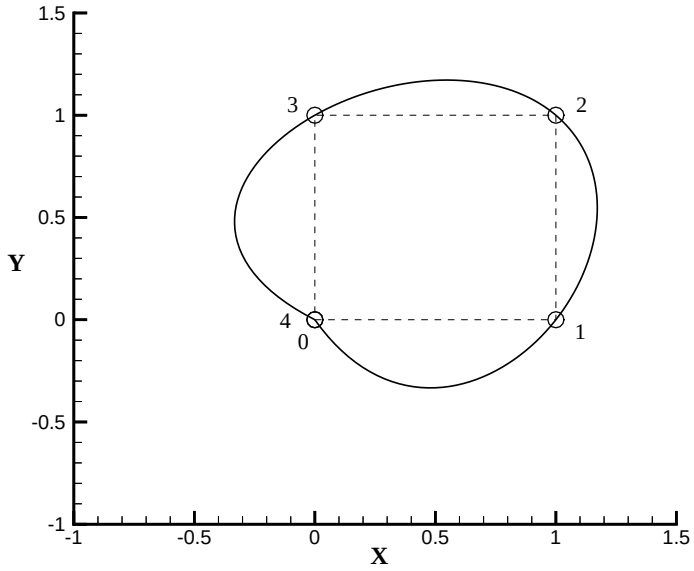


Figure 23: Fourth degree parametric polynomial passing through the four corners of a square. Since the first and last points are really distinct, there are five base points. The result is a perfect apple!

does not come up if we want to display the curve on a video screen or plotter, since the curve is automatically traced out as we evaluate x and y at a large number of values of t. Also, if the curve is multi-valued, the value of y at a given x is ambiguous.

If we have to find y at a given x, a simple way is to form a table at a large number of values of t (which we probably want for graphing anyway), which can then be searched for a pair of values of t which bracket the desired value of x. The value of t can then be found by linear interpolation, as can the corresponding value of y. If this is not accurate enough, this value of t can be used as starting value, an iterative procedure used to find a more exact value of t.

The final example is a parametric polynomial passing through the four corners of a square, as shown in figure 23.

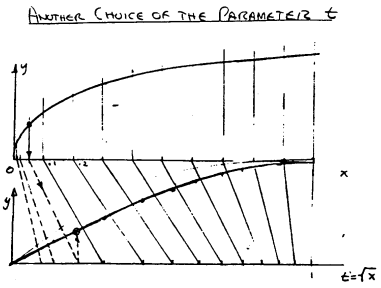


Figure 24: Sketch showing the leading edge of an airfoil in physical coordinates $y(x)$ and in transformed coordinates $y(t)$.

Another choice of the parameter t which works well for airfoil (or sailboat keel) leading edges is $t = \sqrt{(}x)$. For this to work, the origin of the coordinate system must be at the exact leading edge. The parametric representation of the curve is then

$$x(t) = t^2 \qquad y(t) = \sum_{n=0}^{N-1} a_n t^n \tag{21}$$

If the leading edge is round, the slope $\frac{dy}{dx}$ is infinite there, while $\frac{dy}{dt}$ must be finite since it is a polynomial. However, the two are related as follows

$$\frac{dy}{dx} = \frac{dy}{dt}\frac{dt}{dx} = \frac{1}{2}\frac{\frac{dy}{dt}}{\sqrt{x}} \tag{22}$$

Thus we see that as x approaches zero, $\frac{dy}{dx}$ goes to infinity while $\frac{dy}{dt}$ remains finite. This is illustrated in figure 24.

We can, with a little patience, derive a relationship between the leading edge radius, ρ_L, and the slope of the curve at the origin in parametric coordinates, dy/dt. $\rho_L - \frac{dy}{dt}$ relation

The equation of the leading edge circle is

$$(x - \rho_L)^2 + y^2 = \rho_L^2 \tag{23}$$

Solving for y,

$$\begin{aligned} x^2 - 2x\rho_L + \rho_L^2 + y^2 &= \rho_L^2 \\ y &= \pm\sqrt{2x\rho_L - x^2} \end{aligned} \tag{24}$$

Taking the upper branch, and looking at the limit as $x \ll \rho_L$,

$$y \approx \sqrt{2x\rho_L} \tag{25}$$

and the derivative near $x = 0$ is

$$\frac{dy}{dx} = \sqrt{\frac{\rho_L}{2}} \left(\frac{1}{\sqrt{x}} \right) \tag{26}$$

Substituting $t = \sqrt{x}$, and making use of equation 22,

$$\begin{aligned} \frac{dy}{dt} &\approx 2\sqrt{x}\frac{dy}{dx} \\ \frac{dy}{dt} &\approx \sqrt{2\rho_L} \\ \rho_L &\approx \frac{1}{2}\left(\frac{dy}{dt}\right)^2 \end{aligned} \tag{27}$$

At $x = 0$, this is the exact slope of the curve. This is useful information, since we will see later when we discuss spline curves that specifying the slope of a curve at the ends greatly improves the robustness of the fitting process.

Figure 25 gives a quantitative example of this procedure.

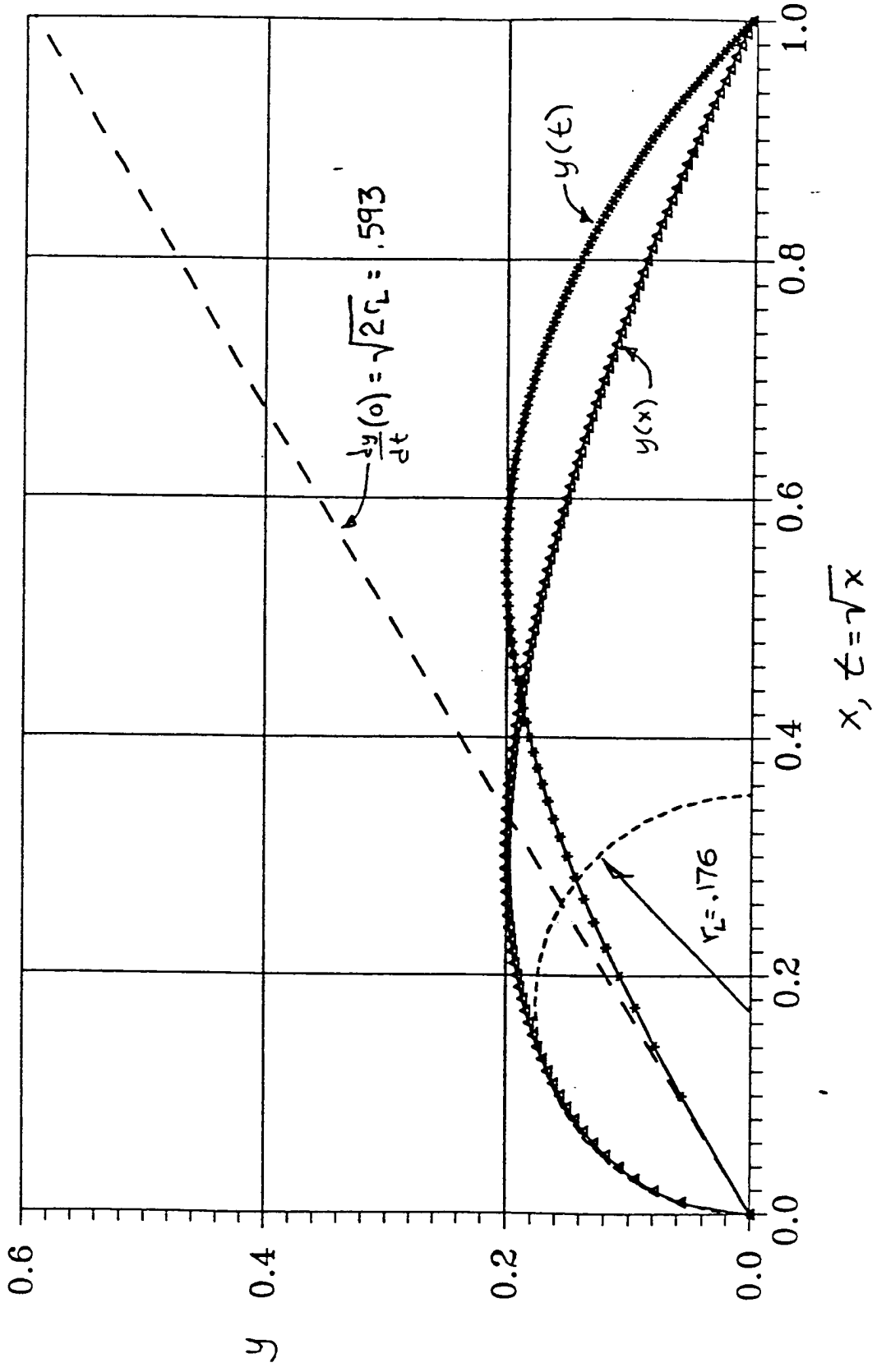


Figure 25: Example of a round-nosed foil section with leading edge radius $\rho_L = 0.176$ and a chord length of 1.0. The slope of the transformed curve clearly matches the value given in the equation above.

4 CUBIC SPLINES

We mentioned earlier in discussing piecewise approximations to curves that one could put together segments of polynomials whose values matched at their boundaries, but in addition, whose slopes and curvatures matched. This type of curve is called a *cubic spline*, and this has turned out to be a real winner. In this section, we will therefore develop the details of the numerical method used to obtain this type of curve.

The physical motivation for a cubic spline is that it is the mathematical representation of the shape of an elastic beam supported by a series of point loads. As indicated in the introduction, slender wooden beams, called *splines*, were used in shipbuilding to produce smooth curves. Figure 26 shows a full-scale waterline being generated in this way. In this case the point loads are provided by nails driven into the mold loft floor (not to be recommended in your living room). A less destructive way of providing the loads to the beam, which is practical when generating small scale drawings, is to use lead weights with small protrusions designed to rest on the spline. These are called *ducks* because some near-sighted person must have thought they looked like one. Figure 27 shows a smooth curve being generated by five ducks. In this case the spline is not wooden, but is made of plastic with a groove along its length to facilitate anchoring the ducks. This represents the high-tech culmination of the era of splines and ducks.

To show that the deflected shape of an elastic beam is comprised of cubic segments, we will look at the simple case of a beam supported by point loads at each end, $x = \pm l/2$, and subjected to a point load W in the middle, at $x = 0$. The governing equation for a slender beam, assuming small deflections from the x axis is

$$M(x) = EI\frac{d^2y}{dx^2} \tag{28}$$

where $M(x)$ is the bending moment at point x along the beam, E is the modulus of elasticity of the material [10] and I is the moment of inertia of the cross section of the beam. In this simple example, vertical equilibrium tells us that the two end reactions must each be $-W/2$, and moment equilibrium about each end of the beam tells us that the bending moment is a piecewise linear function,

$$\begin{aligned} M(x) &= -\frac{W}{2}\left(\frac{l}{2} - x\right) \quad 0 \le x \le l/2 \\ M(x) &= -\frac{W}{2}\left(\frac{l}{2} + x\right) \quad -l/2 \le x\ 0 \end{aligned} \tag{29}$$

[10] Steel: big number, Rubber: small number

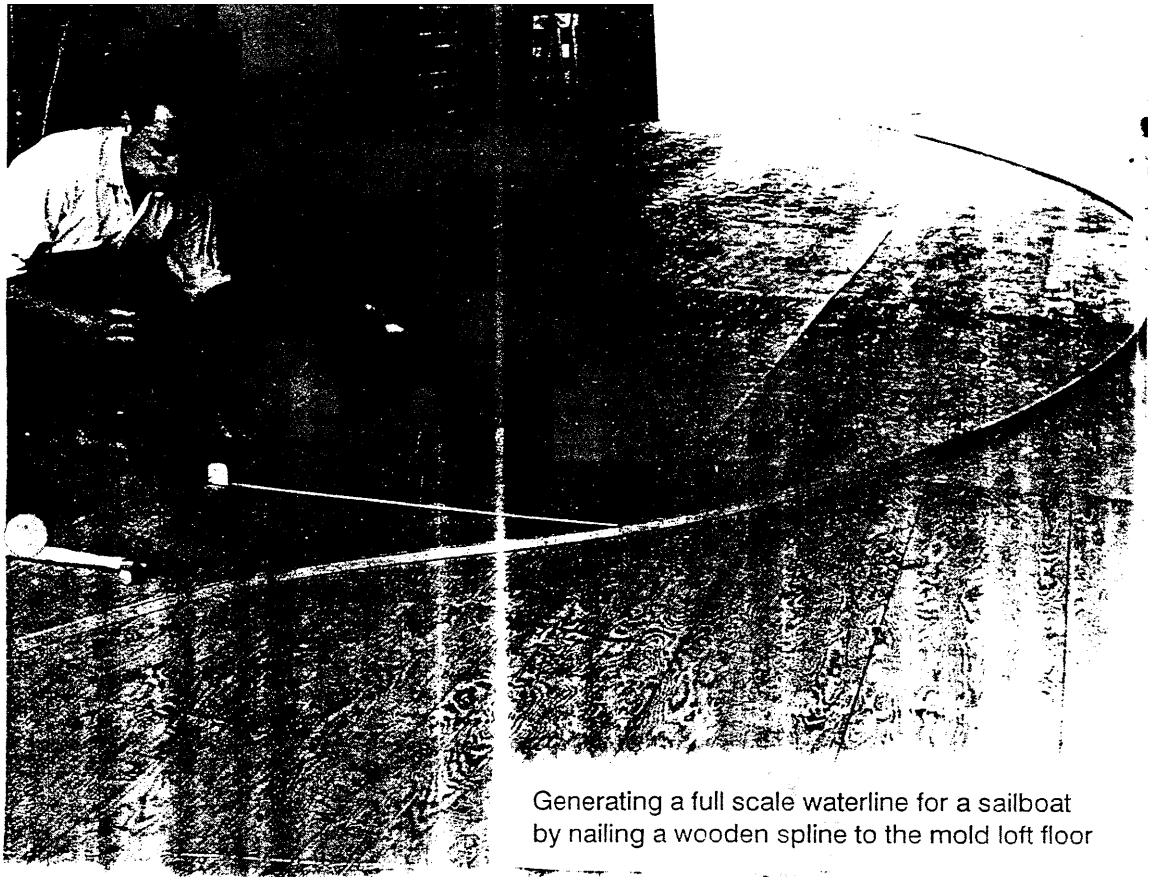


Figure 26: Generating a full scale waterline for a sailboat by nailing a wooden spline to the mold loft floor.

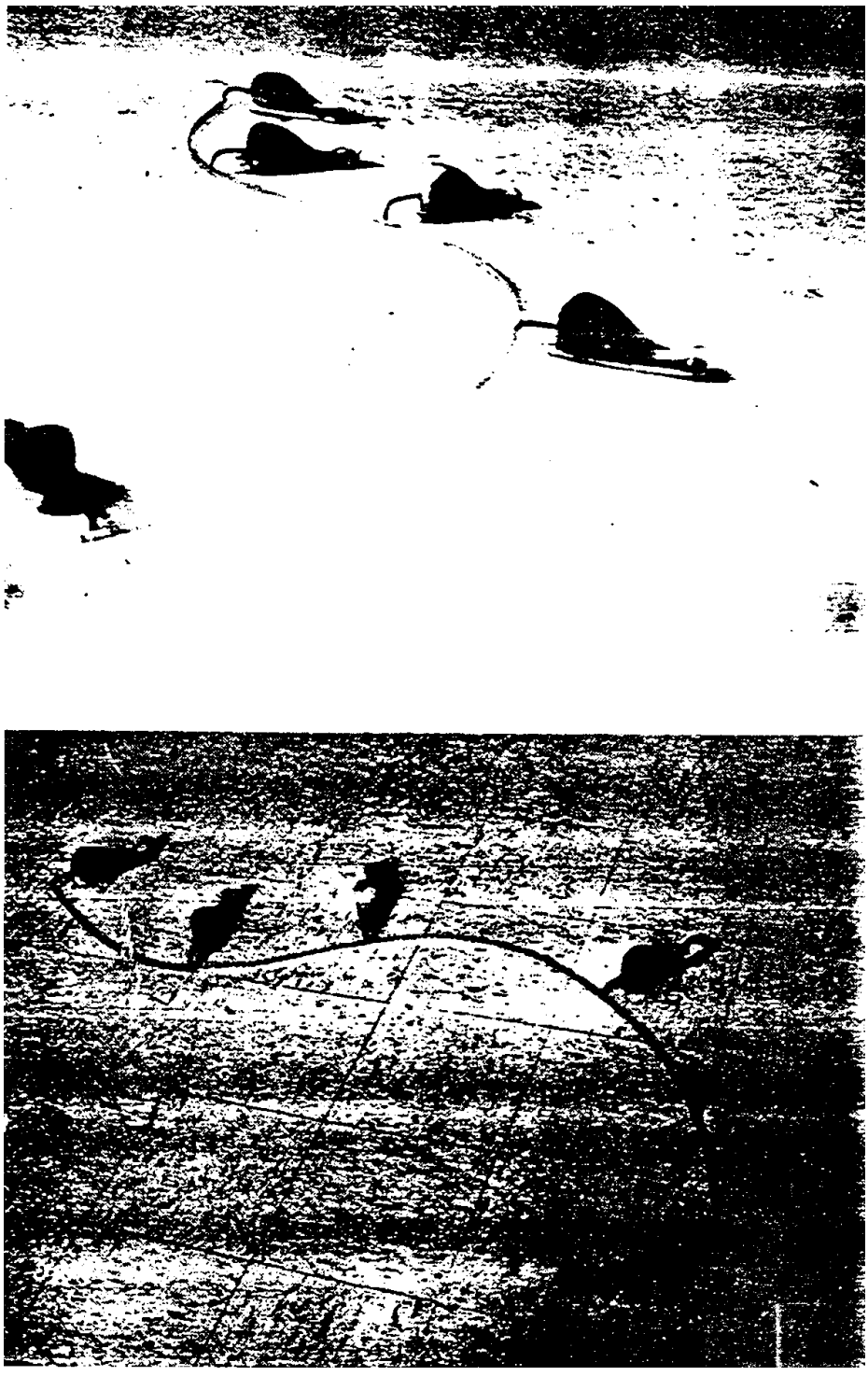


Figure 27: Generating a smooth curve using a plastic spline with lead ducks.

Combining equations 28 and 29 and integrating twice, we obtain the shape of the beam for $0 \leq x \leq l/2$,

$$\begin{aligned} \frac{d^2y}{dx^2} &= \frac{-W}{2EI}\left(\frac{l}{2} - x\right) \\ \frac{dy}{dx} &= \frac{-W}{2EI}\left(\frac{lx}{2} - \frac{x^2}{2} + C_1\right) \\ y &= \frac{-W}{2EI}\left(\frac{lx^2}{4} - \frac{x^3}{6} + C_1x + C_2\right) \end{aligned} \tag{30}$$

Since the slope at $x = 0$ must be zero, from symmetry, the first constant of integration, C_1 must be zero. We can solve for the second constant of integration by imposing the condition that $y(l/2) = 0$. This gives the result that $C_2 = -l^3/24$. The final equation for the shape is therefore

$$y = \frac{W}{2EI}\left(\frac{x^3}{6} - \frac{lx^2}{4} + \frac{l^3}{24}\right) \tag{31}$$

and the maximum deflection of the beam, which is at $x = 0$, is

$$y(0) = \frac{Wl^3}{48EI} \tag{32}$$

If we repeat the preceding steps for the interval $-l/2 \leq x \leq 0$ we obtain the identical expression for $y(x)$, except for one sign change,

$$y = \frac{W}{2EI}\left(-\frac{x^3}{6} - \frac{lx^2}{4} + \frac{l^3}{24}\right) \tag{33}$$

The complete shape of the beam is therefore given by two cubic segments which join at $x = 0$. This is shown in figure 28. It is clear both from the equations of the cubics and from the graph that the two segments have matching value, slope and second derivative at the point where they join. However, the third derivative, which is the rate of change of bending moment with x is discontinuous at $x = 0$. This is a direct consequence of the point load which is applied there.

We will now begin the development of the equations for a general cubic spline curve. We can abandon the elastic beam analogy at this point, since we are really not interested in the relationship between the elastic properties of the beam and the reaction forces at the ducks. What we want is the equations for the set of cubics which pass through a given set of base points, with continuous slope and curvature. The notation is shown in figure 29

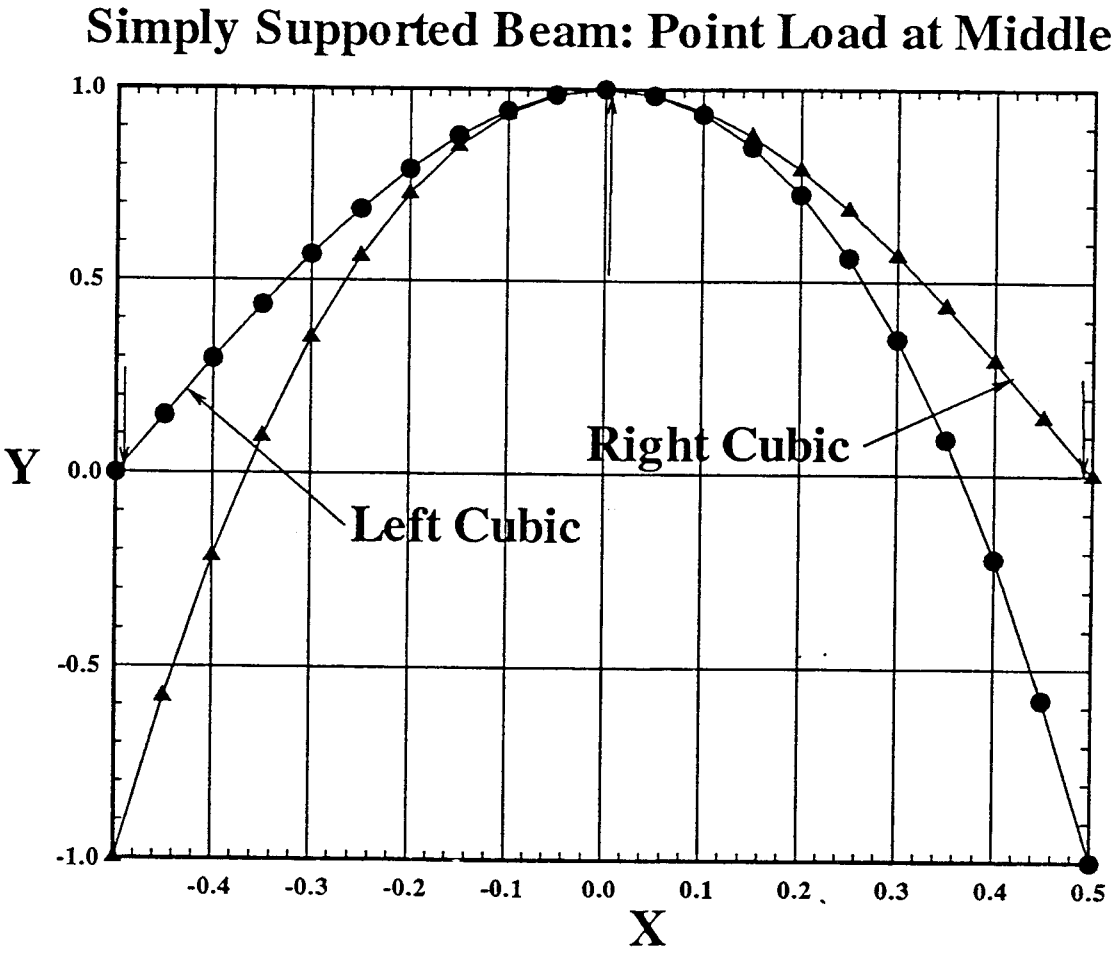


Figure 28: This illustrates how the shape of a deflected beam is made up from two cubic segments.

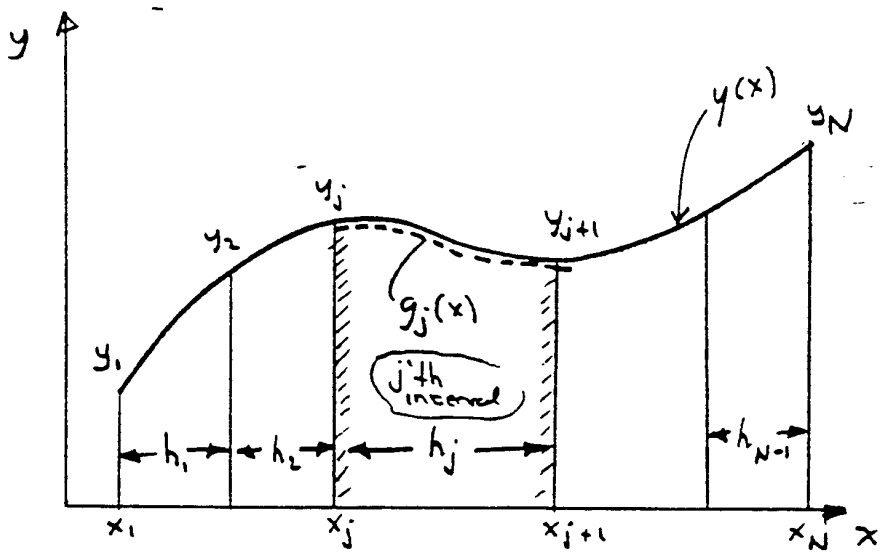


Figure 29: Notation for a cubic spline

Assume that the function $y(x)$ is known at N base points $(x_0, y_0), \ldots, (x_{N-1}, y_{N-1})$. There are $N-1$ intervals with spacing $h_j = x_{j+1} - x_j$. In each interval, the function is to be represented by a cubic equation $g_j(x)$ written in the form:

$$g_j(x) = a_j(x - x_j)^3 + b_j(x - x_j)^2 + c_j(x - x_j) + d_j \qquad for\ x_j \le x \le x_j + 1 \quad (34)$$

The coefficients $a_j, \ldots, d_j$ are to be determined from the following conditions.

The cubics pass through the base points at the ends of each interval

$$g_j(x_j) = y_j \quad (35)$$

$$g_j(x_{j+1}) = y_{j+1} \quad (36)$$

The slopes are continuous at the boundaries of each interval

$$\frac{dg_j}{dx}(x_j) = \frac{dg_{j-1}}{dx}(x_j) \quad (37)$$

$$\frac{dg_j}{dx}(x_{j+1}) = \frac{dg_{j+1}}{dx}(x_{j+1}) \quad (38)$$

The second derivatives are continuous at the boundaries of each interval

$$\frac{d^2 g_j}{dx^2}(x_j) = \frac{d^2 g_{j-1}}{dx^2}(x_j) \tag{39}$$

$$\frac{d^2 g_j}{dx^2}(x_{j+1}) = \frac{d^2 g_{j+1}}{dx^2}(x_{j+1}) \tag{40}$$

This looks like six equations for four unknowns. However, the conditions given in equations 37 to 40 are shared by two adjacent cubics, and we will see subsequently that the problem is properly specified, except at the first and last point, where we will need to specify one extra condition.

We can find d_j immediately from equation 34

$$d_j = y_j \tag{41}$$

Also by setting $x = x_{j+1}$ and using the definition of the interval h_j we obtain

$$y_{j+1} = a_j h_j^3 + b_j h_j^2 + c_j h_j + d_j \tag{42}$$

In order to apply equations 37-40 we need the first and second derivatives

$$\frac{dg_j}{dx} = 3a_j(x - x_j)^2 + 2b_j(x - x_j) + c_j \tag{43}$$

$$\frac{d^2 g_j}{dx^2} = 6a_j(x - x_j) + 2b_j \tag{44}$$

An efficient way to develop the necessary equations for the unknown cubic coefficients is given by C.F. Gerald, *"Applied Numerical Analysis"*, 1970. The principal unknowns are taken to be the second derivatives at the base points, which are defined by the symbol S_j,

$$S_j \equiv \frac{d^2 g_j}{dx^2}(x_j) \tag{45}$$

When $x = x_j$ we see from 44 that

$$b_j = \frac{S_j}{2} \tag{46}$$

while at the other end of the interval

$$S_{j+1} = 6a_j h_j + 2b_j$$

so that

$$a_j = \frac{S_{j+1} - S_j}{6h_j} \tag{47}$$

We now have a_j, b_j, d_j in terms of either S_j or y_j. It looks as though we are almost done, but we must still obtain an expression for c_j . First, we can solve 42 for c_j and express a_j, b_j and d_j in terms of S_j

$$\begin{aligned} c_j &= \frac{y_{j+1} - a_j h_j^3 - b_j h_j^2 - d_j}{h_j} \\ &= \frac{y_{j+1} - y_j}{h_j} - \frac{2h_j S_j + h_j S_{j+1}}{6} \end{aligned} \tag{48}$$

The next step is to write down two expressions for the slope at x_j — the first one being the slope at the left end of the j'th interval, and the second one being the slope at the right end of the (j-1)'st interval. These are the same point on the curve, and we require that the slope be continuous.

$$\frac{dg_j}{dx}(x_j) = c_j \tag{49}$$

$$\frac{dg_{j-1}}{dx}(x_j) = 3a_{j-1}h_{j-1}^2 + 2b_{j-1}h_{j-1} + c_{j-1} \tag{50}$$

Combining 48, 49 and 50

$$\begin{aligned} \frac{y_{j+1} - y_j}{h_j} - \frac{2h_j S_j + h_j S_{j+1}}{6} &= 3\big(\frac{S_j - S_{j-1}}{6h_{j-1}}\big)h_{j-1}^2 + 2\frac{S_{j-1}}{2}h_{j-1} \\ &\quad + \frac{y_j - y_{j-1}}{h_{j-1}} - \frac{2h_{j-1}S_{j-1} + h_j S_j}{6} \end{aligned}$$

This can be simplified as follows,

$$h_{j-1}S_{j-1} + 2\big(h_{j-1} + h_j\big)S_j + h_j S_{j+1} = 6\big(\frac{y_{j+1} - y_j}{h_j} - \frac{y_j - y_{j-1}}{h_{j-1}}\big) \tag{51}$$

The last two terms in 51 can de recognized as first divided differences which we have seen before in Newton's method. Defining

$$D_j \equiv \frac{y_{j+1} - y_j}{h_j}$$

Equation 51 finally becomes

$$h_{j-1}S_{j-1} + 2\big(h_{j-1} + h_j\big)S_j + h_jS_{j+1} = 6\big(D_j - D_{j-1}\big)$$
$$for\ j = 1, 2, \ldots, N-2 \tag{52}$$

This equation may be used to form a set of simultaneous equations for the unknown values of S_j. For example, suppose that we have 5 base points, so that there are three interior boundaries. Writing out equation 52 with $j = 1$, $j = 2$, and $j = 3$ produces the following set of equations,

$$\left|\begin{array}{lllllll} h_0S_0 & 2(h_0+h_1)S_1 & h_1S_2 & 0 & 0 & = & 6(D_1 - D_0) \\ 0 & h_1S_1 & 2(h_1+h_2)S_2 & h_2S_3 & 0 & = & 6(D_2 - D_1) \\ 0 & 0 & h_2S_2 & 2(h_2+h_3)S_3 & h_3S_4 & = & 6(D_3 - D_2) \end{array}\right| \tag{53}$$

We now have 3 equations for 5 unknowns, so that it is clear that we need two more equations which must come from the end conditions. The simplest end condition is to specify the second derivative at the two ends. A special case of this would be to specify that the second derivatives at the ends are zero. This is equivalent to the case of an actual spline supported only by ducks, since the point loads at the ends cannot produce a bending moment there.

In any event, if we specify the end second derivatives, we see from equation 45 that S_0 and S_{N-1} are no longer unknowns. We could therefore move them to the right hand side of the set of simultaneous equations given in 53, with the result that the number of equations is now equal to the number of unknowns. However, in order to provide greater flexibility in specifying end conditions, we will, instead, leave S_0 and S_{N-1} as unknowns, and add two equations which simply state that these are equal to the specified values. In an actual computer code, it would be more efficient to use the first approach, but right now we are more interseted in being systematic than in being computationally efficient. The augmented set of equations for the case of five points with specified end second derivatives would then look as follows,

$$\left|\begin{array}{lllllll} S_0 & 0 & 0 & 0 & 0 & = & \frac{d^2y}{dx^2}(x_0) \\ h_0S_0 & 2(h_0+h_1)S_1 & h_1S_2 & 0 & 0 & = & 6(D_1 - D_0) \\ 0 & h_1S_1 & 2(h_1+h_2)S_2 & h_2S_3 & 0 & = & 6(D_2 - D_1) \\ 0 & 0 & h_2S_2 & 2(h_2+h_3)S_3 & h_3S_4 & = & 6(D_3 - D_2) \\ 0 & 0 & 0 & 0 & S_4 & = & \frac{d^2y}{dx^2}(x_4) \end{array}\right| \tag{54}$$

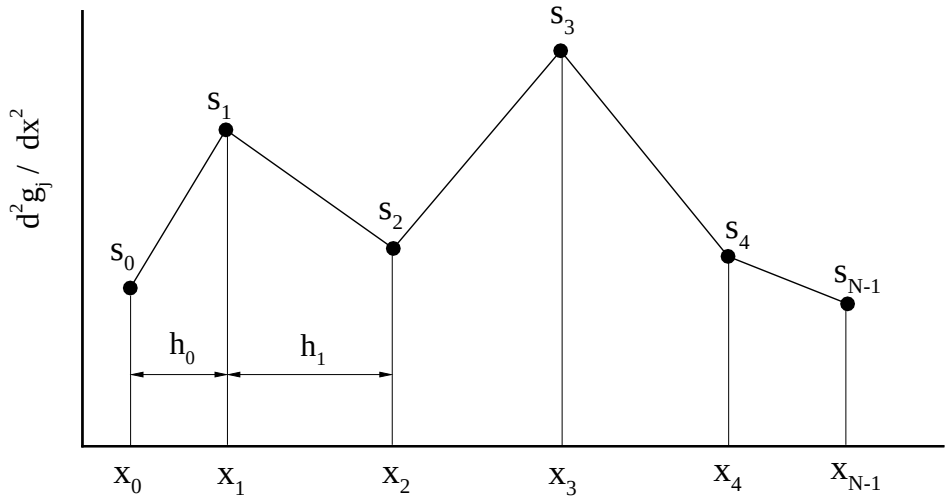


Figure 30: Piecewise linear second derivative of a cubic spline

The next step is to solve the set of simultaneous equations given in 54, which can be done easily by calling any general purpose simultaneous equations solver which most computer installations have available. However, if you stand back and look at the left hand side of the simultaneous equations shown in 54, you can see that all of the elements are zero except in three diagonal bands consisting of the principal diagonal, one above and one below. This, logically enough, is called a tri-diagonal matrix, which can be solved much more efficiently. In addition, if we know in advance that it is tri-diagonal, we do not need to save computer storage for N^2 elements, but only for $3N$ elements. We will not go into the details of how this is done here, but if you ever come across a cubic spline computer code which does not take advantage of the tri-diagonal feature of the simultaneous equations, you can temporarily become obnoxious, and tell all your friends how it *should* have been done.

In any event, we now know the second derivative of the spline at the base points, but since it is a piecewise linear function, we now know it everywhere as sketched figure 30

Once we have the values of S_j, it is easy to obtain the four cubic coefficients in each interval. The first coefficient a_j is obtained from 47, b_j from 46, c_j from 48 and d_j from 41. With the cubic coefficients determined in each interval, the value of $y(x)$ can be obtained by first finding which interval y is in, and then evaluating the cubic from the coefficients corresponding to that interval. Once the preliminary work of finding the cubic coefficients has been done, succesive evaluations of the cubic spline are very fast.

In some applications, specification of the end slopes may be desired. If the slope at the left end is prescribed, we can write down from 48

$$D_0 - \frac{2h_0S_0}{6} - \frac{h_0S_1}{6} = \frac{dy}{dx}(x_0) \tag{55}$$

so that this gives us another equation for S_0 and S_1. The equation for the slope at the right end is similar. The complete set of simultaneous equations for four points with slopes prescribed at each end would then be as follows,

$$\left|\begin{array}{llll} 2h_0S_0 & h_0S_1 & 0 & 0 \\ h_0S_0 & 2(h_0+h_1)S_1 & h_1S_2 & 0 \\ 0 & h_0S_1 & 2(h_1+h_2)S_2 & h_2S_3 \\ 0 & 0 & -h_2S_2 & -2h_2S_3 \end{array}\right| \begin{array}{l} = 6(D_0 - \frac{dy}{dx}(x_0)) \\ = 6(D_1 - D_0) \\ = 6(D_2 - D_1) \\ = 6(D_2 - \frac{dy}{dx}(x_3)) \end{array} \tag{56}$$

Specifying two end slopes or two end second derivatives are just two examples of the many possibilities available. We could also specify a slope and a second derivative at one end, or a slope at one end and a second derivative at the other end. We therefore always start with the basic set of interior equations given in 53 and add any two end equations.

One final type of end condition will be discussed here. There are times when we have no idea what the end slope or end second derivative should be. An easy way out might be to specify zero for the second derivatives, but this might be a poor choice for a curve which resembles a circular arc with constant curvature. Another alternative is to assume that the second derivative at the ends is a linear extrapolation of the values at the two adjacent base points, as shown in the sketch below,

At the left end,

$$\frac{S_2 - S_1}{h_1} = \frac{S_1 - S_0}{h_0} \tag{57}$$

which then gives us another equation relating the first three values of S,

$$h_1S_0 - (h_0+h_1)S_1 + h_0S_2 = 0 \tag{58}$$

which could be used in place of one of the other end conditions.

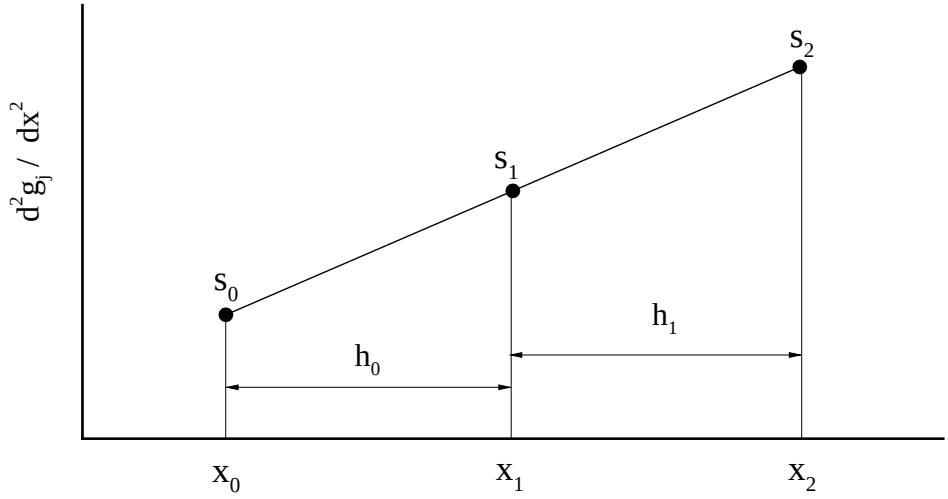
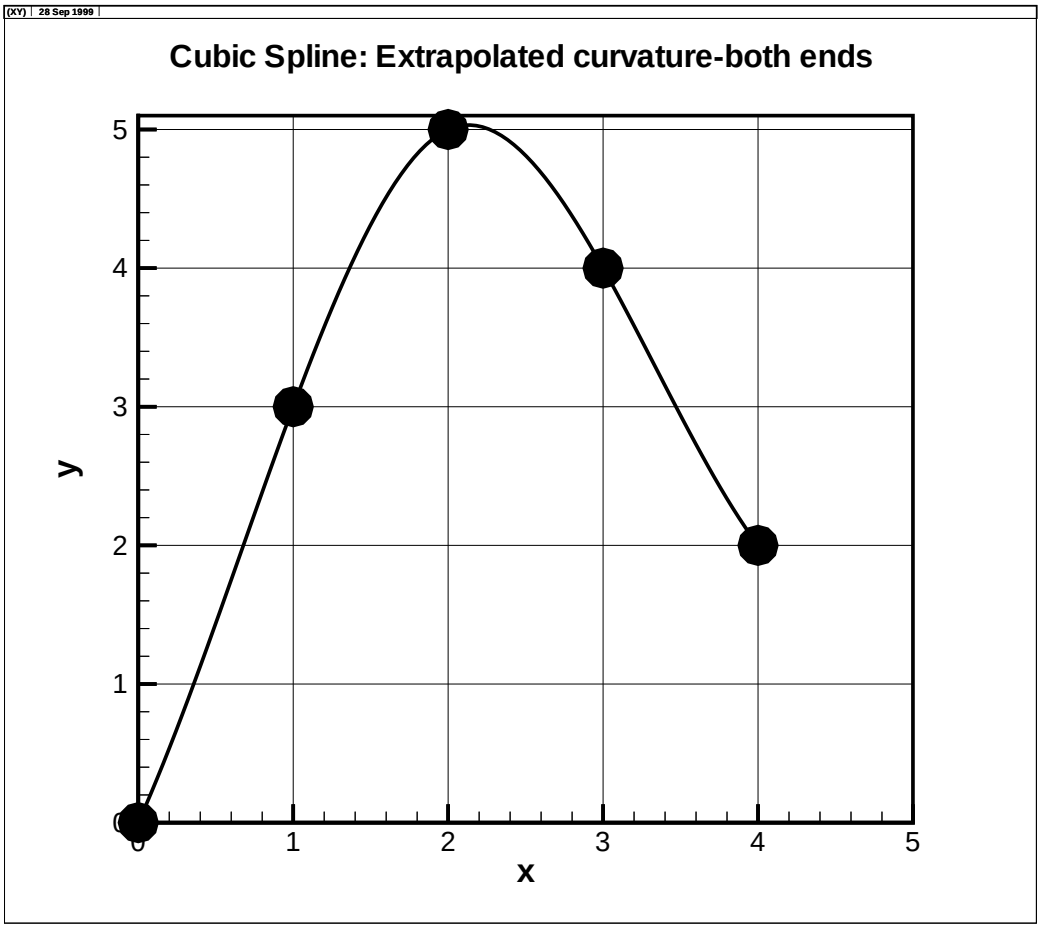
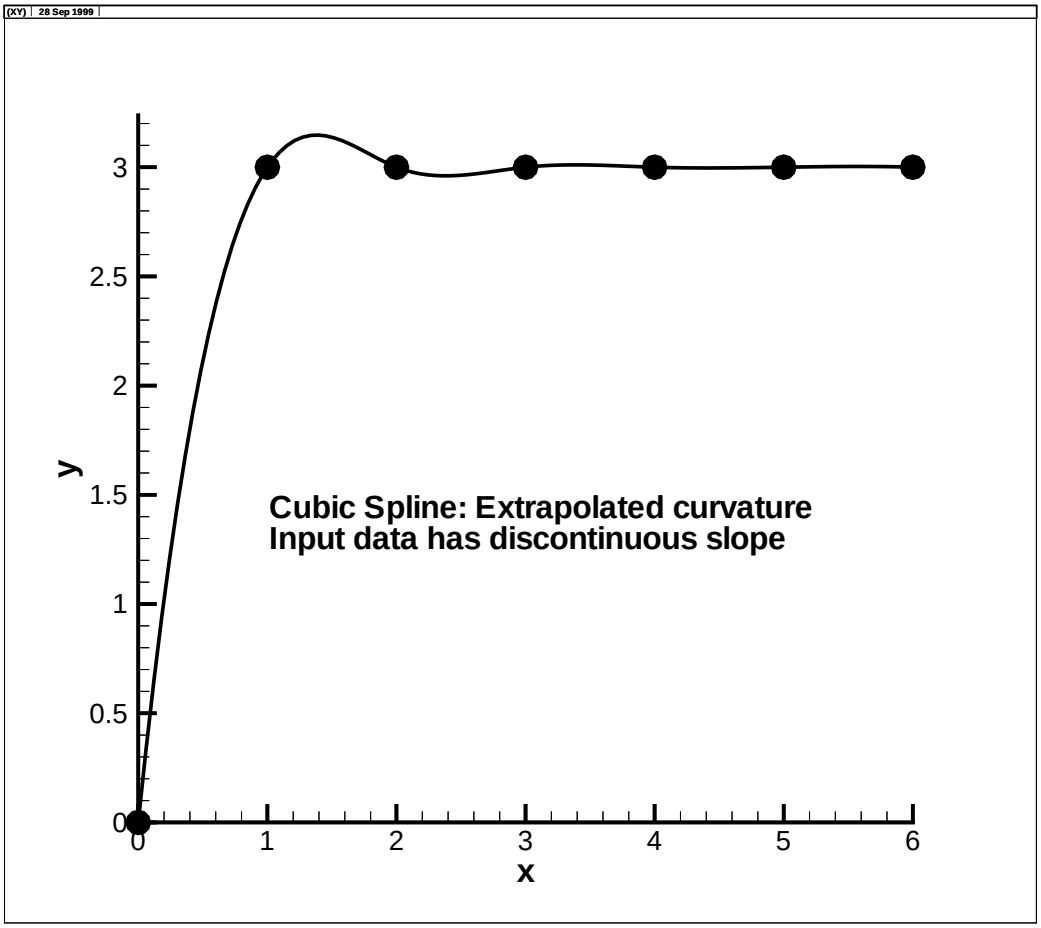


Figure 31: Extrapolated end slope condition

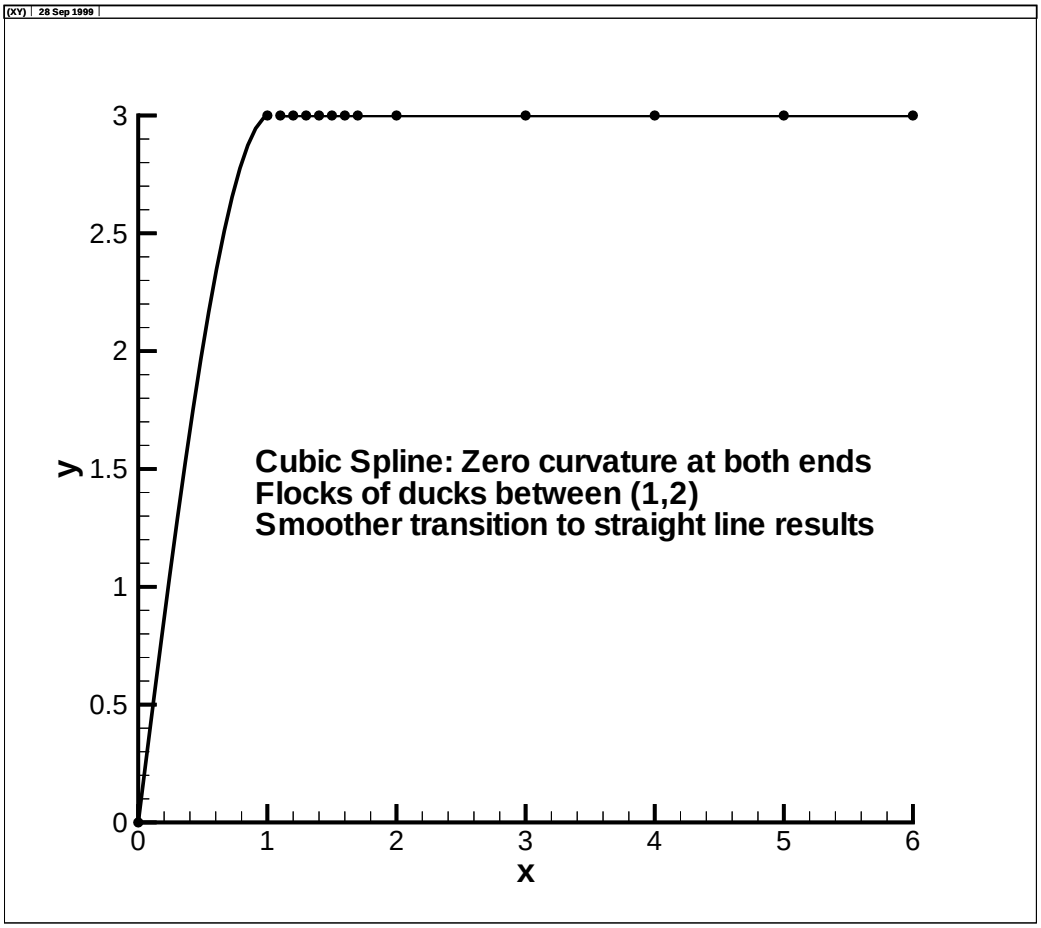
Cubic Spline Examples



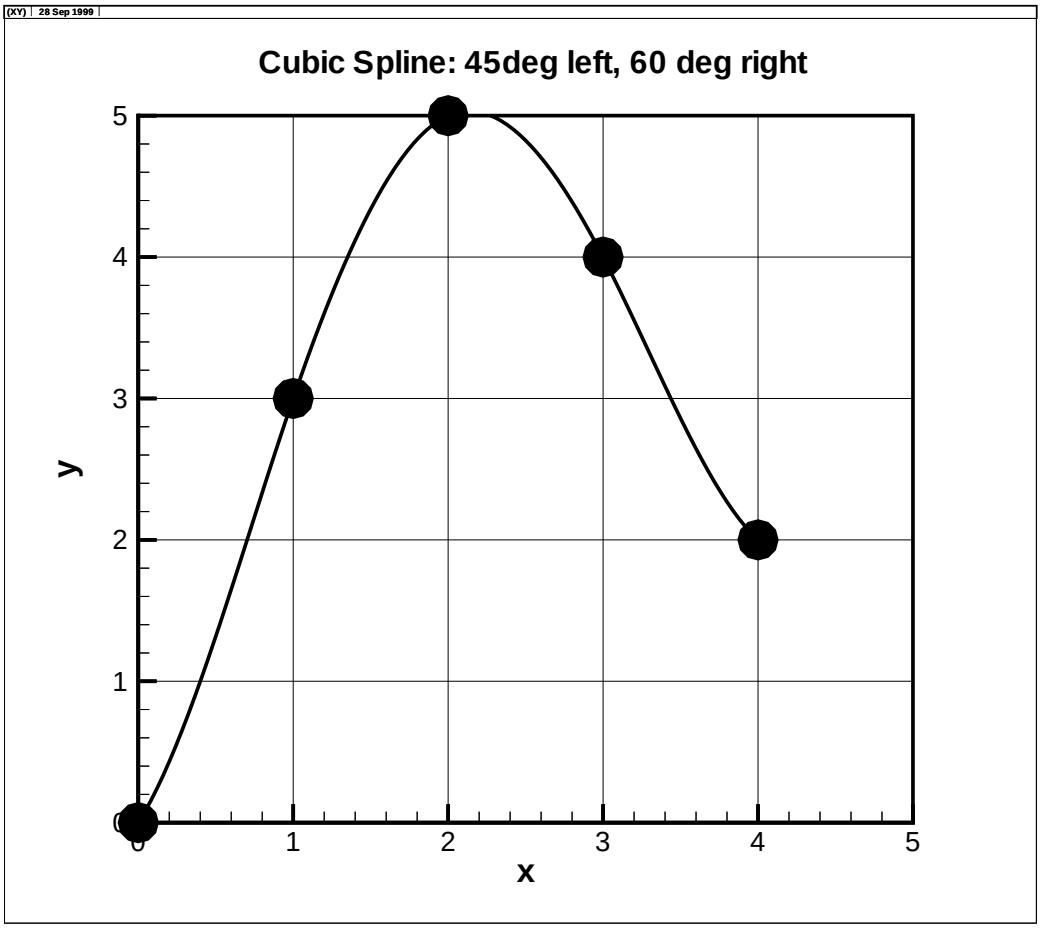
Cubic Spline Examples (continued)



Cubic Spline Examples (continued)



Example—Use of End Slope Condition



5 B-SPLINES

5.1 Introduction

The representation of curves and surfaces by B-splines has become widespread in the computer-aided design community in recent years. There are two reasons for this. First of all, an extremely wide variety of geometries can be represented by an elegantly unified class of functions. Second, the shape of B-spline curves and surfaces can be controlled in a very intuitive way in a computer graphical environment. In fact, the latter attribute is such that a CAD system user can generate and control the shape of objects with little or no knowledge of their mathematical basis.

We have already seen how one can fit sets of data points with polynomials. The result is a set of coefficients which define the curve. However, except in very simple cases, an examination of a list of polynomial coefficients doesn't provide us with much of a feeling of what the curve is going to look like. More often than not, the coefficients will vary substantially in magnitude and frequently alternate in sign. If we were to try to design a curve by playing with, say, ten polynomial coefficients, we would probably give up very quickly.

On the other hand, the "coefficients" in a B-spline representation of a curve have a geometrical significance. They form a *control polygon* whose vertices may individually be adjusted to alter the shape of the curve. As the polygon is adjusted, the shape of the curve changes in a very intuitive and predictable way.

B-splines may also be used to define surfaces, and the control polygon in this case becomes a *control net*. Again, one can pull on one or more of the vertices of the control net to adjust the shape of the surface.

A rigorous development of the theory of B-splines is (fortunately) beyond the scope of this subject. However, most of the mathematics of B-splines can be developed in a simple-minded way by starting with some elementary cases and then proceeding onward with blind faith. Fortunately, the nice thing about geometric modeling is that if it looks right it is right— provides you look close enough.. Of course, the converse is also true. So, here we go!

5.2 Quadratic Bézier Curves

We will sneak up on B-splines gradually by starting with a class of curves developed by Bézier [11]. As we will soon see, B-splines were invented later as a generalization of

[11] Bézier is an applied mathematician who works for the Renault car company. All Renaults are therefore shaped from Bézier curves, except for the wheels, which– if they were– would cause an annoying thump at highway speeds.

Bézier curves, so we are actually looking at a particular type of B-spline. Let us first develop the simplest case, the quadratic Bézier curve.

Consider a two-dimensional curve expressed in a particular parametric form

$$\begin{aligned} x = x(u) &= \sum_{i=0}^{2} \xi_i \mathcal{N}_i^2(u) \\ y = y(u) &= \sum_{i=0}^{2} \eta_i \mathcal{N}_i^2(u) \end{aligned} \tag{59}$$

where u is a parameter which is a monotonically increasing function of arc length along the curve, starting with a value of zero at one end and ending with a value $u = 1.0$. The coordinates x, y along the curve are therefore obtained as weighted sums of a set of *basis functions* $\mathcal{N}_i^2(u)$. In addition, the weights ξ_i, η_i are to be interpreted geometrically as the coordinates of the vertices of a *control polygon*. In this simple case, there are only three vertices, so the control polygon consists of two connected straight line segments.

The basis functions $\mathcal{N}_i^2(u)$ will be polynomials of degree two selected in such a way that

1. The curve starts at (ξ_0, η_0)
2. The curve ends at (ξ_2, η_2)
3. The curve is tangent to the control polygon at (ξ_0, η_0)
4. The curve is tangent to the control polygon at (ξ_2, η_2)

Before going on, let's take a moment to explain the notation for the basis functions. Current literature in the field seems to have settled on the symbol N to denote B-spline basis functions. This seems like a poor choice since it implies an integer constant (particularly for Fortran programmers!), but it makes sense for the 13.016 notes to conform. However, as a mild protest, we will use a calligraphic N ($\mathcal{N}$). The superscript 2 denotes the degree of the basis function (remember, the degree of a polynomial is the order minus one). In the general case, the degree is denoted in the literature by the symbol p, i.e. $\mathcal{N}_i^p$ the basis function of degree p associated with the $i'th$ vertex.

Getting back to the problem at hand, we will next show that the following set of basis functions, which are plotted in figure 32, satisfies the required conditions:

$$\begin{aligned} \mathcal{N}_0^2(u) &= 1 - 2u + u^2 \\ \mathcal{N}_1^2(u) &= 2u - 2u^2 \\ \mathcal{N}_2^2(u) &= u^2 \end{aligned} \tag{60}$$

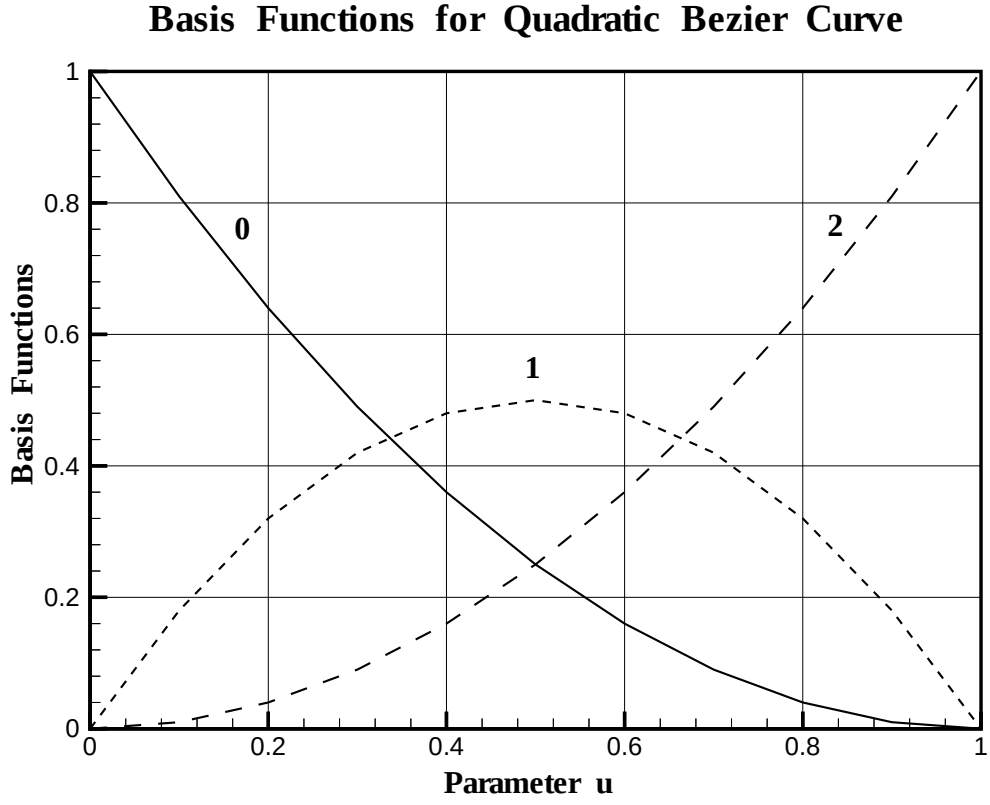


Figure 32: Basis functions $\mathcal{N}_i^2(u)$ for a quadratic Bézier curve

It is easy to see that the curve will start at (ξ_0, η_0). When $u = 0$, $\mathcal{N}_0^2 = 1.0$ and $\mathcal{N}_1^2 = 0,\ \mathcal{N}_2^2 = 0$. Hence, from equation 59 $x(0) = \xi_0,\ y(0) = \eta_0$. Similar conditions obviously apply at the other end. The slope requirement takes a little more work.

$$\frac{dy}{dx} = \frac{\frac{dy}{du}}{\frac{dx}{du}} = \frac{\sum_{i=0}^{2} \eta_i \frac{d}{du}\mathcal{N}_i^2(u)}{\sum_{i=0}^{2} \xi_i \frac{d}{du}\mathcal{N}_i^2(u)} \tag{61}$$

For this particular set of basis functions, we can carry out the differentiation and write out the summation in equation 61 as

$$\frac{dy}{dx} = \frac{\eta_0(-2+2u) + \eta_1(2-4u) + \eta_2(2u)}{\xi_0(-2+2u) + \xi_1(2-4u) + \xi_2(2u)} \tag{62}$$

At the left end, when $u = 0$ we can see from equation 62 that

$$\frac{dy}{dx} = \frac{(\eta_1 - \eta_0)}{(\xi_1 - \xi_0)} \tag{63}$$

while at the right end, when $u = 1.0$,

$$\frac{dy}{dx} = \frac{(\eta_2 - \eta_1)}{(\xi_2 - \xi_1)} \tag{64}$$

So all the required conditions are met. Moreover, the derivative condition can be interpreted graphically, which will be useful in the general case. Note from equation 62 that so long as the slopes of the first two basis functions at $u = 0$ are equal and opposite, and that the slope of the third basis function at $u = 0$ is zero, equation 63 will result. Similarly, the slopes of the second and third basis functions at $u = 1$ must be equal and opposite, while the slope of the first basis function at $u = 1$ must be zero. This is clearly evident from figure 32.

Figure 33 shows an example of a quadratic Bézier curve, together with a sample calculation of one point on the curve. The curve is clearly doing just the right thing. If you imagine that the control polygon is a pair of sticks hinged together at the second vertex, as you change the angle between the two sticks, the curve will follow along, remaining tangent at both ends.

Now, what is the parameter u? We saw earlier that parametric polynomial representation of curves could be constructed using an approximate arc length along the curve as the parameter. But in this case, it is clear from figure 33 that u is not proportional to arc length since the spacing between the symbols on the curve is not constant. Except for unusual cases, *the parameter u along a B-spline curve is not proportional to arc length.* It isn't anything. It just happens.

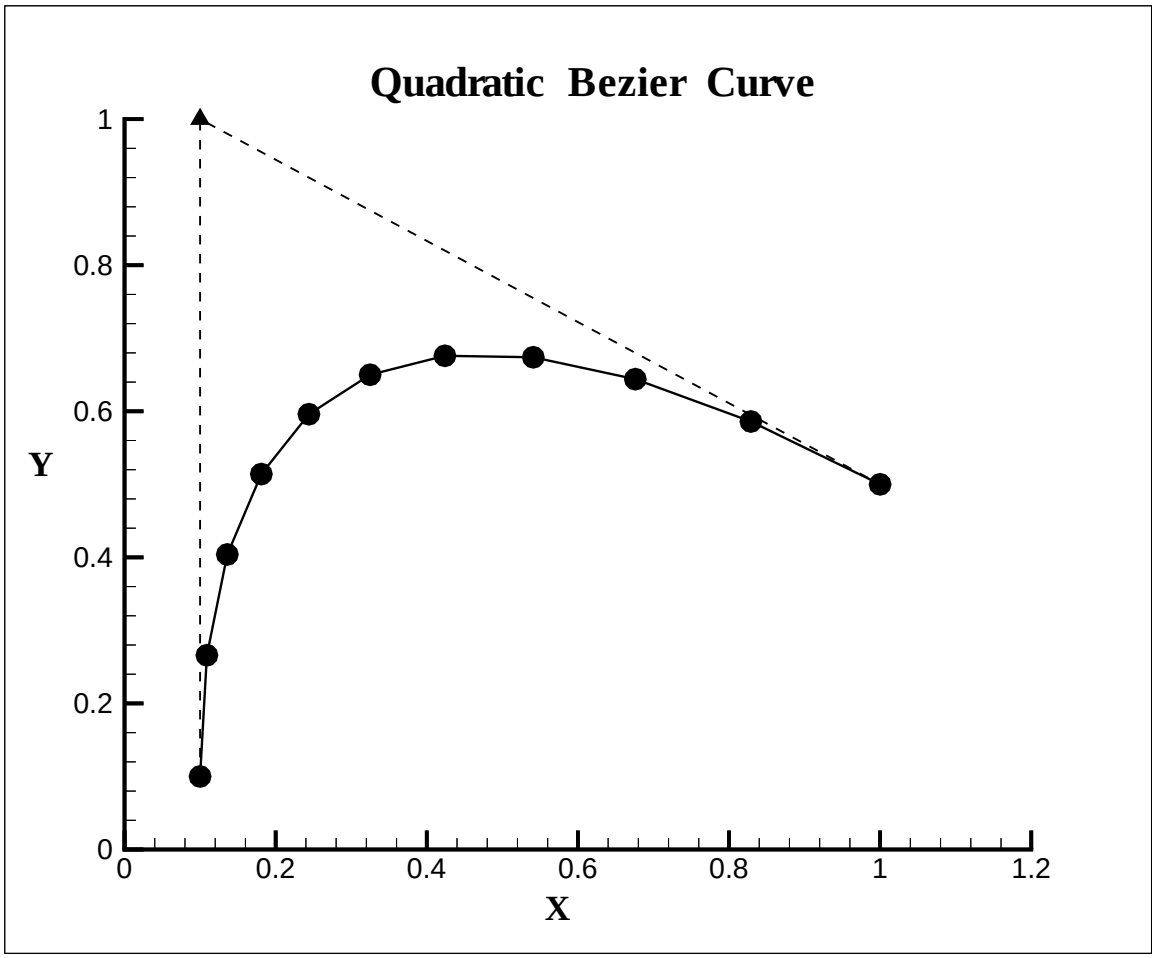


Figure 33: Sample quadratic Bézier curve. The symbols are plotted at evenly spaced values of the parameter u;$(0, (0.1), 1)$.

5.3 Cubic Bézier Curves

The approach followed in the preceding section can be repeated for the case of a cubic Bézier curve. The number of vertices must now be four, and the degree of the basis functions is $p = 3$. With these changes, equation 59 becomes,

$$\begin{aligned} x = x(u) &= \sum_{i=0}^{3} \xi_i \mathcal{N}_i^3(u) \\ y = y(u) &= \sum_{i=0}^{3} \eta_i \mathcal{N}_i^3(u) \end{aligned} \tag{65}$$

We can find a set of four cubics which again meet the conditions that the curve pass through the end points and be tangent to the first and last control polygon segments. However, in this case the solution is not unique, and one additional condition must be imposed. This condition is that the curve must become a straight line in the limit as the polygon degenerates to a straight line. For example, if the polygon becomes a horizontal straight line, $\eta_i = \eta$, and equation 65 becomes

$$y(u) = \sum_{i=0}^{3} \eta_i \mathcal{N}_i^3(u) = \eta \sum_{i=0}^{3} \mathcal{N}_i^3(u) = \eta \tag{66}$$

so that

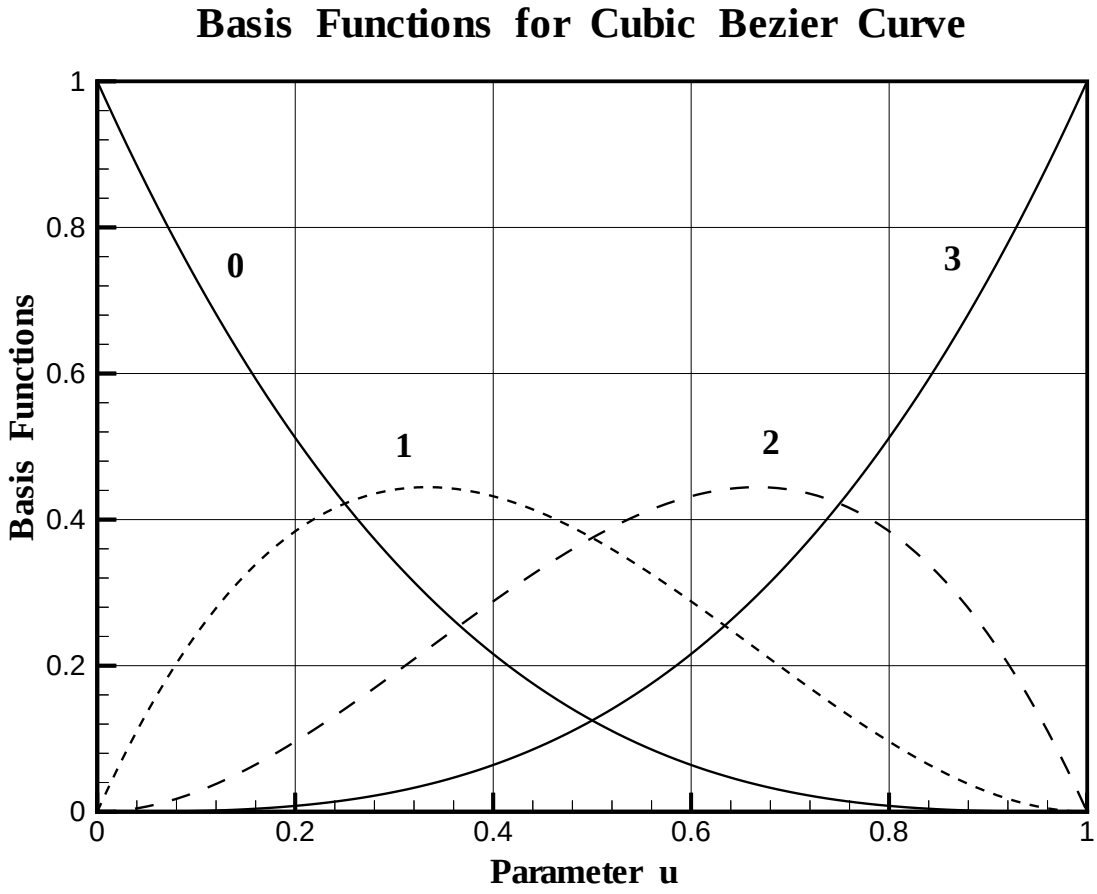
$$\sum_{i=0}^{3} \mathcal{N}_i^3(u) = 1.0 \tag{67}$$

This is similar to the earlier finding that the sum of the Lagrange interpolation polynomials must be one. With this additional condition, a unique set of basis functions can be found and these are plotted in figure 34. The equations for each of the four cubics are given with the figure. One can see that the all basis functions except the first are zero at $u = 0$, that the slopes of the first two basis functions at $u = 0$ are equal and opposite and that the slopes of the remaining two basis functions at $u = 0$ are zero. Corresponding conditions are met at the other end, for $u = 1$. Finally, the sum of all four basis functions at any value of u are equal to one.

A sample cubic Bézier curve is shown in figure 35. The first three vertices are the same as for the quadratic example, but we have added a fourth vertex at $(0.8, 0.2)$. Again, the curve seems to just the right thing. Note that the parameter value along the curve is not proportional to arc length— it changes more rapidly at the right end of the curve than at the left.

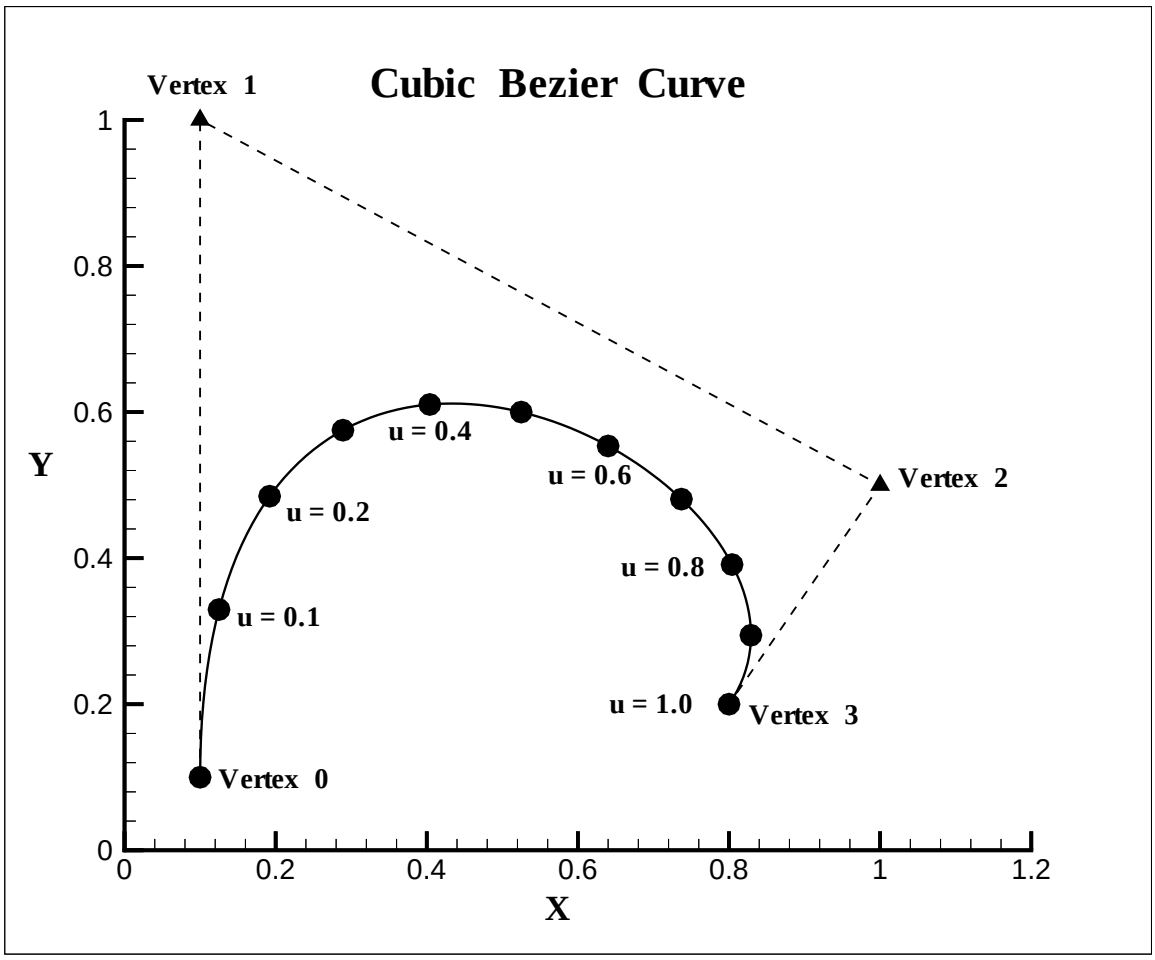
Bézier curves can be extended to any degree by adding more vertices and devising suitable basis functions. As we will show later, a systematic procedure exists for computing basis functions of any degree so that one does not really have to battle with each case by trial and error. However, increasing the degree of the polynomial is not always desirable, so that it would be nice to be able to increase the number of vertices (and hence, represent more complicated curves) without going to higher degree basis functions.

We saw before that this could be accomplished by *piecewise* polynomials, where the cubic spline was a good example. The same idea, applied as an extension to the Bézier curves, led to the development of the B-spline.



$$
\begin{aligned}
\mathcal{N}_0^3 &= (1-u)^3 \\
\mathcal{N}_1^3 &= 3u(1-u)^2 \\
\mathcal{N}_2^3 &= 3u^2(1-u) \\
\mathcal{N}_3^3 &= u^3
\end{aligned}
$$

Figure 34: Basis functions for a cubic Bézier curve.




```
TITLE = "B-SPLINE CURVES"
VARIABLES = X, Y
ZONE T="VERTS", I=4
0.10000    0.10000
0.10000    1.00000
1.00000    0.50000
0.80000    0.20000
```

```
ZONE T="VERTS", I=4
0.10000    0.10000
0.12500    0.32960
0.19200    0.48480
0.28900    0.57520
0.40400    0.61040
0.52500    0.60000
0.64000    0.55360
0.73700    0.48080
0.80400    0.39120
0.82900    0.29440
0.80000    0.20000
```

Figure 35: Sample cubic Bézier curve. The symbols are plotted at evenly spaced values of the parameter u;$(0, (0.1), 1)$.

5.4 Quadratic B-splines

Simple Case: Four Vertices

Let us start out with the simple case of a quadratic B-spline with four vertices. Instead of having the basis functions defined over the interval from $u = 0$ to $u = 1$, we will have two sub-intervals, from $u = 0$ to $u = 1$ and from $u = 1$ to $u = 2$. The basis functions will be polynomials of degree two (order 3) in each interval, and will have matching value and first derivative at the joint $u = 1$. The second derivative will be allowed to be discontinuous at that point. Except for being one degree lower, this sounds just like the cubic spline, where the point of discontinuous derivatives is at the "duck".

In the world of B-splines, the values of the parameter u where basis functions are pieced together are called *internal knots*. So, somehow ducks have become knots. The equation for the B-spline curve looks just the same as for a Bézier curve,

$$\begin{aligned} x = x(u) &= \sum_{i=0}^{3} \xi_i \mathcal{N}_i^2(u) \\ y = y(u) &= \sum_{i=0}^{3} \eta_i \mathcal{N}_i^2(u) \end{aligned} \tag{68}$$

except that if you look closely, you can see that the number of vertices is 4, while the degree of the basis functions is only 2. This requires a different set of basis functions, and these are shown in figure 36, together with their equations.

These do not look much different from the cubic Bézier curve basis functions shown in figure 34. They obviously have the same "right" properties at the ends, and they add up to one for all values of u. The slopes match at the knot ($u = 1$), and you might be able to convince yourself that the second derivatives are discontinuous there. If the graph doesn't convince you, you can differentiate the polynomial expressions twice and set $u = 1$.

We can evaluate the B-spline curve using the same four vertices as before, and the result is shown in figure 37. The curve again starts and stops the way that it should, but it is much different from the cubic Bézier curve. Note, in particular, that the curve appears to be tangent to the control polygon when $u = 1$, and that the point of tangency is half way between the second and third vertex. Hence, the B-spline curve follows the polygon much more closely than the cubic Bézier curve, but it doesn't look as smooth.

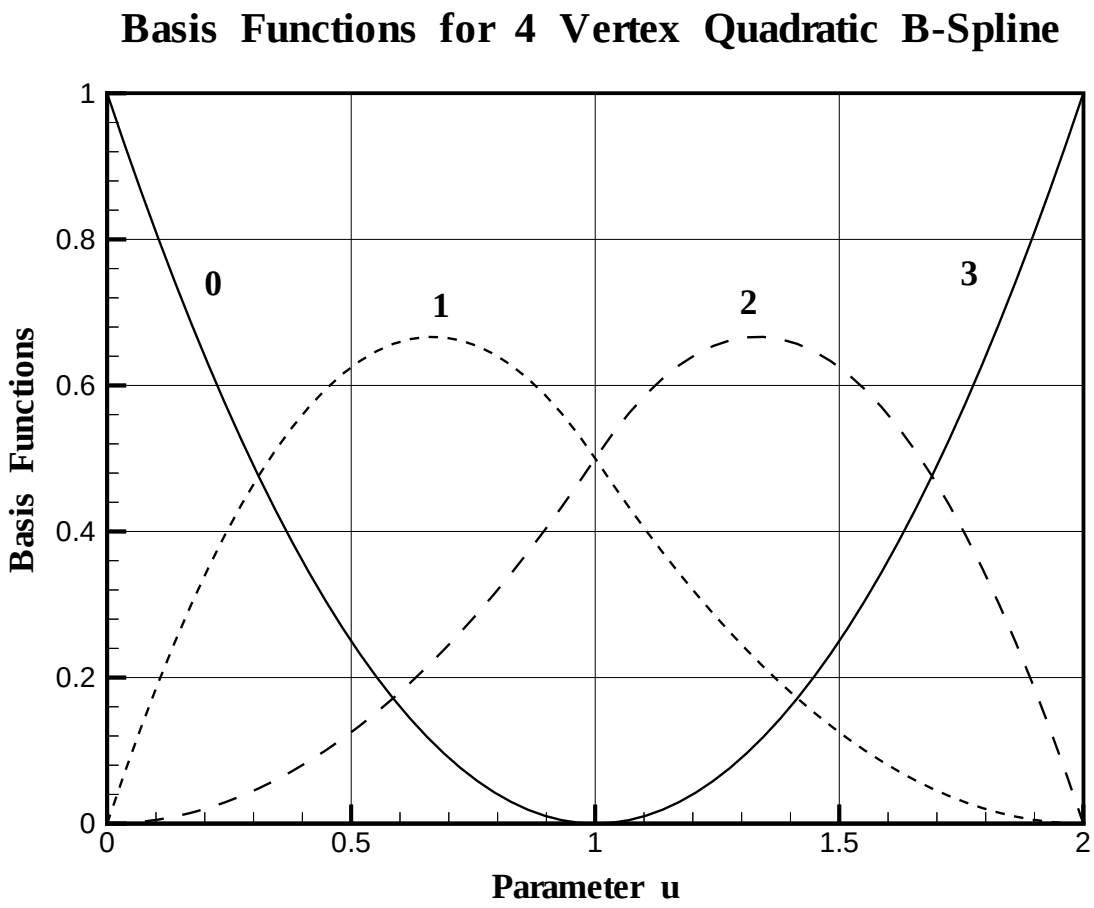
The fact that the curve is tangent to the polygon at $u = 1$ in this example isn't just luck, but is an inherent property of quadratic B-splines. From figure 36 we see that at $u = 1$

$$\begin{aligned} \mathcal{N}_0^2(1.0) &= 0 \\ \mathcal{N}_1^2(1.0) &= 0.5 \\ \mathcal{N}_2^2(1.0) &= 0.5 \\ \mathcal{N}_3^2(1.0) &= 0 \end{aligned} \tag{69}$$

so that from equation 68

$$\begin{aligned} x(1.0) &= \frac{1}{2}(\xi_1 + \xi_2) \\ y(1.0) &= \frac{1}{2}(\eta_1 + \eta_2) \end{aligned} \tag{70}$$

Similarly, by differentiating the basis functions and setting $u = 1$ it can be shown that the slope of the curve is equal to the slope of the second polygon segment. So a computer is hardly necessary to evaluate a quadratic B-spline. Given a control polygon, you don't have to be a great artist to sketch a curve which is tangent at both ends and at the mid-point of the center segment.



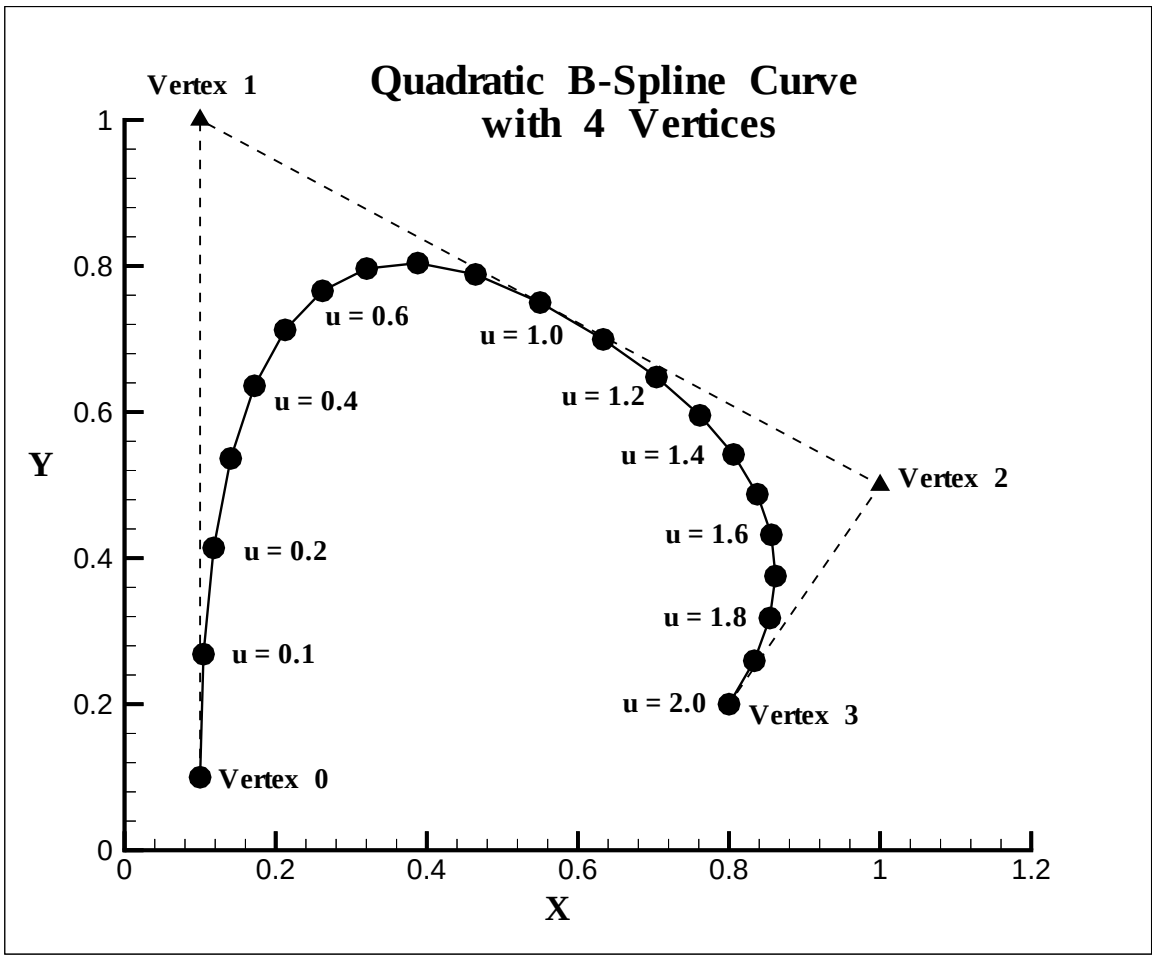
First Interval

$$\mathcal{N}_0^2 = 1 - 2u + u^2$$
$$\mathcal{N}_1^2 = 2u - \frac{3}{2}u^2$$
$$\mathcal{N}_2^2 = \frac{1}{2}u^2$$
$$\mathcal{N}_3^2 = 0$$

Second Interval

$$\mathcal{N}_0^2 = 0$$
$$\mathcal{N}_1^2 = 2 - 2u + \frac{1}{2}u^2$$
$$\mathcal{N}_2^2 = -2 + 4u - \frac{3}{2}u^2$$
$$\mathcal{N}_3^2 = 1 - 2u + u^2$$

Figure 36: Basis functions of a quadratic B-spline with four vertices. The internal knot is at $u = 1$




```
ZONE T="VERTS", I=4
0.10000    0.10000
0.10000    1.00000
1.00000    0.50000
0.80000    0.20000
```

```
ZONE T="VERTS", I=4   0.55000    0.75000
0.10000    0.10000    0.63350    0.69950
0.10450    0.26850    0.70450    0.64800
0.11800    0.41480    0.76150    0.59550
0.14050    0.53650    0.80600    0.54200
0.17200    0.63640    0.83750    0.48750
0.21250    0.71250    0.85600    0.43200
0.26200    0.76600    0.86150    0.37550
0.32050    0.79650    0.85400    0.31800
0.38800    0.80400    0.83350    0.25950
0.46450    0.80400    0.80000    0.20000
```

Figure 37: Sample quadratic B-spline curve with four vertices. The symbols are plotted for evenly spaced values of the parameter u;$(0, (0.1), 1)$.

General Case: Arbitrary Number of Vertices

We can continue this process indefinitely by adding one more vertex, together with one more internal knot. The parameter values corresponding to the knots do not need to be uniformly spaced. If they are not, the curve is called a *Non-Uniform B-Spline*. And— guess what— if the knots are uniformly spaced in u the curve is called a *Uniform B-Spline*. Right now, let's consider uniformly spaced knots, spaced at $u = 1, 2, \ldots, u_{max} - 1$. The number of internal knots is equal to $u_{max} - 1$ and the parameter, u, runs from zero to u_{max}. For a B-spline of any degree,

$$u_{max} = N^v - p \tag{71}$$

where N^v is the number of vertices. For example, for the quadratic Bézier curve, $N^v = 3$, $p = 2$ so $u_{max} = 1$. For the cubic Bézier curve, $N^v = 4$, $p = 3$, so $u_{max} = 1$ again. However, for the four vertex quadratic B-spline curve developed in the preceding section, $N^v = 4$, $p = 2$ so that $u_{max} = 2$. Since $u_{max} \geq 1$, $N^v \geq p + 1$, and when $N^v = p + 1$, there are no internal knots, and a Bézier curve is generated.

As an example, figure 38 shows the basis functions for a seven vertex curve. Thus,

$$u_{max} = 7 - 2 = 5$$

and there are $u_{max} - 1 = 4$ internal knots. The basis functions are all comprised of quadratic segments with matching values and slopes at each of the internal knots. The equations for the first two intervals are given with the figure. Since the pattern repeats itself, the corresponding equations for the remaining intervals can easily be deduced.

Note, in particular, that the third, fourth and fifth basis functions are all identical in shape. As the number of vertices is increased, almost all of the basis functions will look exactly like these. The only exceptions are the two basis functions at each end, which have the special property that insures that the curve starts and stops in the right way.

The "basic lump" consists of three parabolic segments that begin and end with zero value and slope, match value and slope at the two knot values where they are joined, and add up to unity.

Figure 39 shows a sample curve generated by this set of basis functions. It is clear that the curve is tangent to the midpoint of each of the interior segments of the control polygon, and, of course, it starts and stops in the right way.

Note also that the curve has only two inflection points, between the third and fourth, and between the fourth and fifth vertices. An important attribute of B-splines of any degree is that they can have no more inflection points than that of

the control polygon.[12] If you don't want a curve to have any inflections, it is easy to examine and correct the control polygon to insure that this is the case.

5.5 Cubic B-Splines

The basis functions for cubic B-splines are piecewise cubics, with matching value, slope and curvature at the knots. If this sounds familiar, it should, since the equations for the basis functions are exactly the same as for the cubic spline treated earlier.

Figure 40 shows the basis functions for a cubic B-spline with 7 vertices. In accordance with equation 71, the maximum value of the parameter is now $7 - 3 = 4$ and there are three internal knots. The fourth basis function is the "basic lump" which will be repeated as the number of vertices increases beyond seven. This "lump" is the same as a cubic spline with 5 ducks, with end conditions of zero slope and curvature at each end. The equations for the cubic segments for the first two intervals are given with the figure. The rest of the segments are similar, with the quantities within square brackets [ ] adjusted for the particular interval. Again we see that the first and second basis functions have equal and opposite slopes at $u = 0$, with corresponding conditions at the other end.

Figure 41 shows the resulting curve, using the same 7 vertices as in the preceding example. Note that the curve is much smoother, and does not necessarily touch the control polygon, nor is it tangent to it at the mid segments. Nevertheless, it follows the polygon in a general sense.

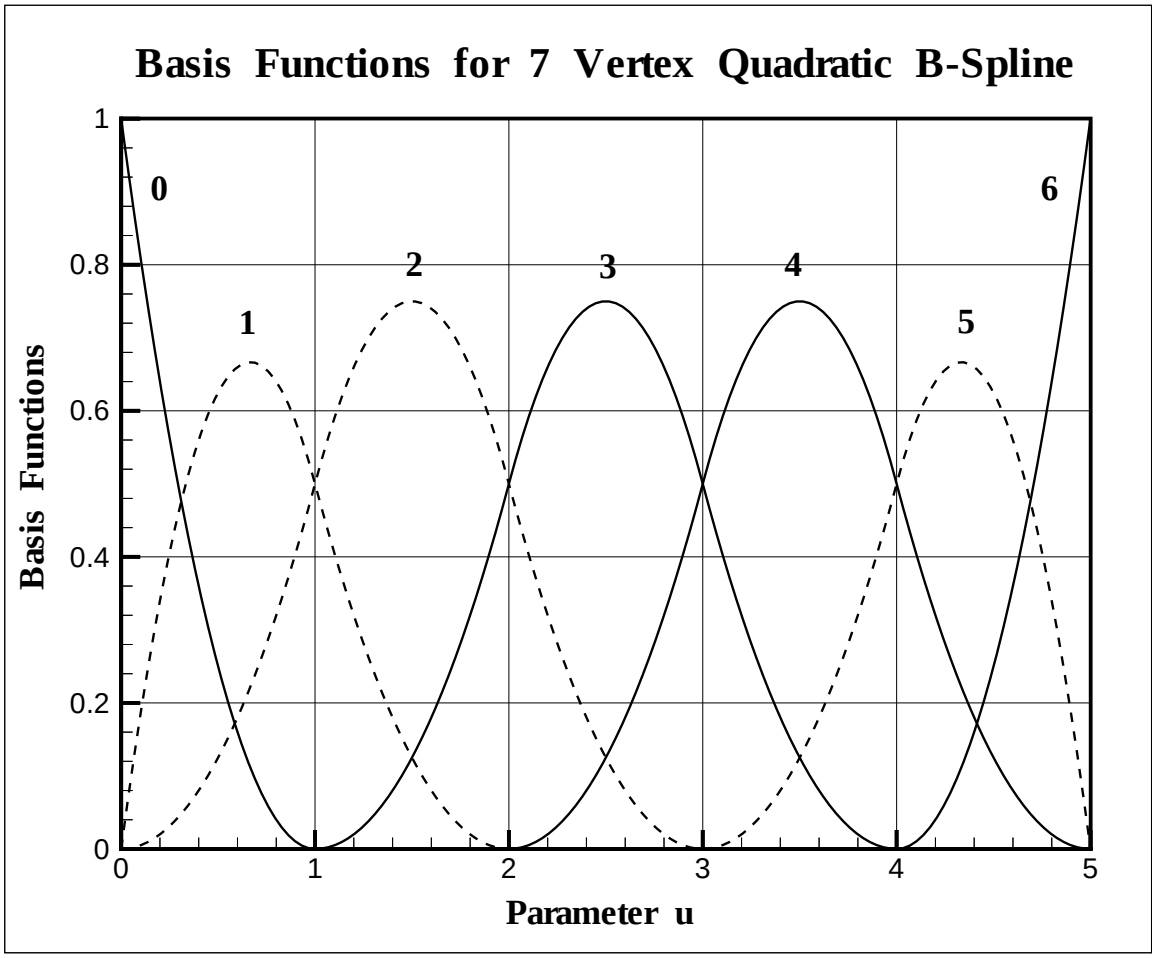
It's finally time to look at a useful example. Figure 42 shows the basis functions for a 10 vertex cubic B-spline in perspective view. The fourth through seventh basis functions can be seen to be identical "basic lumps", shifted by one unit in the parameter u. The perspective brings out the fact that the basis functions are zero over much of the parameter range. This means that a particular vertex can only influence that portion of the curve where its corresponding basis function is non-zero. For example, the third vertex will only influence the curve between parameter values of zero and three. This is a very important attribute of B-spline curves and surfaces, since it means that the designer can tinker with one part without disturbing the rest of the shape.

Finally, figure 43 shows a 10 vertex cubic B-spline representation of a two-dimensional hydrofoil section. The six interior knots are shown as circular symbols on the curve. The resulting foil shape is very smooth, and could actually be used in real life. Yet, the shape of the foil is completely defined by the coordinates of the 10 vertices, and the fact that it was generated as a *uniform, cubic B-spline*. Anyone with a numerically

[12] What does an inflection mean when applied to a polygon? If you start driving along the polygon and turn right at the second vertex, then you have to turn right at all succeeding vertices to avoid receiving a moving violation for creating an inflection.

controlled milling machine (and the right software) can build it with this information alone. If the shape could only be transmitted by coordinates of the surface itself, much more data would be required, particularly near the high-curvature leading edge, and there would be no assurance that the user would not introduce errors through the use of in appropriate interpolation schemes [13].

[13] Unless, of course, they had taken 13.016.



First Interval

$$
\begin{aligned}
\mathcal{N}_0^2 &= 1 - 2u + u^2 \\
\mathcal{N}_1^2 &= 2u - \frac{3}{2}u^2 \\
\mathcal{N}_2^2 &= \frac{1}{2}u^2 \\
\mathcal{N}_3^2 &= 0 \\
\mathcal{N}_4^2 &= 0 \\
\mathcal{N}_5^2 &= 0 \\
\mathcal{N}_6^2 &= 0
\end{aligned}
$$

Second Interval

$$
\begin{aligned}
\mathcal{N}_0^2 &= 0 \\
\mathcal{N}_1^2 &= \frac{1}{2}(1 - [u-1])^2 \\
\mathcal{N}_2^2 &= \frac{1}{2} + [u-1] - [u-1]^2 \\
\mathcal{N}_3^2 &= \frac{1}{2}[u-1]^2 \\
\mathcal{N}_4^2 &= 0 \\
\mathcal{N}_5^2 &= 0 \\
\mathcal{N}_6^2 &= 0
\end{aligned}
$$

Figure 38: Basis functions for quadratic B-spline with 7 vertices.

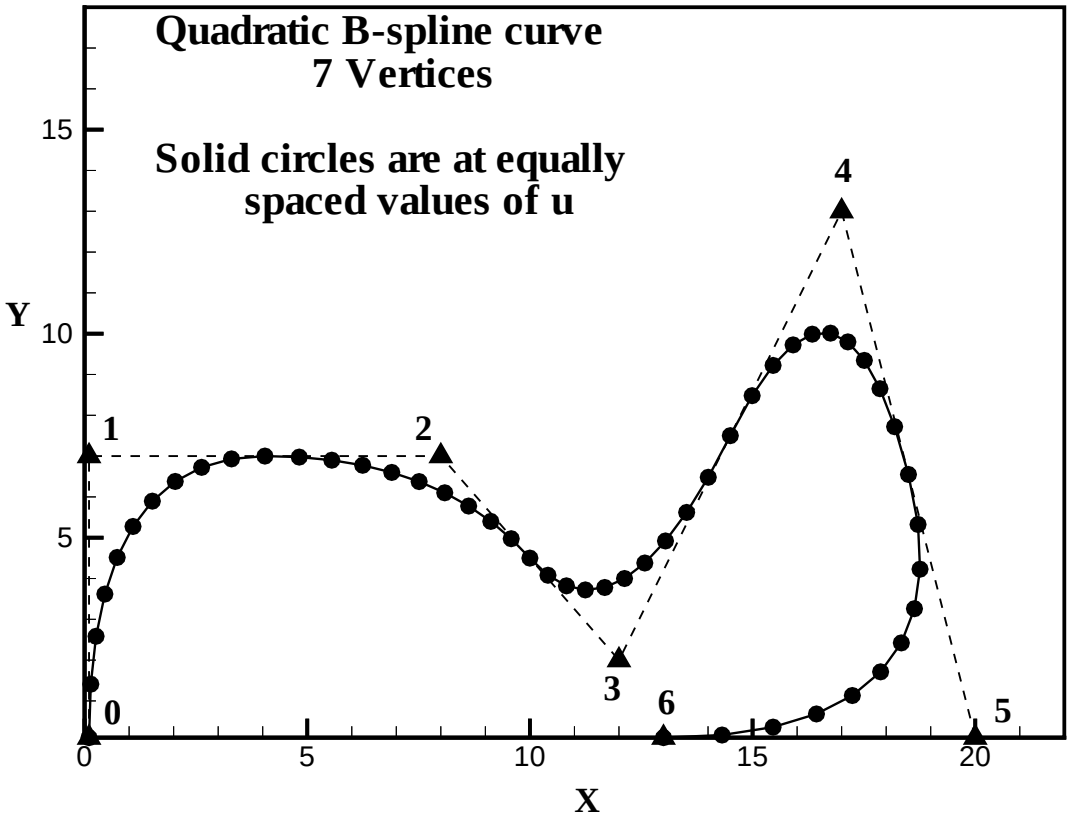
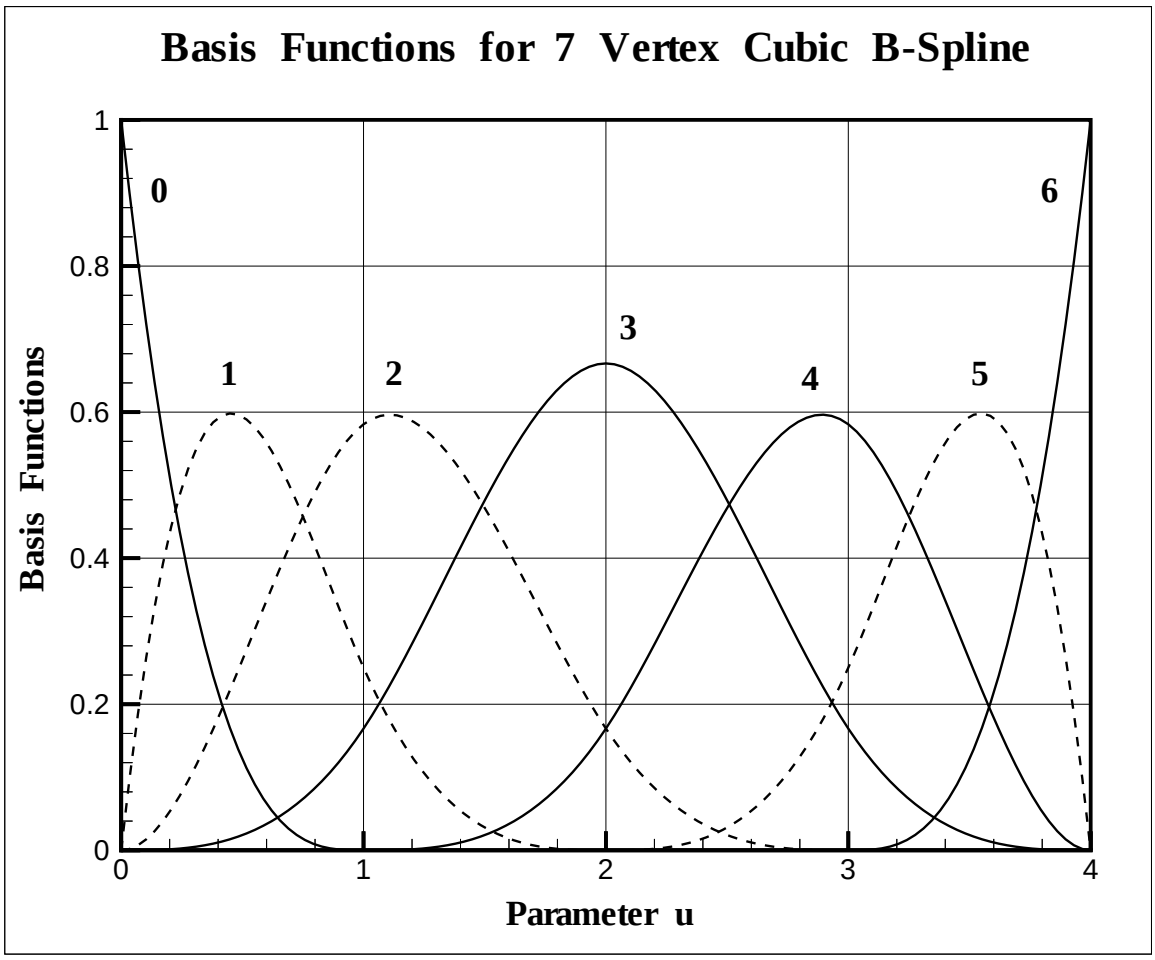


Figure 39: Sample quadratic B-spline curve with 7 vertices.



First Interval

$$
\begin{aligned}
\mathcal{N}_0^3 &= (1-u)^3 \\
\mathcal{N}_1^3 &= \frac{1}{4} + \frac{3}{4}([1-u] + [1-u]^2) \\
&\quad -\frac{7}{4}[1-u]^3 \\
\mathcal{N}_2^3 &= \frac{3}{2} + \frac{11}{12}u^3 \\
\mathcal{N}_3^3 &= \frac{1}{6}u^3 \\
\mathcal{N}_4^3 &= 0 \\
\mathcal{N}_5^3 &= 0 \\
\mathcal{N}_6^3 &= 0
\end{aligned}
$$

Second Interval

$$
\begin{aligned}
\mathcal{N}_0^3 &= 0 \\
\mathcal{N}_1^3 &= \frac{1}{4}[2-u]^3 \\
\mathcal{N}_2^3 &= \frac{1}{6} + \frac{1}{2}([2-u] + [2-u]^2) \\
&\quad -\frac{7}{12}[2-u]^3 \\
\mathcal{N}_3^3 &= \frac{2}{3} - [2-u] + \frac{1}{2}[2-u]^3 \\
\mathcal{N}_4^3 &= \frac{1}{6}[u-1]^3 \\
\mathcal{N}_5^3 &= 0 \\
\mathcal{N}_6^3 &= 0
\end{aligned}
$$

Figure 40: Basis functions for cubic B-spline with 7 vertices

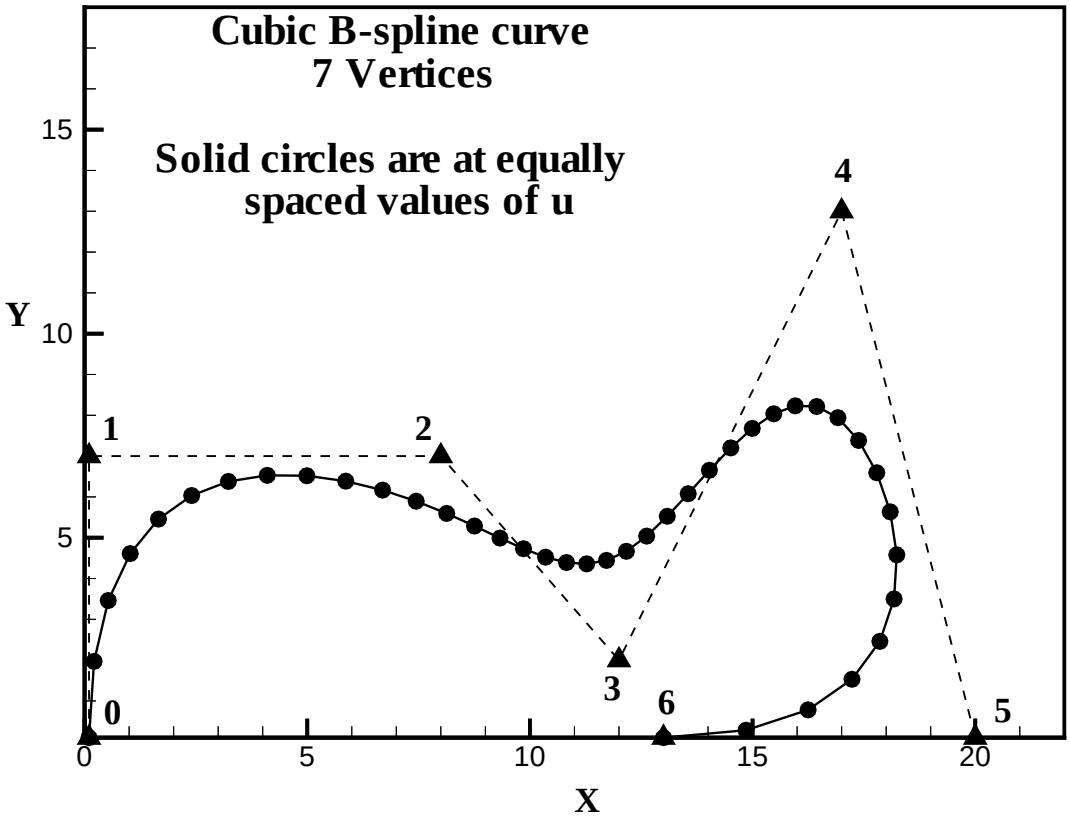


Figure 41: Sample cubic B-spline curve with 7 vertices

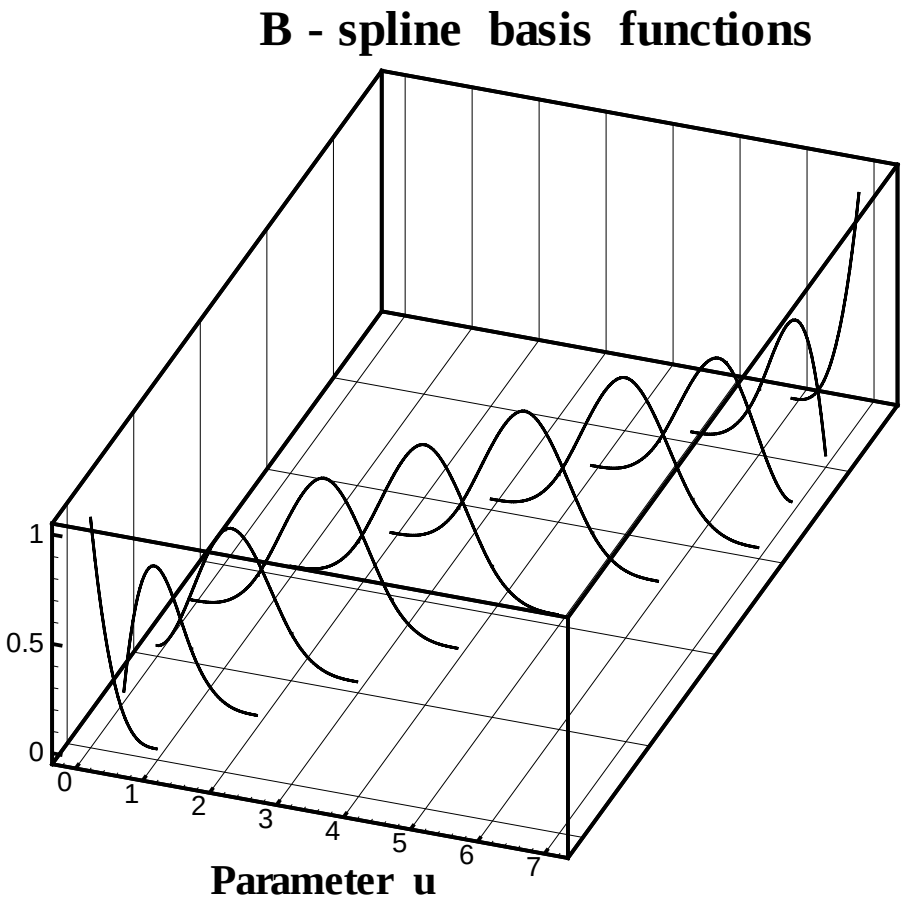


Figure 42: Basis functions for cubic B-spline with 10 vertices

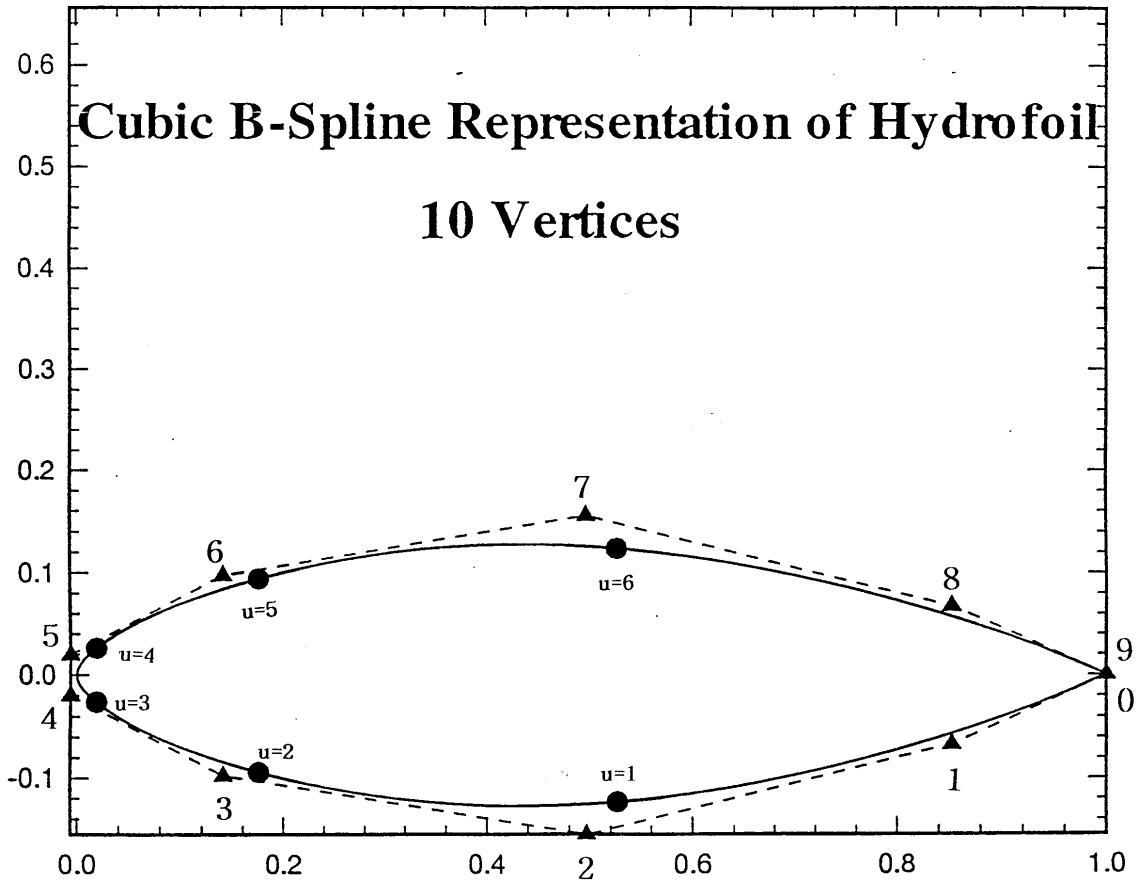


Figure 43: Cubic B-spline representation of a hydrofoil

5.6 The General Case

So far, we have presented the equations for the various sets of basis functions by showing that they have the desired properties. My guess is that the developers of the theory of B-splines must have done a certain amount of this themselves before deciding to get elegant. But, we'll never know, since the cut-and-try approach seldom gets published. However, it turns out that a general formula exists for basis functions of any degree, with any uniform or non-uniform distribution of knots. This was developed by Cox and deBoor, and may be found in a recent text by Piegl and Tiller [14].

The equation is surprisingly simple, but requires that we first introduce the concept of the *knot vector*. We saw that for a uniform B-spline, the interior knots can be written as a sequence of values of u, $(1, 2, 3, \ldots, u_{max} - 1)$. Even for a uniform B-spline, we did not have to use integer values of u, since any linear scaling of the parameter should not have any effect on the results.

For example, for the quadratic B-spline curve with four vertices, the parameter u could be defined to run from (0,1) with the internal knot at $u = 0.5$. The basis functions would look just the same, except for the scale of the u axis. In some references, the parameter u is always normalized to the interval $(0, 1)$.

We also saw, that the basis functions at the ends required special attention. Cox and deBoor showed that the special end conditions could be met by introducing multiple knots at each end. For curves that begin and end at the first and last vertex, start tangent to the first polygon segment and end tangent to the last polygon segment, there must be $p+1$ multiple knots at each end. For example, the knot vector u_i for a uniform cubic B-spline with 7 vertices is

$$u_i = (0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4)$$

while the knot vector for a cubic Beźier curve is

$$u_i = (0, 0, 0, 0, 1, 1, 1, 1)$$

The knot vector for a quadratic B-spline with 7 vertices is

$$u_i = (0, 0, 0, 1, 2, 3, 4, 5, 5, 5)$$

and so forth. The total number of knots in the knot vector is always $N^v + p + 1$.

If we want, we can make the knot intervals non-uniform, and can even have two or more interior knots with the same parameter value,

$$u_i = (0, 0, 0, 1, 2, 2, 3, 4, 4, 4)$$

[14] Les Piegl and Wayne Tiller, **The $\mathcal{NURBS}$ Book**, Second Edition, Springer, 1997.

With the knot vector defined, we can now write down the Cox and deBoor result, which is a recursion formula for the basis functions. In other words, to get the basis functions for a third degree ($p = 3$) B-spline, you must compute the zeroth degree, first degree, second degree, and then finally the third degree values.

The zeroth degree value is simple,

$$\begin{aligned} \mathcal{N}_i^0(u) &= 1 \qquad \text{if } u_i \le u < u_{i+1} \\ \mathcal{N}_i^0(u) &= 0 \qquad \text{otherwise} \end{aligned} \tag{72}$$

Then the next degree is obtained from the preceding one using the relation

$$\mathcal{N}_i^k(u) = \frac{(u - u_i)\mathcal{N}_i^{k-1}(u)}{u_{i+k} - u_i} + \frac{(u_{i+k+1} - u)\mathcal{N}_{i+1}^{k-1}(u)}{u_{i+k+1} - u_{i+1}} \tag{73}$$

with k set equal to 1. Then the operation is repeated for $k = 2$, etc. until $k = p$. Since multiple knots exist at the ends, and may exist in the interior, division by zero could occur in evaluating 73. However, the proper result is obtained if one uses the convention that $0/0 = 0$.

It is remarkable that this simple equation can do the job. Listed below is the C++ code for functions which carry out the calculation. Most of the graphs shown in this section were generated with these functions, so I guess they must be right!

```
//-----------------------------------------------------------------------
//  B-Spline Basis functions and derivatives: Piegl and Tiller algorithm
//  Ref: Les Piegl and Wayne Tiller, "The NURBS Book," Springer, 1997
//
// Symbols used:
//    p: Degree of polynomial; For cubic B-spline, p=3, (nOrder-1)
//    n: Number of Vertices(nV) - 1;  n+1= nV
//    m: Number of knots(nKnots) - 1; m+1 = nKnots
//    m-n-p relation: m = n + p + 1;
//    u: parameter
//    U: knot vector; U[m+1]
//    N: Nonvanishing basis function, P&T p.70; N[p+1]
//-----------------------------------------------------------------------
int ComputeUniformKnotVector(const int n, const int p, double *knot)
{
  int m = n + p + 1;
  double maxknotvalue = double((m) - 2*(p+1) + 2);
  for (int j=0; j<=p; j++) {
    knot[j] = 0.0;
```

```
    knot[m-j] = maxknotvalue;
  }
  for (j=p+1, int i=1; j<=n; j++) {knot[j] = double(i++);}
  return (m+1);
}//..ComputeUniformKnotVector()

//---------------------------------------------------------------------
int FindSpan(int n, int p, double u, double *U)
{
  // Determine the knot span index, P&T p. 68
  // Input: n,p,u,U[m+1]
  // Return: the knot span index
  if (u>=U[n+1]) return(n);  // Special case

  int low = p;
  int high= n+1;             // Do binary search
  int mid = (low+high) / 2;
  while (u<U[mid] || u>=U[mid+1]) {
    if (u<U[mid]) high = mid;
    else          low  = mid;
    mid = (low+high) / 2;
  }
  return(mid);
}//..FindSpan()

//---------------------------------------------------------------------
void BasisFuns(int span, double u, int p, double *U, double *N)
{
  // Nonvanishing basis functions, computed from P&T algorithm, p.70
  //  Input: span, u, p, U[m+1]
  //  Output: N[p+1]
  double saved, temp;
  CVector<double> left(p+1), right(p+1);

  N[0] = 1.0;
  for (int j=1; j<=p; j++) {
    left[j] = u - U[span+1-j];
    right[j]= U[span+j] - u;
    saved = 0.0;
    for (int r=0; r<j; r++) {
      temp = N[r]/(right[r+1]+left[j-r]);
      N[r] = saved+right[r+1]*temp;
```

```
      saved = left[j-r]*temp;
    }
    N[j] = saved;
  }
}//..BasisFuns()
```

Some more examples

The first example, shown in figure 44, compares a second degree, third degree and fourth degree B-spline defined by the *same* set of 5 vertices. As expected, the second degree curve is tangent to the midpoints of the second and third sides of the control polygon. On the other hand, both the third and fourth degree curves are completely removed from the polygon, except at the end points. As the degree increases, the curve becomes smoother, and departs more from the control polygon.

The next example shows the effect of a multiple knot. Figure 45 shows the basis functions for a uniform cubic B-spline with eight vertices. These look the same as the seven vertex case shown in figure 40, except that there are now two identical "basic lumps".

This should be compared with figure 46, which also has 8 vertices, but contains a double knot at $u = 3$. If we use the same seven vertex locations as in figure 41, but make the fourth and fifth one double, the resulting curve will be as shown in figure 47. Note that the curve has been drawn towards the double vertex, and that it has very high curvature in that region.

The next example, figure 48, shows the basis functions for a cubic B-spline with a triple knot at $u = 3$. The basis functions clearly have a discontinuous slope at this point. For a cubic B-spline, all derivatives up to the second are continuous at the knots. A double knot reduces the order of continuity by one, so that only first derivatives are continuous. Finally, a triple knot reduces the continuity to value only.

The curve produced by the same seven vertices, but with the fifth vertex triple, is shown in figure 49. Note that in this case, the curve is drawn to the fifth vertex as though it were an end point. So why not break up the curve into two segments, and fit each one separately?

In many applications, this might be the simplest and most practical way to deal with discontinuities in curves (and surfaces). However, the information that this particular curve is being broken up into two segments at this point has to be communicated somehow. On the other hand, specifying the knot vector and the vertices of the control polygon is completely general in that all the information necessary to generate both continuous and discontinuous curves can be transmitted between CAD/CAM systems in a uniform way.

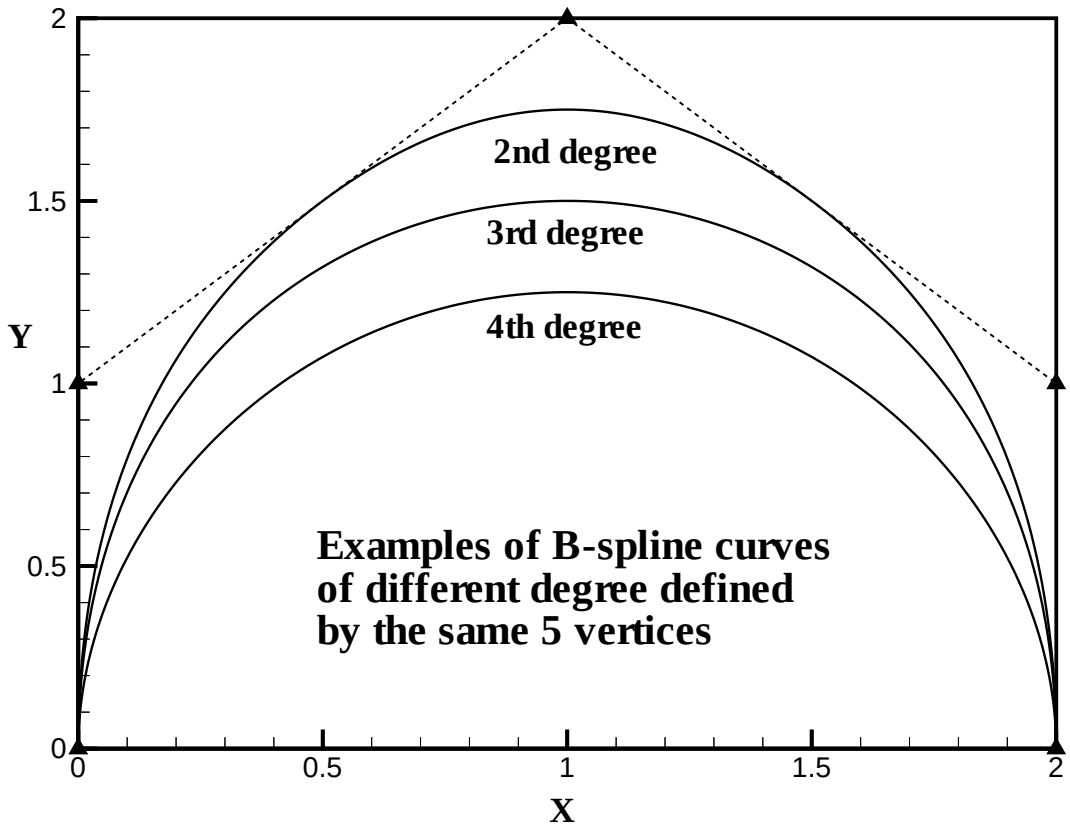


Figure 44: Comparison of B-spline curves of different degree

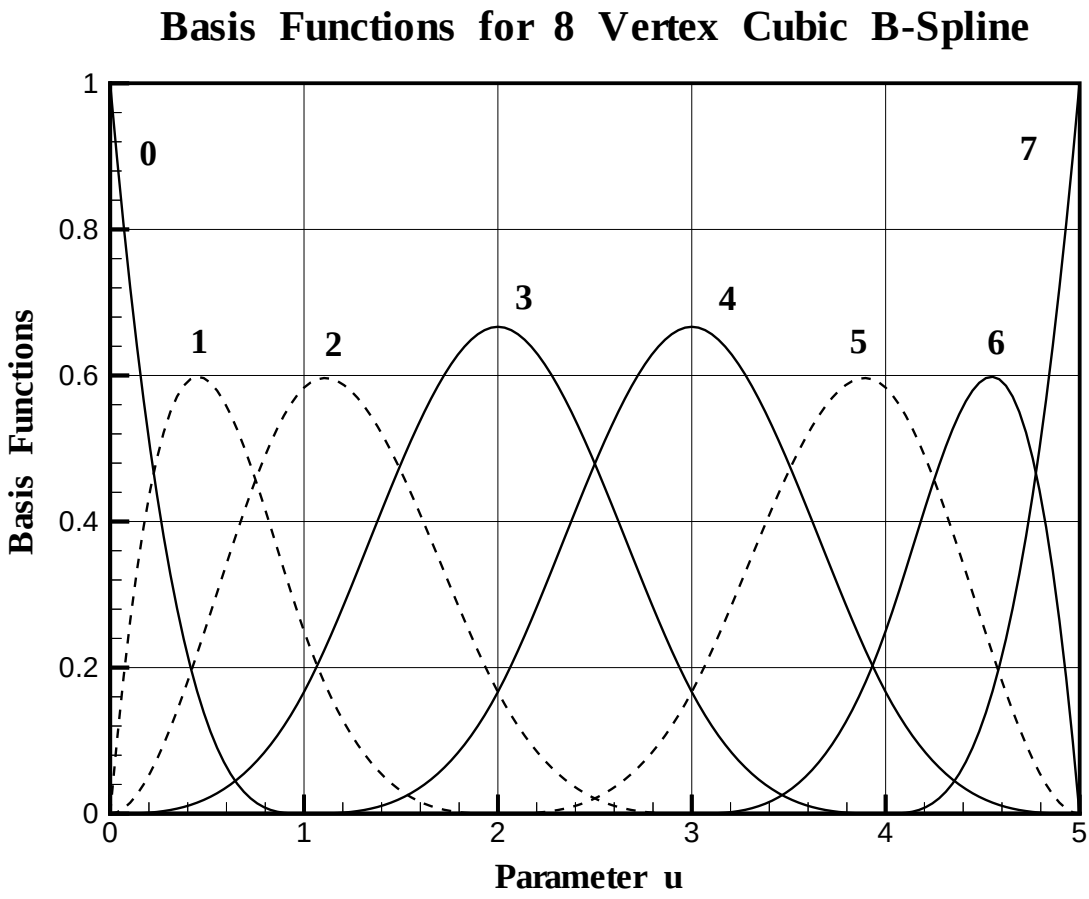


Figure 45: Basis functions for a cubic B-spline with 8 vertices

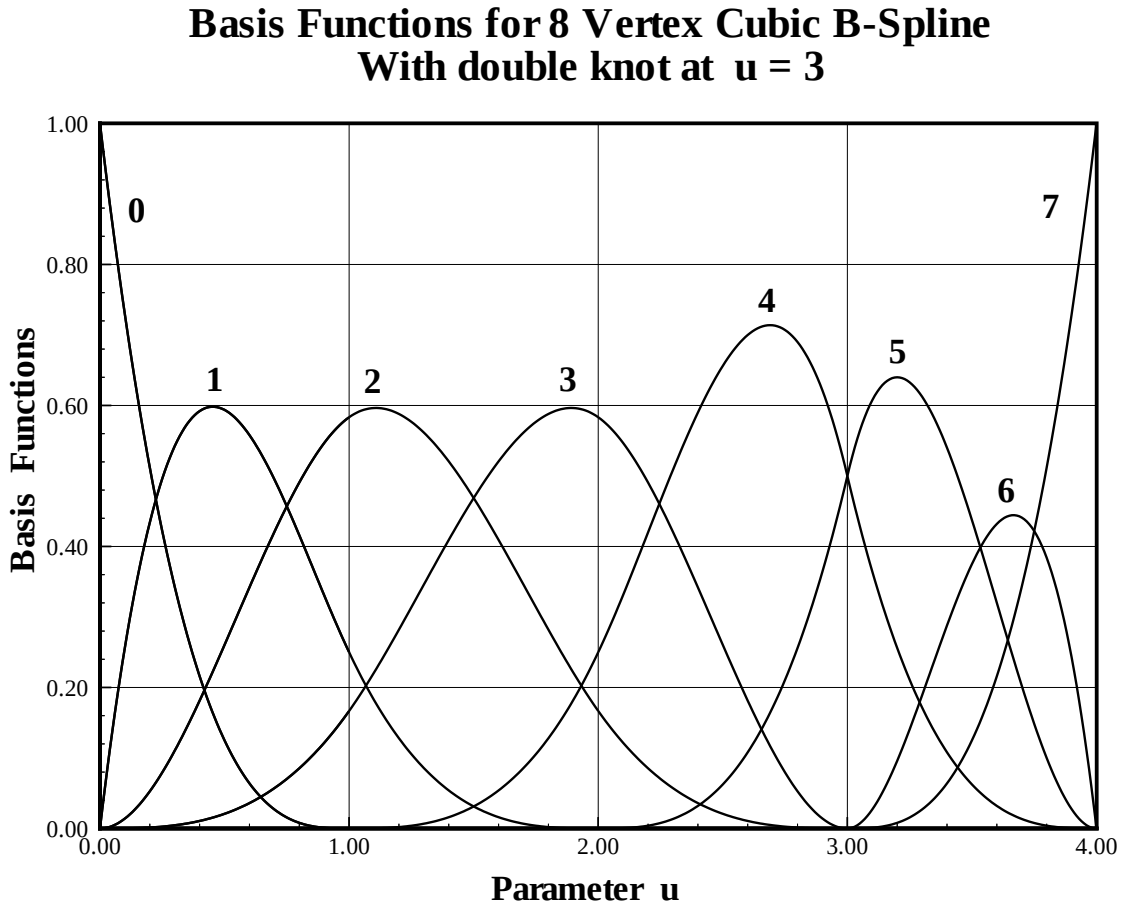


Figure 46: Basis functions for a cubic B-spline with 8 vertices with a double knot at $u = 3$

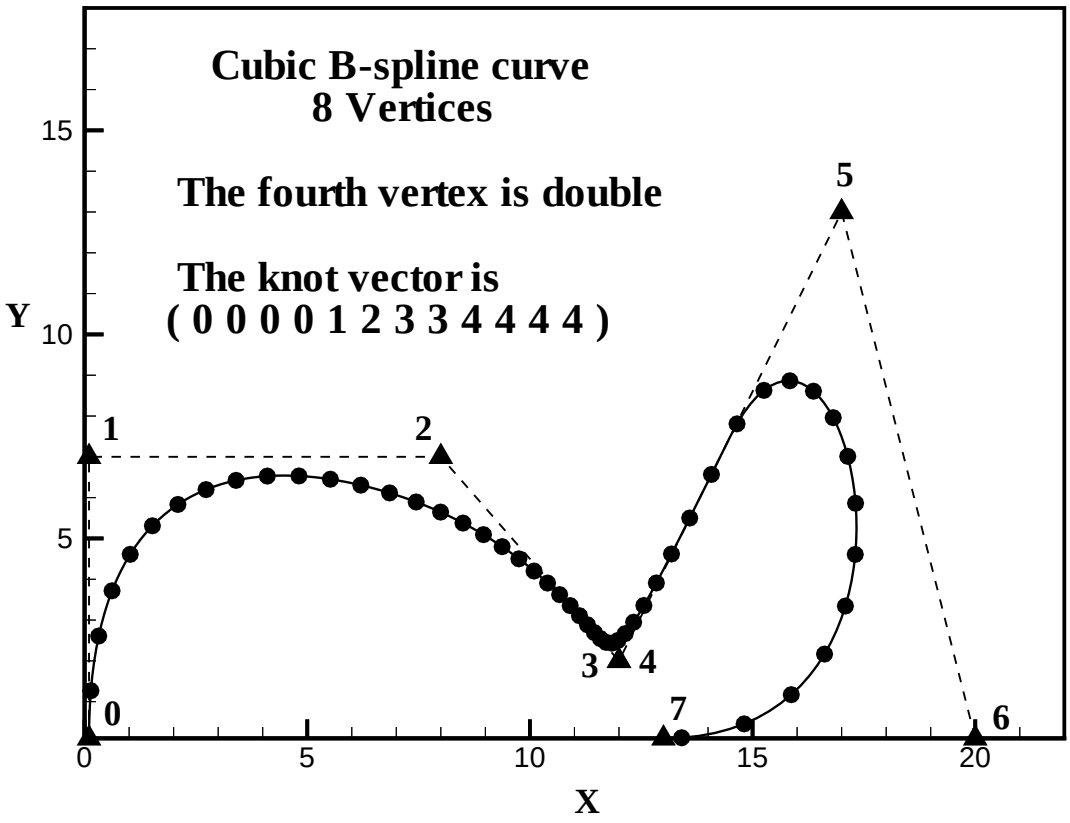


Figure 47: B-spline curve generated from the basis functions shown in the preceding figure

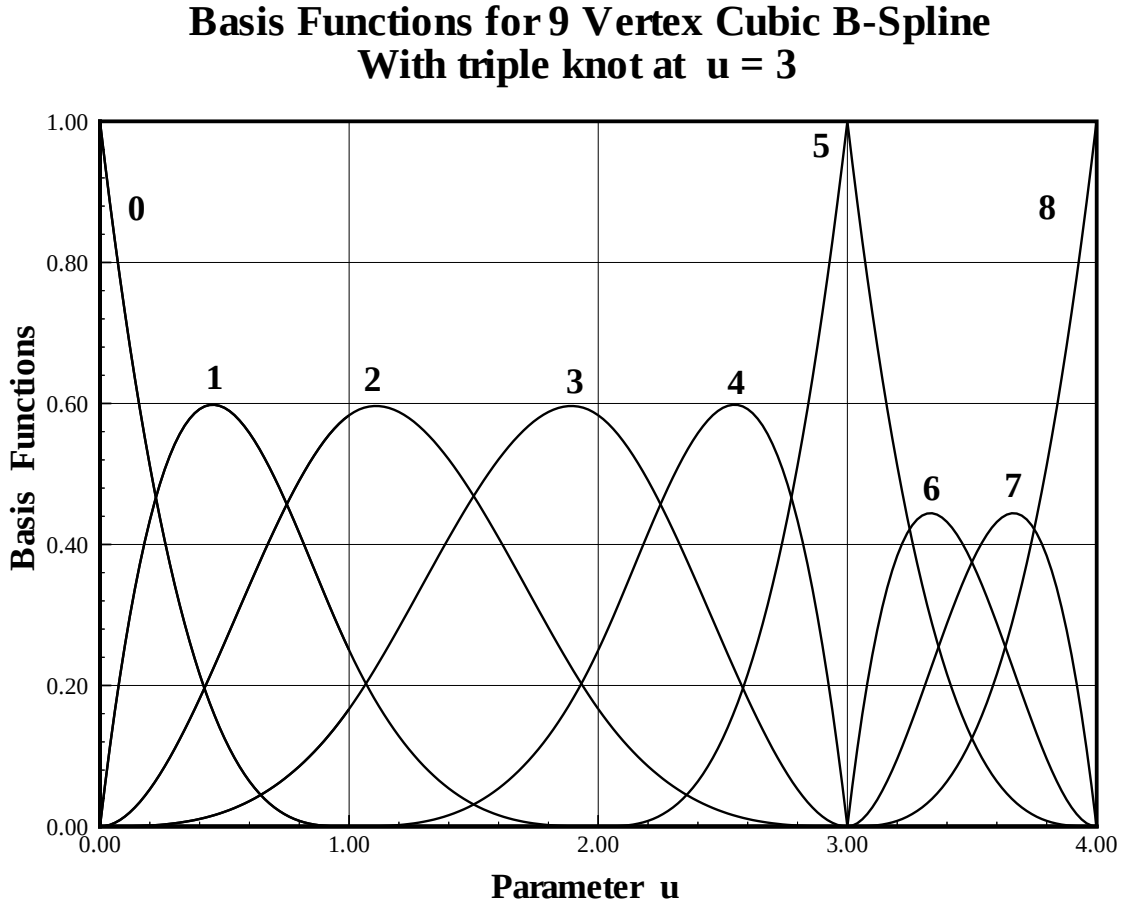


Figure 48: Basis functions for a cubic B-spline with 9 vertices with a triple knot at $u = 3$

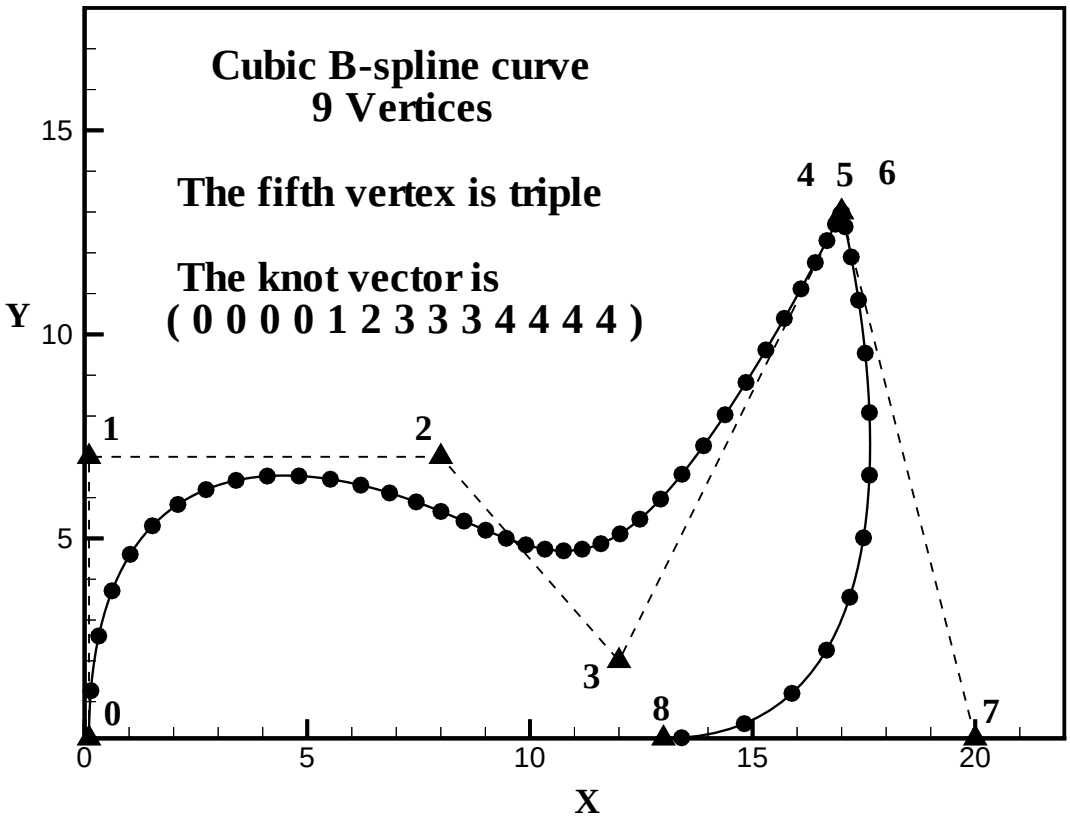


Figure 49: B-spline curve generated from the basis functions shown in the preceding figure

6 RATIONAL B-SPLINES

We have seen that B-spline curves can represent a wide variety of shapes. However, they cannot represent elementary shapes such as circles, ellipses and hyperbolas *exactly*. You might ask why should we worry about that, since we already know how to define such shapes analytically.

Suppose we wanted to represent a precision machine part where a portion of it had to be exactly circular, while adjoining parts were to have some general shape. It would be convenient if the shape of that part could be expressed in a single, unified way that any CAD/CAM system could understand.

This was the motivation for extending B-splines polynomial basis functions to include ratios of polynomials. These are called *rational B-splines*, and it can be shown (but not in 13.016) these can represent general conic curve segments such as ellipses and hyperbolas exactly.

Rational B-splines are defined as follows,

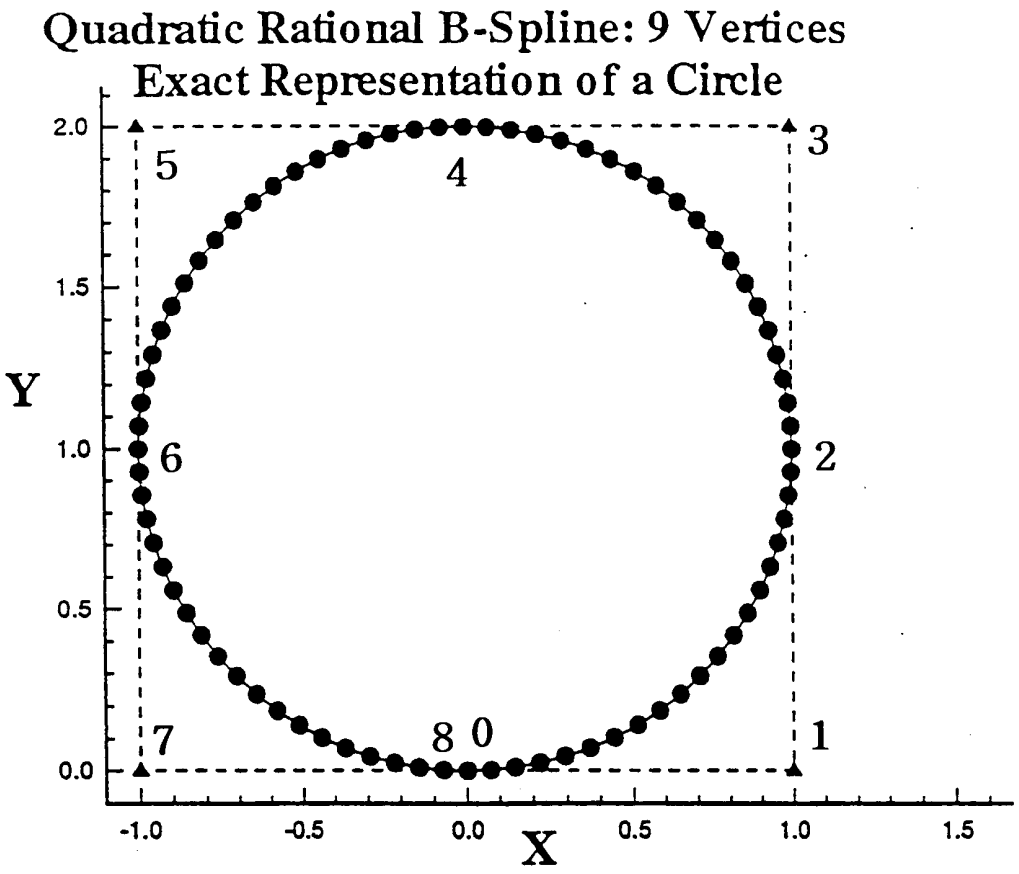
$$\begin{aligned} x(u) &= \frac{\sum_{i=0}^{N^v-1} \xi_i h_i \mathcal{N}_i^p(u)}{\sum_{i=0}^{N^v-1} h_i \mathcal{N}_i^p(u)} \\ y(u) &= \frac{\sum_{i=0}^{N^v-1} \eta_i h_i \mathcal{N}_i^p(u)}{\sum_{i=0}^{N^v-1} h_i \mathcal{N}_i^p(u)} \end{aligned} \tag{74}$$

where h_i are *weights* specified by the user. Note, in particular, that when all weights are one, equation 74 reduces to an "ordinary" or *integral* B-spline. Hence, equation 74 provides the desired unified expression for almost all types of curves. When combined with the knot vector introduced in the preceding section, curves with both continuous and discontinuous derivatives may be produced. Recall that when the interior knots are not equally spaced, or are multiple, a non-uniform B-spline curve results, and that a uniform B-spline results as a special case when the interior knots (guess what) are uniformly spaced. The combination of the arbitrary knot vector and the introduction of rational basis functions produces a class of curves known as Non-Uniform Rational B-Spline curves, or NURBS curves.

As an example, figure 50 shows the world's most complicated way to draw a circle. Rogers and Adams show that one can produce a circle with a second degree (quadratic) rational B-spline with 9 vertices placed along a square. The six internal

knots are each double, and the weights alternate between 1.0 and $\sqrt{2}/2$. The resulting curve is, indeed, a circle [15].

[15] So now the Renault can finally have wheels!



i	ξ_i	η_i	h_i
0	0.0	0.0	1.00000
1	1.0	0.0	0.70711
2	1.0	1.0	1.00000
3	1.0	2.0	0.70711
4	0.0	2.0	1.00000
5	-1.0	2.0	0.70711
6	-1.0	1.0	1.00000
7	-1.0	0.0	0.70711
8	0.0	0.0	1.00000

Knot Vector: $(0,0,0,1,1,2,2,3,3,4,4,4)$

Figure 50: The world's most complicated way to draw a circle.

6.1 3-D B-Spline Curves

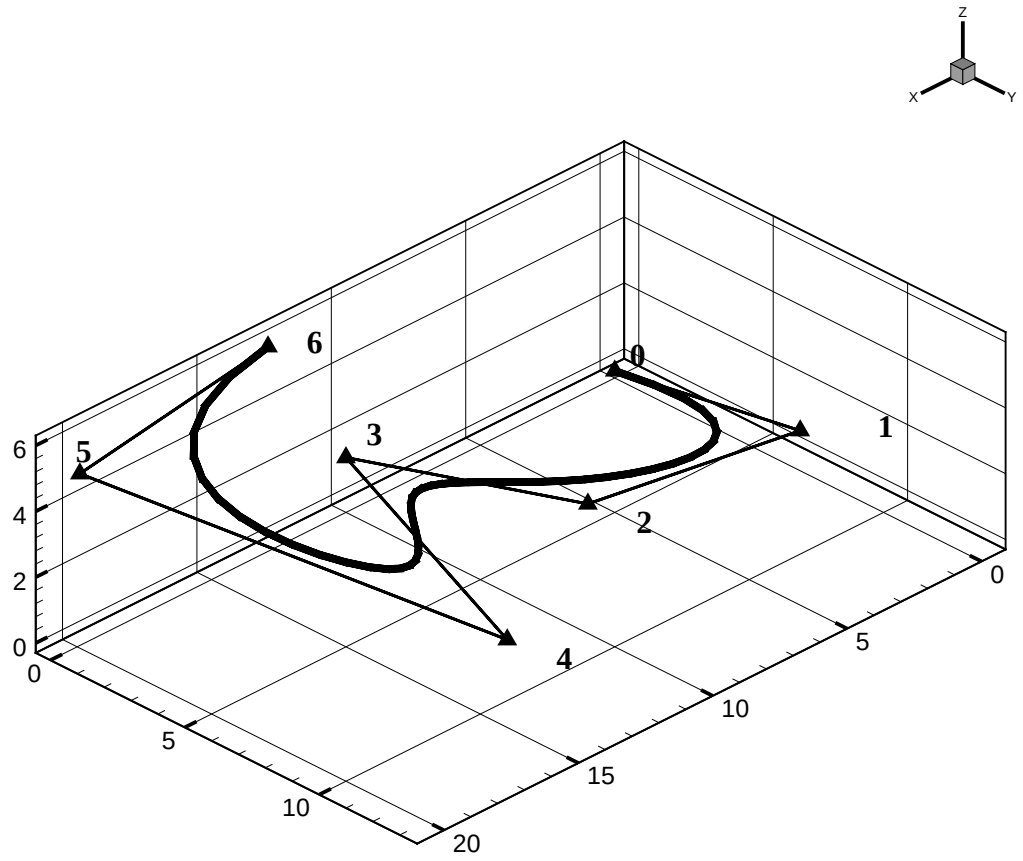


Figure 51: Three dimensional extension of the B-spline curve shown in Figure 8

One of the most remarkable attributes of the theory of B-splines is the ease with which one can extend their definition from two to three dimensions. All we have to do is to consider that the control polygon is three-dimensional, with vertices given by

$$(\xi_0, \eta_0, \zeta_0), (\xi_1, \eta_1, \zeta_1), \ldots, (\xi_{N^v-1}, \eta_{N^v-1}, \zeta_{N^v-1}) \tag{75}$$

and we can obtain the z coordinate of any point on the curve in exactly the same way as before,

$$
\begin{aligned}
x(u) &= \sum_{i=0}^{N^v-1} \xi_i \mathcal{N}_i^p(u) \\
y(u) &= \sum_{i=0}^{N^v-1} \eta_i \mathcal{N}_i^p(u) \\
z(u) &= \sum_{i=0}^{N^v-1} \zeta_i \mathcal{N}_i^p(u)
\end{aligned} \tag{76}
$$

Note that the basis functions are not affected, since they are functions of the single parameter, u.

Figure 51 is a perspective view of a three dimensional B-spline curve developed from the curve shown in Figure 8 simply by adding ζ values to the coordinates of the vertices.

7 B-SPLINE SURFACES

The first step in going from a three-dimensional space curve to a surface is to extend the idea of a control polygon to a *control net*. The vertices of the control net are (ξ, η, ζ) as before, but we now need two indices to keep track of them. As shown in figure 53, the i index advances from 0 to its maximum value minus one along each row, while the j index does the same along each column. Each vertex can therefore be identified as,

$$(\xi_{i.j}, \eta_{i,j}, \zeta_{i,j}) \quad i = 0, N^v - 1,\ j = 0, M^v - 1 \tag{77}$$

where N^v is the number of vertices in the i direction and M^v is the number of vertices in the j direction. In the figure, the number of vertices in both directions is 4.

The equation of the B-spline surface can now be written as follows:

$$\begin{aligned} x(u,v) &= \sum_{i=0}^{N^v-1} \sum_{j=0}^{M^v-1} \xi_{i,j} \mathcal{N}_i^p(u) \mathcal{N}_j^q(v) \\ y(u,v) &= \sum_{i=0}^{N^v-1} \sum_{j=0}^{M^v-1} \eta_{i,j} \mathcal{N}_i^p(u) \mathcal{N}_j^q(v) \\ z(u,v) &= \sum_{i=0}^{N^v-1} \sum_{j=0}^{M^v-1} \zeta_{i,j} \mathcal{N}_i^p(u) \mathcal{N}_j^q(v) \end{aligned} \tag{78}$$

Points on the surface are obtained as the sum of the product of the coordinates of each of the vertices of the control net with *two* basis functions. Two parameters are needed to define a point on the surface, so we have replaced the parameter u with two new ones, u and v. However, it is important to recognize that the basis functions themselves are just the same as before.

The degree of the two sets of basis functions need not be the same (although they generally are). $\mathcal{N}_i^p$ is the basis function of degree p associated with the i-th vertex, while $\mathcal{N}_j^q$ is the basis function of degree q associated with the j-th vertex.

Suppose we set the parameter $v = 0$. Then, $\mathcal{N}_0^q = 1.0$ and all the rest are zero. As the parameter u is varied, a three-dimensional space curve is generated, which is just the same as if we had formed a control polygon consisting of only the first row of the control net. The other four edges of the surface are generated in a similar manner.

This is clearly evident from figure 53 which shows a cubic B-spline surface generated by the (4×4) control net. The edge curves are therefore cubic Bézier curves.

For any other values of the parameters u and v, one simply evaluates one set of basis functions at the value of u, evaluates the other set at the value of v, multiples

them by the appropriate vertex coordinates, and adds them up. The resulting surface is displayed in the figure as a gridwork of lines taken at equally spaced values of the two parameters.

A very smooth surface results, which follows the net in a predictable way.

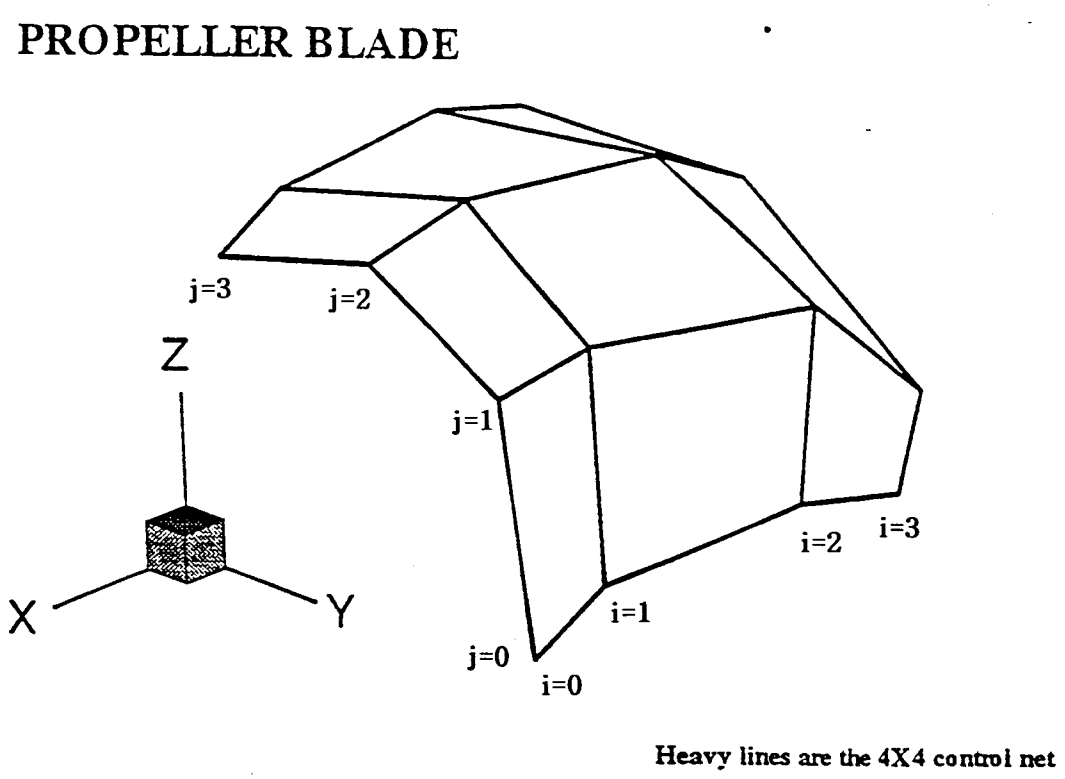


Figure 52: Control polygon net for a B-Spline surface

Figure 54 shows a Cubic B-spline surface generated by a 9×9 net. Hence, the u and v parameter values each run from zero to 6. The resulting surface is nearly the same as before, but one could control the local shape by moving one or more of the individual vertices.

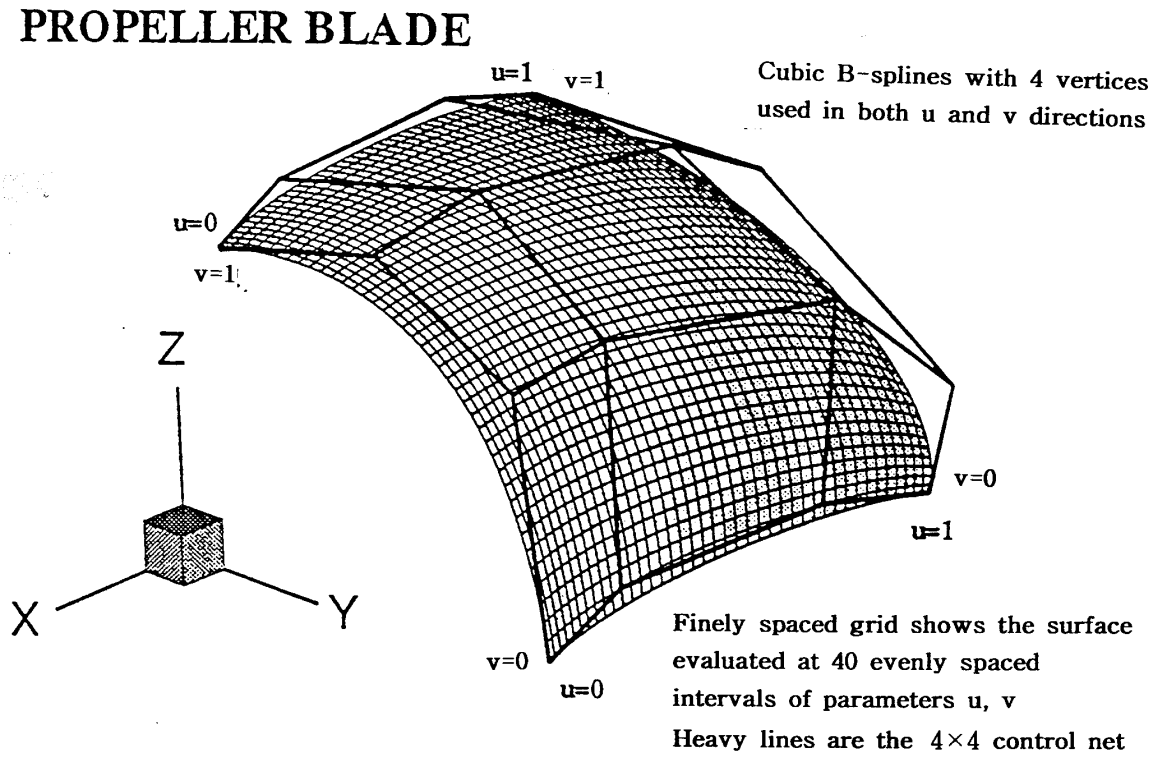


Figure 53: B-Spline surface

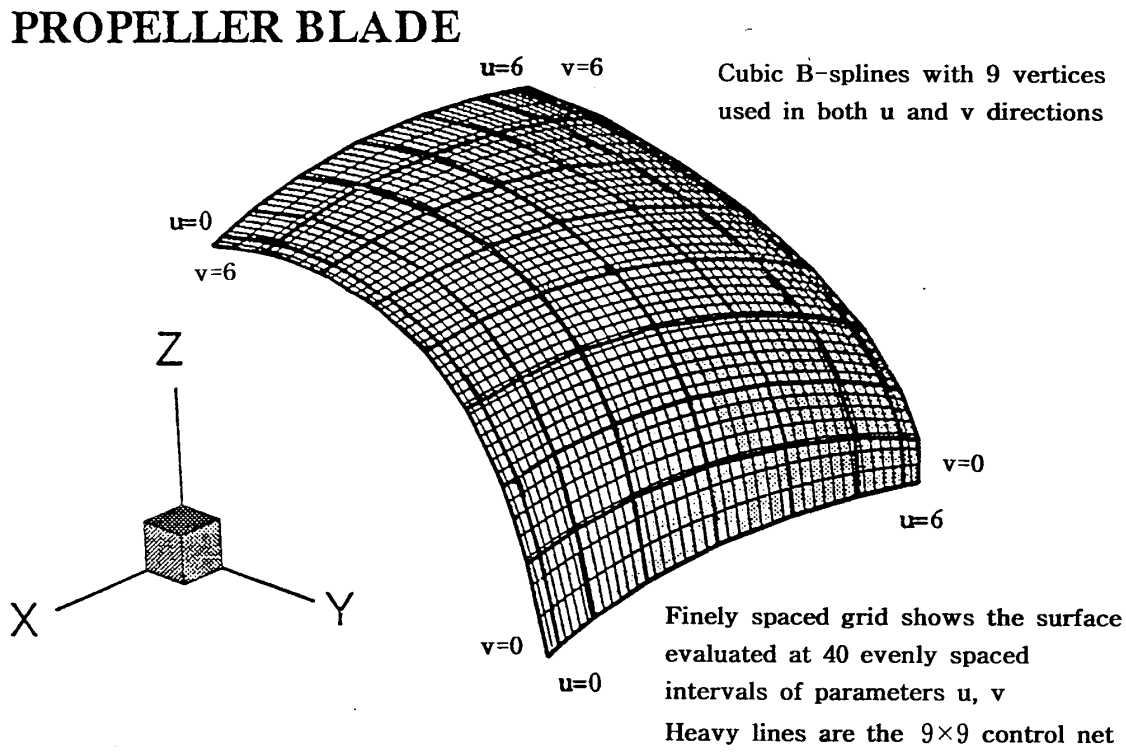


Figure 54: Another B-Spline surface

8 NORMAL VECTORS

A normal vector, $\vec{n} = (n_x, n_y, n_z)$, is a vector of unit length which is perpendicular to a plane which is tangent to a surface at a given point. Accurate determination of normal vectors is important, both for manufacturing and for hydrodynamic and structural analysis.

If an object is to be manufactured by a numerically controlled machine, it is not sufficient to know the shape of the surface. This is because milling cutters have a finite radius, and as a result, the path of the axis if the machine must be offset from the desired surface by the radius of the cutter. The direction of the offset to the surface is in the direction of the normal vector.

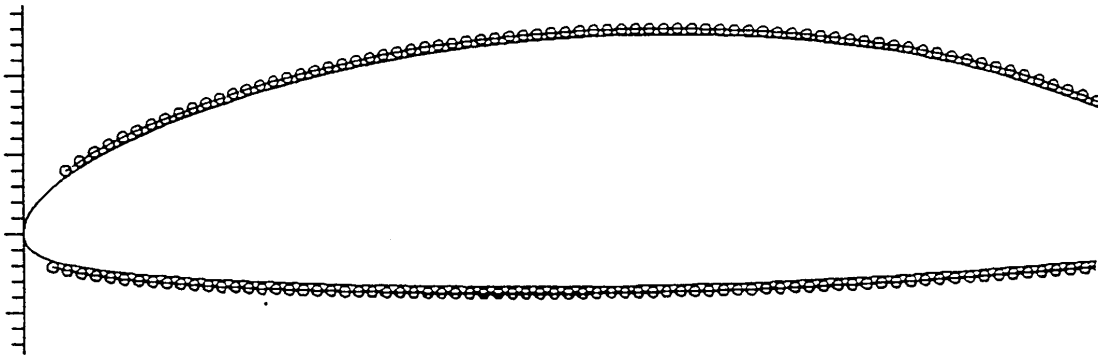


Figure 55: Illustration of path of milling cutter in machining a foil section.

Components for offshore platforms and ship hulls are frequently fabricated from steel plates of constant thickness. If the outside of the hull is built to the desired surface shape, the inside is offset by the plating thickness. Interior structure such as bulkheads and frames must be shaped exactly to fit the inside of the plating. Hence, an offset surface is again required.

Hydrodynamic calculations frequently need to know the direction of the surface normal. For an ideal fluid, the boundary condition on the surface is that the component of fluid velocity in the direction of the surface normal must be equal to the normal component of the velocity of the body— otherwise, there would be flow through the boundary. Slight inaccuracies in the normal vector can introduce unacceptable errors in a flow calculation.

The normal vector can be computed exactly for any B-spline curve or surface. For a plane, two-dimensional curve, the components of the normal vector are,

$$
\begin{aligned}
n_x &= -\frac{\frac{\partial y}{\partial u}}{\sqrt{\left(\frac{\partial y}{\partial u}\right)^2 + \left(\frac{\partial x}{\partial u}\right)^2}} \\
n_y &= +\frac{\frac{\partial x}{\partial u}}{\sqrt{\left(\frac{\partial y}{\partial u}\right)^2 + \left(\frac{\partial x}{\partial u}\right)^2}} \\
n_z &= 0
\end{aligned} \tag{79}
$$

This is illustrated in figure 56.

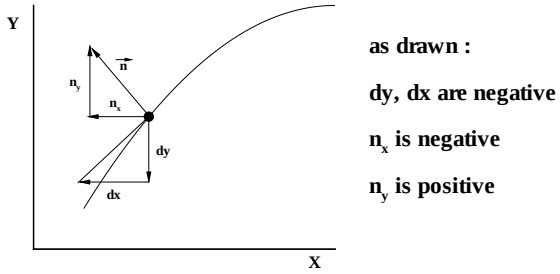


Figure 56: Construction of the normal vector

The derivatives required in Equation 79 can be obtained from the sum of the products of the derivatives of the basis functions with the coordinates of the control polygon, as shown in Equation (3). Since the basis functions are piecewise polynomials, their derivatives can be obtained easily. For example, figure 57 shows the first and second derivatives of the basis functions for a uniform cubic B-spline with 7 vertices.

The unit normal vector for a surface is a little more complicated. Instead of working with the three coordinates of a point on a surface, let us consider a single position vector,

$$\vec{P}(u,v) = [x(u,v), y(u,v), z(u,v)] \tag{80}$$

Then, the quantities

$$\frac{\partial \vec{P}(u,v)}{\partial u} du \text{ and } \frac{\partial \vec{P}(u,v)}{\partial v} dv \tag{81}$$

are differential vectors tangent to the surface at point P. A vector normal to the surface can then be constructed by forming the cross product of these two vectors. Finally, this can be made a unit vector by dividing by the magnitude of the cross product of the two vectors,

$$\vec{n} = \frac{\frac{\partial \vec{P}}{\partial u} \times \frac{\partial \vec{P}}{\partial v}}{\left| \frac{\partial \vec{P}}{\partial u} \times \frac{\partial \vec{P}}{\partial v} \right|} \tag{82}$$

8.1 Curvature of plane curves

The smoothness of a surface can best be judged by examining the distribution of curvature. Unwanted inflections, or "waviness", can easily be seen on the full size object under the proper lighting conditions, and this can be replicated on a high-resolution workstation screen using three dimensional rendering techniques. However, a quantitative measure of curvature changes and lines of inflection is frequently desirable.

Let us first consider the curvature of a planar B-spline curve. The traditional equation for curvature is

$$\kappa = \frac{1}{\rho} = \frac{\frac{d^2y}{dx^2}}{\left(1 + (\frac{dy}{dx})^2\right)^{3/2}} \tag{83}$$

We can evaluate Equation 83 by computing dy/dy and d^2y/dx^2 from the B-spline. However, there is a problem when the slope is vertical. Both the numerator and denominator will approach infinity, and while the ratio may be finite, there is no telling what the computer program will produce.

One solution is to test if the slope is large, and to rotate coordinates by ninety degrees if this is the case. One can then evaluate a analogous expression involving dx/dy instead of dy/dx. This works perfectly well, but requires the additional test on dy/dx which might someday be forgotten. A more robust equation can be derived by introducing the parametric forms of the derivatives in Equation 83. Let us start with the denominator:

$$\left(1 + \left(\frac{dy}{dx}\right)^2\right)^{3/2} = \left(1 + \left(\frac{\frac{dy}{dt}}{\frac{dx}{dt}}\right)^2\right)^{3/2} = \frac{\left((\frac{dx}{dt})^2 + (\frac{dy}{dt})^2\right)^{3/2}}{(\frac{dx}{dt})^3} \tag{84}$$

and the numerator (in excruciating detail) is:

$$\frac{d^2y}{dx^2} = \frac{d}{dx}\left(\frac{dy}{dx}\right) = \frac{d}{du}\left(\frac{\frac{dy}{du}}{\frac{dx}{du}}\right)\frac{du}{dx} = \left(\frac{\frac{d^2y}{du^2}\frac{dx}{du} - \frac{d^2x}{du^2}\frac{dy}{du}}{\left(\frac{dx}{du}\right)^2}\right)\left(\frac{1}{\frac{dx}{du}}\right) \tag{85}$$

Combining Equations 84 and 85 gives the final expression for the curvature,

$$\kappa = \frac{1}{\rho} = \frac{\frac{d^2y}{du^2}\frac{dx}{du} - \frac{d^2x}{du^2}\frac{dy}{du}}{\left((\frac{dx}{du})^2 + (\frac{dy}{du})^2\right)^{3/2}} \tag{86}$$

This last expression is simple to evaluate, and is robust, since the derivatives with respect to the parameter u can never be infinite.

Figure 58 gives an example of the use of both the normal vector and curvature for a foil section represented by a cubic B-spline. Here, the local curvature of a large number of points on the foil surface is plotted as a vector in the direction of the surface normal. This is called a *porcupine plot* for obvious reasons, and serves as a good visual check on the smoothness of the foil section.

8.2 Curvature of surfaces

Consider a point, P, on a surface, with normal vector $\vec{n}$. If we pass a plane through the point P which is tangent to the normal vector, its intersection with the surface will produce a plane curve, with curvature κ at point P. However, this is not unique, since we can rotate this plane around the normal vector to any angle, and each intersection curve with the surface may have a different curvature.

If we locate the maximum curvature κ_{max} and the minimum curvature κ_{min}, the corresponding angular orientations of the plane are called the principal axes of curvature. It can be shown (but not here) that the principal axes of curvature are always at right angles.

This is easy to visualize for a cylinder or cone, where the maximum curvature is in a plane at right angles to its axis, while the minimum curvature (which is zero in this case) is in a plane containing the axis.

In general, there is no single value for the curvature at a given point on a surface. However, two measures of curvature are frequently used:

1. *Average Curvature*: The average of the minimum and maximum.

2. *Gaussian Curvature*: The product of the minimum and maximum.

If the minimum curvature is zero, which is the case for conic sections, the Guassian curvature will be zero, no matter how large the maximum curvature is. This is a useful measure in determining how developable a surface is. A piece of paper is easy to roll up into a cylinder or cone, but it is impossible to make it into a sphere. Steel plates used for offshore platforms (or automobile bodies) are similar. Small curvatures can be introduced by elastic bending, and large curvatures can be produced by simple rolling machines— but only in one direction. Compound curvatures require a much more complicated forming process. Therefore, a map of the Gaussian curvature distribution of a surface will be essential in planning a fabrication process.

Equations for the average and Gaussian curvature are given in *Rogers and Adams* and are not included in the present edition of the notes. As for the plane curve, the curvature is computed from the first and second derivatives of a point on the surface with respect to the two parameters u and v. The important point is that their evaluation is relatively simple, and is *exact*.

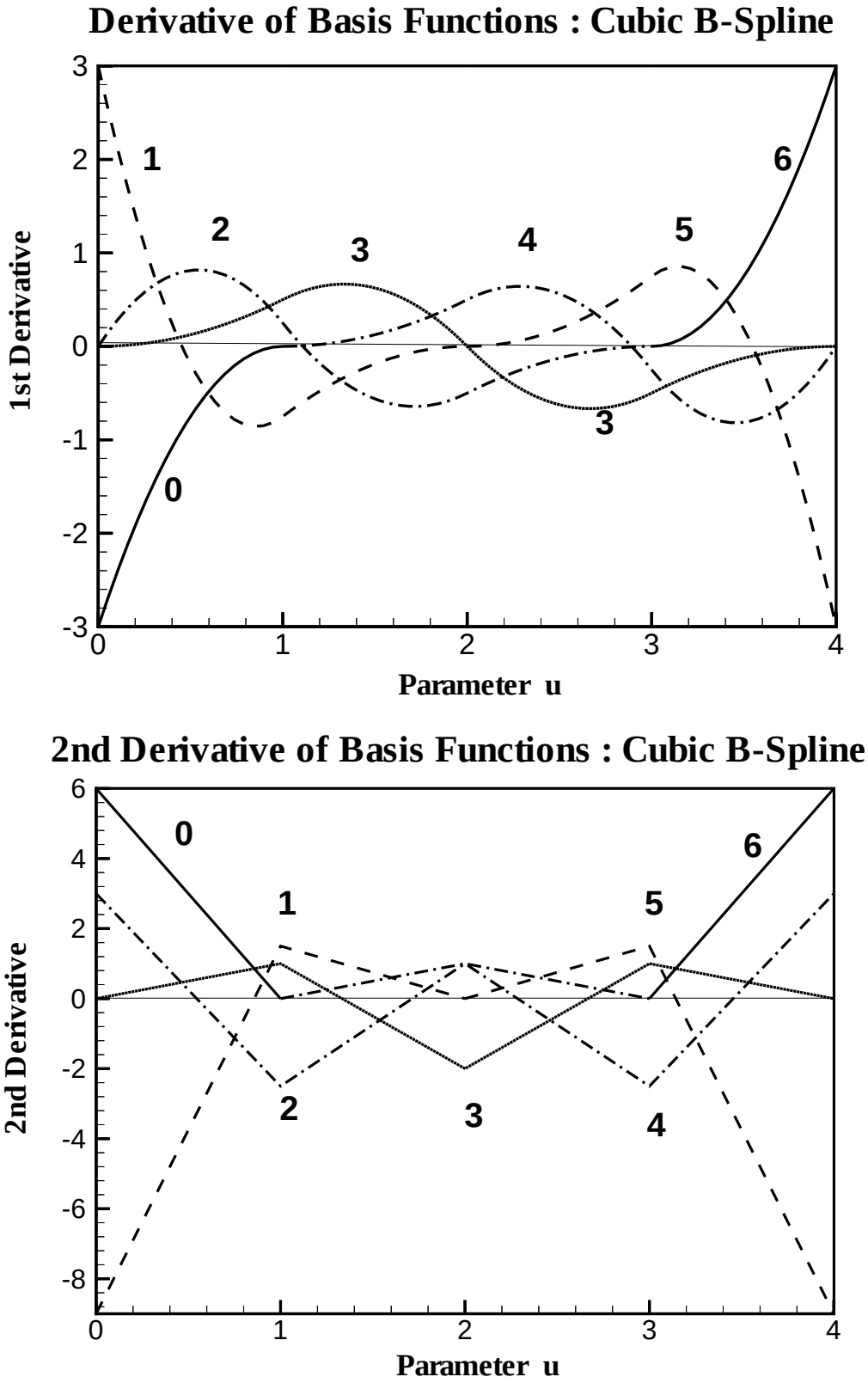


Figure 57: Derivatives of cubic B-spline

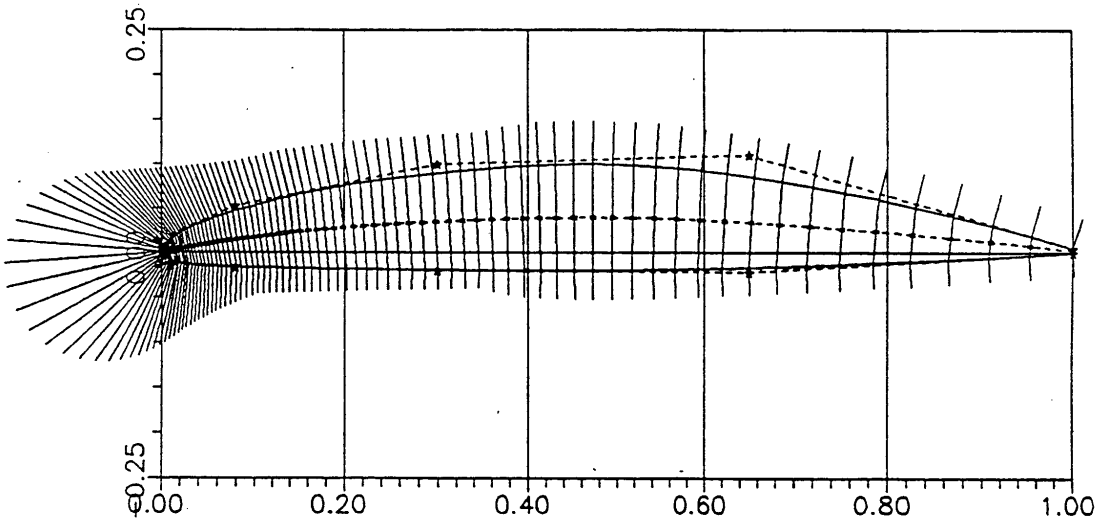


Figure 58: Porcupine plot showing the curvature distribution of a foil represented by a cubic B-spline.

9 Fitting Existing Data with B-splines

We indicated earlier that it is easy to *design* curves and surfaces with B-splines since manual adjustment of the vertices produces very predictable results. Suppose, instead, that we want to fit a B-spline to a given set of base points. One way to do it is to display the points on the computer screen, locate a set of vertices and their resulting B-spline curve, and then adjust the vertices by trial and error until the B-spline passes through the base points. This actually works quite well, particularly once the operator has had a little practice.

However, there are occasions when it is necessary to perform the fitting process automatically. We saw that this was easy in the case of polynomials or cubic splines, since algorithms could be developed which produced curves that passed exactly through a given set of base points. So why not do it with B-splines?

Let's look at the equations defining a two-dimensional B-spline curve:

$$\begin{aligned} x(u) &= \sum_{i=0}^{N^v-1} \xi_i \mathcal{N}_i^p(u) \\ y(u) &= \sum_{i=0}^{N^v-1} \eta_i \mathcal{N}_i^p(u) \end{aligned} \tag{87}$$

If we are given a set of N^v base points $(x_n(u), y_n(u),\ u = 0, 1, \ldots N^v - 1)$ we can write down equation 87 for each base point, thus obtaining a set of simultaneous equations for the unknown vertex locations ξ_i, η_i. However, we can only do this if we also know the value of the parameter u for each of the base points. We need u in order to find the basis functions.

The problem is that we don't know what u is. We saw in numerous previous examples that u was not necessarily proportional to arc length along the curve, but that it just came out somehow. On the other hand, there doesn't seem to be any law against having u proportional to arc length. So the standard *textbook* procedure for fitting a B-spline to given data is to assign parameter values to each input point which are equal to the polygonal arc length along the curve, scaled so that the maximum value of the parameter is $N^v - p$. The polygonal arc length, s_n, is obtained in the same way as was done for the parametric polynomial fit treated earlier.

$$s(0) = 0 \qquad s_n = s_{n-1} + \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2} \quad n = 1, \ldots N^v - 1 \tag{88}$$

and the corresponding parameter values are,

$$u_n = \left(\frac{N^v - p}{s_{N^v-1}} \right) s_n \tag{89}$$

To illustrate this, let's try fitting $\sin(x)$ from zero to 180 degrees using a uniform third degree (cubic) B-spline with 7 vertices. We will require the B-spline to pass through 7 points equally spaced in x. The following table gives the 7 values of the function, the poly-arc length and the scaled parameter values;

x	$\sin(x)$	poly-arc	parameter:u
0.000	0.00000	0.000	0.00000
30.000	0.50000	30.004	0.66671
60.000	0.86603	60.006	1.33338
90.000	1.00000	90.007	2.00000
120.000	0.86603	120.007	2.66662
150.000	0.50000	150.009	3.33329
180.000	0.00000	180.013	4.00000

We now write equation 87 for each of the 7 base points,

$$\begin{aligned}
\xi_0\mathcal{N}_0^3(u_0) + \xi_1\mathcal{N}_1^3(u_0) + \xi_2\mathcal{N}_2^3(u_0) + \xi_3\mathcal{N}_3^3(u_0) + \xi_4\mathcal{N}_4^3(u_0) + \xi_5\mathcal{N}_5^3(u_0) + \xi_6\mathcal{N}_6^3(u_0) &= x_0 \\
\xi_0\mathcal{N}_0^3(u_1) + \xi_1\mathcal{N}_1^3(u_1) + \xi_2\mathcal{N}_2^3(u_1) + \xi_3\mathcal{N}_3^3(u_1) + \xi_4\mathcal{N}_4^3(u_1) + \xi_5\mathcal{N}_5^3(u_1) + \xi_6\mathcal{N}_6^3(u_1) &= x_1 \\
\xi_0\mathcal{N}_0^3(u_2) + \xi_1\mathcal{N}_1^3(u_2) + \xi_2\mathcal{N}_2^3(u_2) + \xi_3\mathcal{N}_3^3(u_2) + \xi_4\mathcal{N}_4^3(u_2) + \xi_5\mathcal{N}_5^3(u_2) + \xi_6\mathcal{N}_6^3(u_2) &= x_2 \\
\xi_0\mathcal{N}_0^3(u_3) + \xi_1\mathcal{N}_1^3(u_3) + \xi_2\mathcal{N}_2^3(u_3) + \xi_3\mathcal{N}_3^3(u_3) + \xi_4\mathcal{N}_4^3(u_3) + \xi_5\mathcal{N}_5^3(u_3) + \xi_6\mathcal{N}_6^3(u_3) &= x_3 \\
\xi_0\mathcal{N}_0^3(u_4) + \xi_1\mathcal{N}_1^3(u_4) + \xi_2\mathcal{N}_2^3(u_4) + \xi_3\mathcal{N}_3^3(u_4) + \xi_4\mathcal{N}_4^3(u_4) + \xi_5\mathcal{N}_5^3(u_4) + \xi_6\mathcal{N}_6^3(u_4) &= x_4 \\
\xi_0\mathcal{N}_0^3(u_5) + \xi_1\mathcal{N}_1^3(u_5) + \xi_2\mathcal{N}_2^3(u_5) + \xi_3\mathcal{N}_3^3(u_5) + \xi_4\mathcal{N}_4^3(u_5) + \xi_5\mathcal{N}_5^3(u_5) + \xi_6\mathcal{N}_6^3(u_5) &= x_5 \\
\xi_0\mathcal{N}_0^3(u_6) + \xi_1\mathcal{N}_1^3(u_6) + \xi_2\mathcal{N}_2^3(u_6) + \xi_3\mathcal{N}_3^3(u_6) + \xi_4\mathcal{N}_4^3(u_6) + \xi_5\mathcal{N}_5^3(u_6) + \xi_6\mathcal{N}_6^3(u_6) &= x_6
\end{aligned}$$

A similar set of equations relates the η coordinates of the vertices to the y coordinates of the base points.

The coefficient matrix in this set of equations simply consists of each basis function evaluated at each of the parameter values assigned to the base points. Since this is the same for both sets of unknowns, ξ_i, η_i, one only needs to solve one set of simultaneous equations for two right hand sides, which is only slightly more work than for one set of right hand sides.

The basis functions for this example are plotted in figure 59, and the resulting coefficient matrix is tabulated on the next page. Entering figure 59 at the parameter value of each base point, one can easily verify the entries in the table— at least qualitatively.

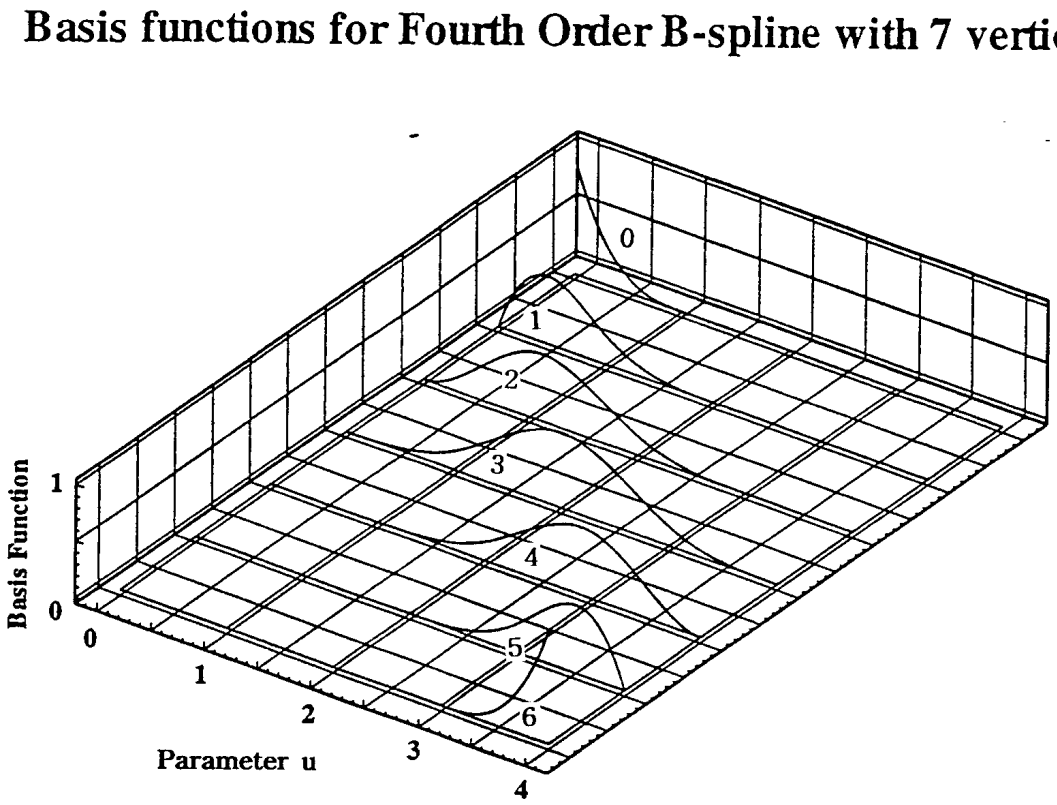


Figure 59: Basis functions used in sample fit.

Coefficient matrix for sample fit

1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
0.03702	0.51849	0.39510	0.04939	0.00000	0.00000	0.00000
0.00000	0.07406	0.54937	0.37040	0.00618	0.00000	0.00000
0.00000	0.00000	0.16667	0.66667	0.16667	0.00000	0.00000
0.00000	0.00000	0.00618	0.37040	0.54937	0.07406	0.00000
0.00000	0.00000	0.00000	0.04939	0.39510	0.51849	0.03702
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	1.00000

Note that the sum of the elements in each row of the matrix add up to one, as they should, and that each row contains at most four non-zero elements. The solution of this system of equations gives the following values for the coordinates of the 7 vertices:

Computed vertex coordinates for fit to sine function

ξ_n	η_n
0.00000	0.00000
14.99886	0.25974
44.99661	0.78629
89.99998	1.10686
135.00342	0.78629
180.00000	0.00000

Once the vertex positions are known, we can evaluate equation 87 at as many parameter values as we wish to obtain a curve of the B-spline approximation to the sine function, and this is shown in figure 60. The resulting B-spline curve goes though each of the base points, as it should, and the control polygon looks reasonable.

Inspired by this good result, lets try to fit the foil section shown earlier in the notes. In this case, there are twelve base points, starting with the trailing edge, passing over the lower surface to the leading edge, and returning along the upper surface to the trailing edge. The result is shown in figure 61. The B-spline passes exactly through all the base points, but we have somehow created a whale. It's a nice whale, but it is not exactly what we ordered!

The problem is that arc length is a very bad choice for the parameter values in this case. If you look at the previous foil example, the parameter varies much more rapidly in the high curvature leading edge region than near the trailing edge. So, assigning suitable parameter values to the base points is critical in this case. On the other hand, the curvature changed more gradually for the sine curve, so the simple arc length option worked well enough.

There are two ways to turn the whale into a foil. One way is to chose better parameter values for the base points, and the other is to alter the knot vector— thus producing a non-uniform B-spline. Either method will solve the problem. Unfortunately, it is not obvious how either of these quantities should be changed in the general case. Some automated procedures require substantially more input data points than vertices, and solve the problem by least-squares. Additional knots may be automatically introduced in regions where the error is large, and the process repeated until the required accuracy is obtained. While the result may be satisfactory, one generally ends up with far more vertices than would be required if the fit were performed with human interaction.

We will show in the next section how one can "tune up" a B-spline fitting method to work well for a particular class of problems. However, it is important to recognize that specialized methods cannot be guaranteed to be robust, so that one must always examine the results closely.

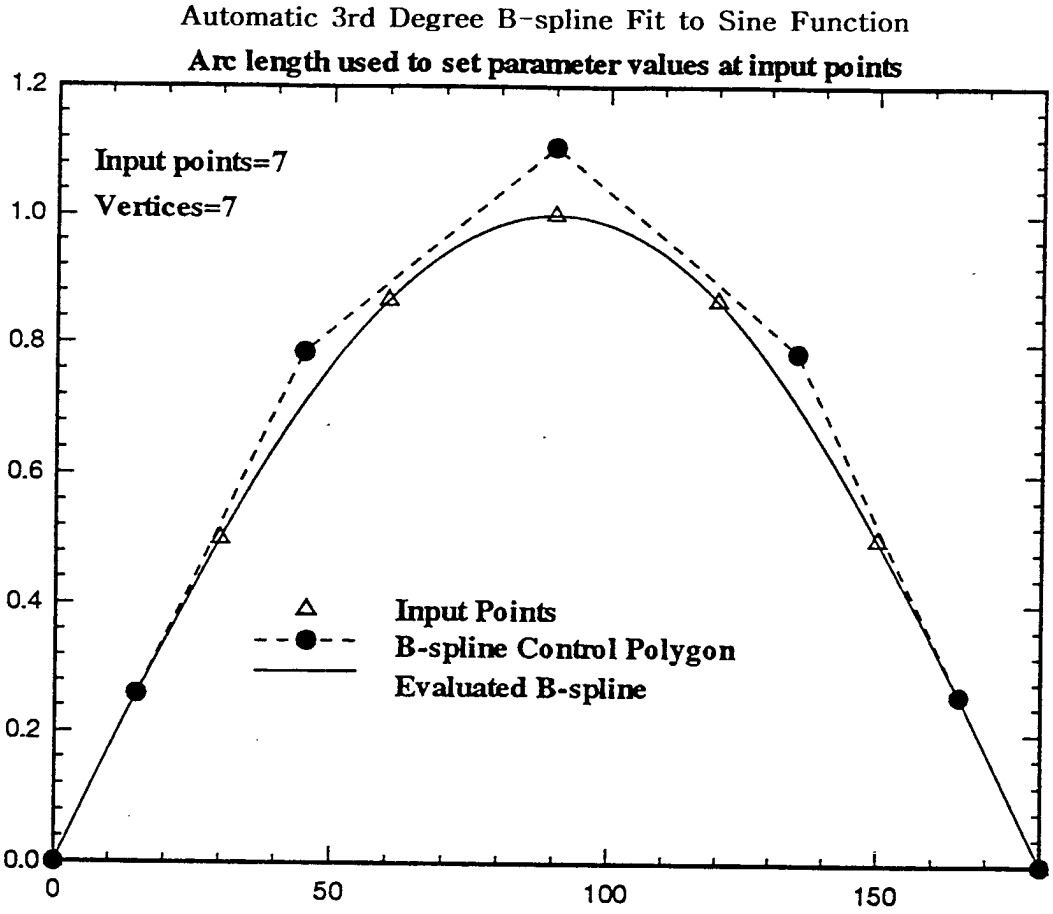


Figure 60: Third degree B-spline fit to sine function using 7 base points. The parameter values assigned to each base point are proportional to the polygonal arc length.

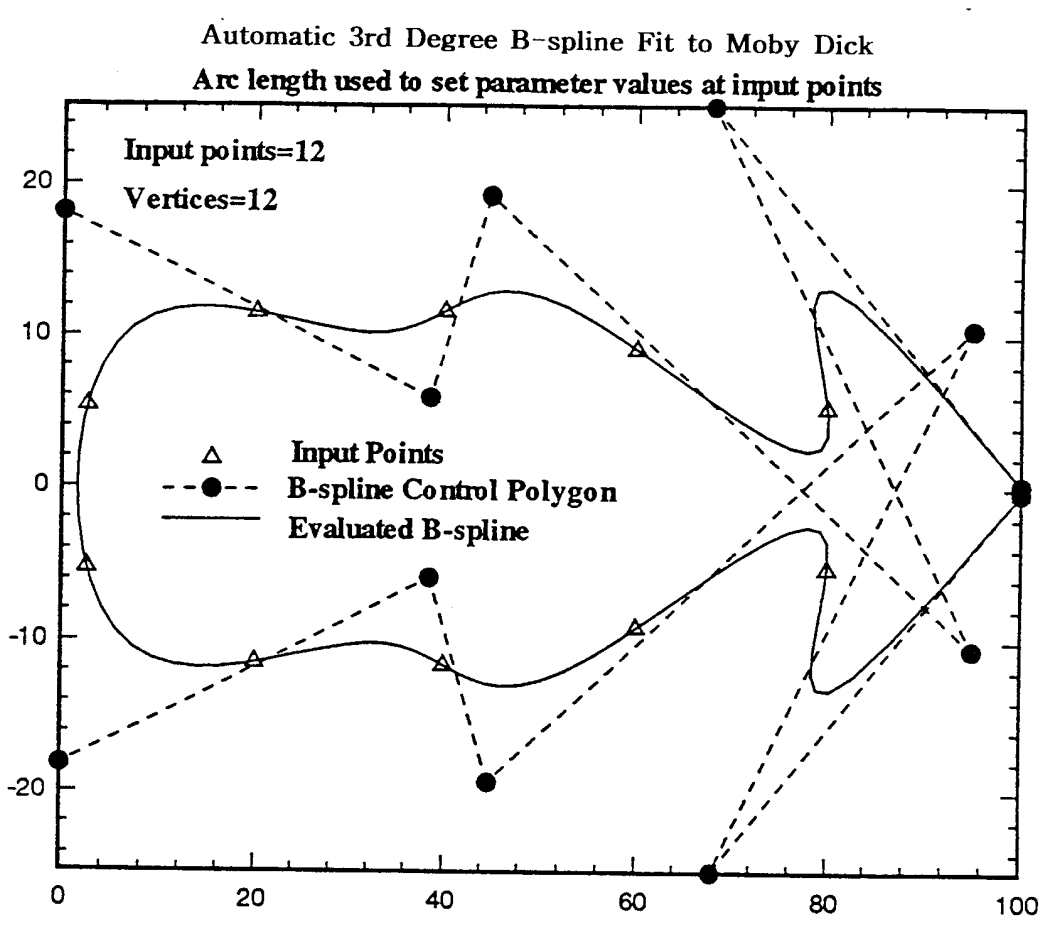


Figure 61: Third degree B-spline fit to an NACA0024 foil section using 12 base points. The parameter values assigned to each base point are proportional to the polygonal arc length.

9.1 Fitting Foils and other Slender Objects

It is clear from figure 61 that the problem is due to the fact that the x coordinates of the vertices are not monotonic, and that large inflections in the B-spline are created when the control polygon doubles back on itself. If you were fitting the data manually, it would be intuitive to space the vertices in a regular fashion along the x axis from the leading to trailing edges, and then to adjust the y coordinates until the curve looked right.

This procedure can be automated, and it turns out that it works very well for objects that are slender. If we look again at equation 87, we can see that if the base points *and* the x coordinates of the vertices are given, we can solve for the values of the parameter u at each base point,

$$x(u_n) \equiv x_n = \sum_{i=0}^{N^v-1} \xi_i \mathcal{N}_i^p(u_n) \tag{90}$$

Since the basis functions are polynomials of degree p, equation 90 is non-linear. However, it can be solved rapidly using Newton's method, starting with any reasonable guess for the parameter u,

$$(u_n)_{j+1} = (u_n)_j - \frac{(x_n)_j - x_n}{\frac{dx}{du}} \tag{91}$$

where $(u_n)_j$ is the current j-th guess for the parameter u corresponding to the n-th base point, $(x_n)_j$ is the value of x obtained from equation 90 using this value of u and x_n is the desired value for the n-th base point. The derivative $\frac{dx}{du}$

$$\frac{dx}{du} = \sum_{i=0}^{N^v-1} \xi_i \frac{d}{du}\left(\mathcal{N}_i^p(u_n)\right) \tag{92}$$

can be computed very easily, and is an exact (to machine precision) result.

Once the parameter values are known, then a set of simultaneous equation for the y coordinates of the base points can be solved. This part of the procedure is the same as the "textbook" method described in the preceding section.

One last refinement needs to be mentioned. Note that the round leading edge of the foil will not touch the control polygon if the degree of the B-spline is greater than 2. In order to place the leading edge at the desired x location, the x coordinates of the pair of vertices closest to the leading edge must be moved slightly. This adjustment can also be automated, using Newton's method as before. The details will not be presented here.

Figure 62 shows a B-spline fit to the same NACA0024 foil section as before, again using 12 vertices. The result is now very satisfactory. Note the position of the

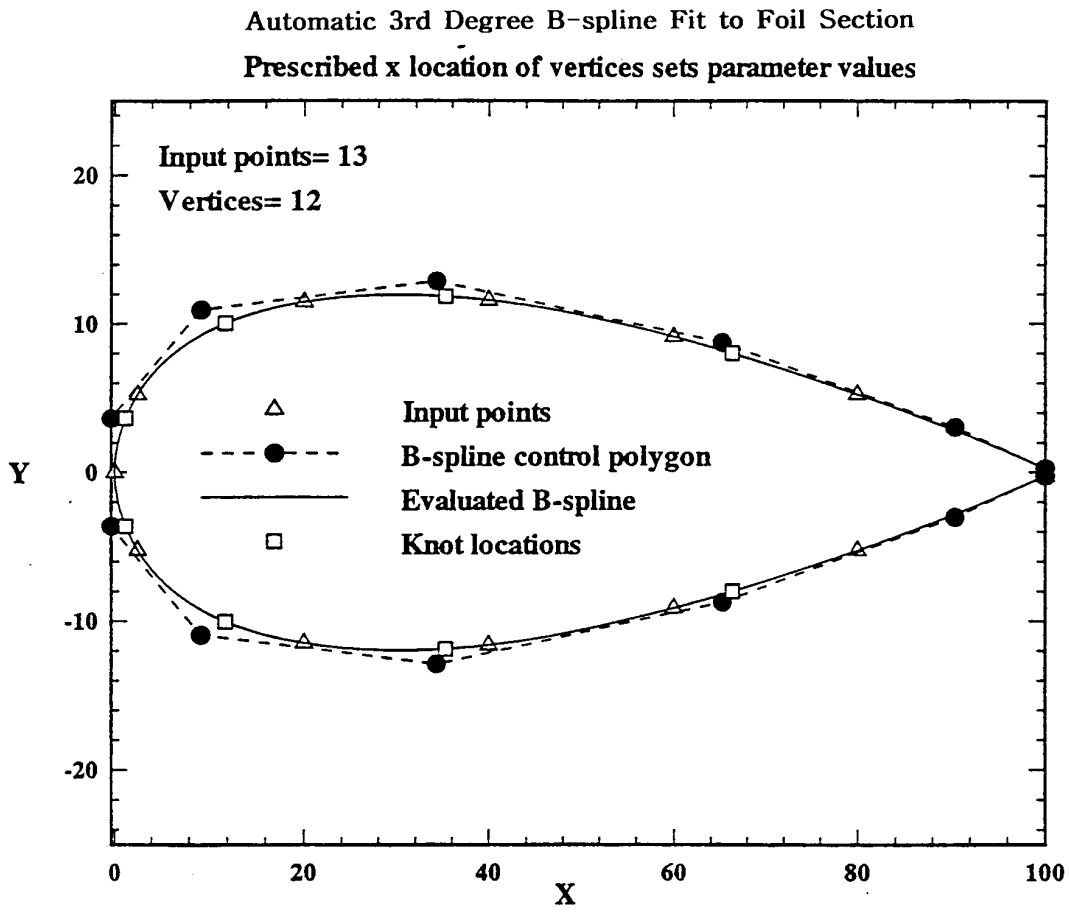


Figure 62: Third degree B-spline fit to an NACA0024 foil section using 12 base points. The parameter values assigned to each base point are determined by setting the ξ coordinates of the control polygon vertices.

knots (indicated by square symbols), which are evidently concentrated near the high curvature region near the leading edge.

10 NUMERICAL INTEGRATION

10.1 INTRODUCTION

There are many applications in Ocean Engineering which require the calculation of *mass properties* of arbitrary shapes— namely the area, centroid and moment of inertia. In mathematical terms, this means that given an arbitrary function $y(x)$, we need to be able to determine

$$I_0(a,b) = \int_a^b y(x)dx \tag{93}$$

$$I_1(a,b,x_0) = \int_a^b (x-x_0)y(x)dx \tag{94}$$

$$I_2(a,b,x_0) = \int_a^b (x-x_0)^2 y(x)dx \tag{95}$$

where (a,b) are the lower and upper limits of integration, respectively, and x_0 is the point about which the first and second moments are being taken. The position of the centroid and the value of the moment of inertia can then be expressed in terms of these three basic integrals.

An obvious application is the determination of the buoyant forces and moments acting on floating or submerged objects. But there are numerous other applications where integrals of arbitrary functions are needed. While these can be carried out analytically in some cases, more frequently one must resort to numerical methods.

Everyone is familiar with one or more elementary numerical integration formulas. However, in a computer environment, it may be advisable to use numerical integration methods which are more powerful than the ones which you are familiar with. This section of notes provides a brief look at a variety of numerical integration formulas which range from slow and simple to fast and complicated. To derive them, we will revisit Lagrange interpolation methods developed at the beginning of the term.

10.2 TRAPEZOIDAL RULE

The simplest numerical integration formula may be found by approximating the function $y(x)$ by a straight line in the interval (a,b). In this case, an approximation to equation (93) can be written down immediately as

$$I_0(a,b) \approx \frac{b-a}{2}[y(a)+y(b)] = \frac{h}{2}[y(a)+y(b)] \tag{96}$$

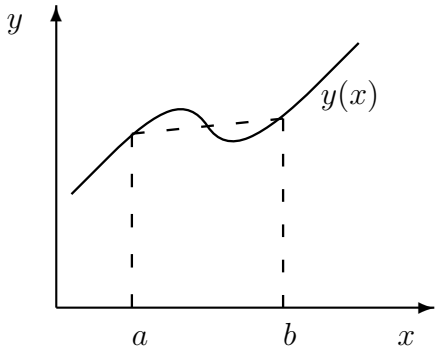


Figure 63: Linear approximation of $y(x)$

and this is known as the trapezoidal rule. This result came much too easily, so we will now seek a more complicated way to derive it.

What we really did, was to approximate the unknown function $y(x)$ by a polynomial of degree one, we then integrated the resulting polynomial to obtain a result explicitly as the product of a set of weight functions, $W^0_{i,k}$ and the values of the function at a set of $k+1$ base points, y_0 and y_1. The subscript k then denotes the degree of the approximating polynomial, while the superscript 0 indicates that these are the weight functions for I_0. The trapezoidal rule weight functions for the first moment would then be $W^1_{i,k}$. This notation is a little awkward, and the extra sub and superscripts will generally be omitted whenever it is clear which order rule and which moment is being considered.

We already know how to obtain a polynomial approximation to a function explicitly in terms of the values of the function at selected base points using Lagrange interpolation. For convenience in this case, suppose we set $x_0 = 0$ and $x_1 = h$. The two Lagrange interpolation coefficients are then

$$L_0 = -\frac{x-h}{h} \qquad L_1 = \frac{x}{h} \tag{97}$$

and the first degree polynomial approximation is

$$y(x) \approx L_0 y_0 + L_1 y_1 \tag{98}$$

and the integral is

$$I_0 \approx y_0 \int_0^h L_0(x)dx + y_1 \int_0^h L_1(x)dx = y_0 W^0_{0,1} + y_1 W^0_{1,1} \tag{99}$$

We can therefore calculate the weights as follows:

$$W_0 = \frac{-1}{h}\int_0^h (x-h)dx = \frac{-1}{h}\left|\frac{x^2}{2} - hx\right|_0^h = \frac{h}{2} \tag{100}$$

$$W_1 = \frac{1}{h}\int_0^h x dx = \frac{1}{h}\left|\frac{x^2}{2}\right|_0^h = \frac{h}{2} \tag{101}$$

This was certainly harder than the first way of obtaining the answer, but we now have a systematic procedure for obtaining the weights for any moment and for any order integration formula based on the Lagrange interpolation polynomials. For example, the weights for the first moment of a function approximated by a first degree polynomial taken about the point $x = 0$

$$W_0^1 = \frac{-1}{h}\int_0^h (x^2 - hx)dx = \frac{-1}{h}\left|\frac{x^3}{3} - \frac{hx^2}{2}\right|_0^h = \frac{h^2}{6} \tag{102}$$

$$W_1^1 = \frac{1}{h}\int_0^h x^2 dx = \frac{1}{h}\left|\frac{x^3}{3}\right|_0^h = \frac{h^2}{3} \tag{103}$$

Similarly, the weights for the second moment taken about the point $x = 0$ can be shown to be

$$W_0^3 = \frac{h^3}{12} \qquad W_1^3 = \frac{h^3}{4} \tag{104}$$

10.3 TRAPEZOIDAL RULE WITH MULTIPLE INTERVALS

Obviously a first degree polynomial approximation to a function is not very accurate, unless the function happens to be a straight line. However, a piecewise approximation, consisting of a sequence of straight line approximations, can be made as accurate as you want by taking a sufficiently large number of intervals. As we have indicated before, most computer graphic displays are done this way.

Suppose we take the interval from (a, b) and divide it into N equally spaced sub intervals of width $h = (b-a)/N$. The integral can then be written down immediately as the sum of the integrals over each sub interval

$$\int_a^b y(x)dx \approx \frac{h}{2}\left[(y_0 + y_1) + (y_1 + y_2) + (y_2 + y_3) + \ldots (y_{N-1} + y_N)\right] \tag{105}$$

This can be simplified to give the result

$$\int_a^b y(x)dx \approx h\left[\frac{1}{2}y_0 + y_1 + y_2 + y_3 + \cdots + y_{N-1} + \frac{1}{2}y_N\right] \tag{106}$$

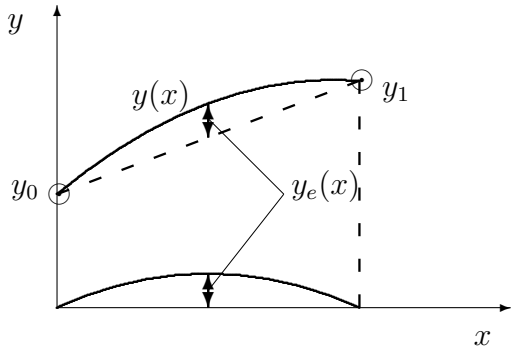


Figure 64: Estimating the error in the integral

10.4 ERROR IN TRAPEZOIDAL RULE

We can obtain an estimate of the error in the trapezoidal rule by considering that the true function is really a polynomial of degree two, and calculating the difference in the integral of the first degree and second degree polynomials. This is illustrated in figure 64. Considering again that the two base points are located at $x = 0$ and $x = h$, respectively, the difference between the first degree and second degree approximations to the true function must be a parabola of the form

$$y_e(x) = cx(x - h) \tag{107}$$

where c is a constant to be determined. The difference is, of course, zero when $x = 0$ and $x = h$. The constant c can be related to the second derivative of the function $y(x)$ as follows,

$$y_e^{'}(x) = c(2x - h) \qquad y_e^{''}(x) = 2c \qquad y_e(x) = \frac{y_e^{''}}{2}x(x - h) \tag{108}$$

Noting that $y_e^{''} = y^{''}$, the error in the integral from $(0, h)$ is

$$E = \frac{y^{''}}{2}\int_0^h x(x - h)dx = \left|\frac{y^{''}}{6}x^3 - \frac{y^{''}hx^2}{4}\right|_0^h = -\frac{y^{''}h^3}{12} \tag{109}$$

If the true function were a polynomial of degree higher than two, a much more rigorous proof would show that equation (109) was still correct, but with the value of the second derivative evaluated at some point within the interval.

If we do not know much about the function to be integrated, and in particular, cannot estimate its second derivative, equation (109) would not seem to be of much help. However, an important piece of information is that the error is proportional to the cube of the interval size.

Suppose that we take the original interval and divide it into two sub intervals, as discussed in the preceding section. Assuming that the unknown second derivative is roughly constant, the error in the integral in each sub interval is reduced by $2^3 = 8$. However, there are now two intervals, so that the total error is reduced by 4.

In general, the total integration error with trapezoidal rule will be proportional to $1/N^2$.

As an example, consider the integral of $\sin(x)$ in the interval from zero to one radian. We can, of course, write down the exact answer,

$$\int_0^1 \sin(x)dx = -\cos x|_0^1 = 0.45970 \tag{110}$$

With one interval, $h = 1$ and the trapezoidal rule result would be

$$\int_0^1 \sin(x)dx \approx \frac{1}{2}\left[\sin(0) + \sin(1)\right] = 0.42074 \tag{111}$$

and the corresponding error would be 0.03896. Repeating the calculation with the number of intervals doubled each time gives the following results,

N	$INTEGRAL$	$ERROR$	$.03896/N^2$
1	.42074	.03896	.03896
2	.45008	.00962	.00974
4	.45730	.00240	.00244
8	.45910	.00060	.00061
16	.45955	.00015	.00015
32	.45966	.00004	.00004
64	.45969	.00001	.00001
128	.45970	.00000	.00000

Thus we see that the trapezoidal rule converges to the right answer eventually, and that the rate of convergence is just about as predicted. We can get three significant figure accuracy (which is enough for many purposes) with only eight intervals, but it took 128 intervals to obtain five significant figure accuracy. There must be a better way. There is! Read the next section.

10.5 SIMPSON'S RULE

The next logical step is to increase the degree of the approximating polynomial from one to two, which therefore requires the use of three base points. For convenience,

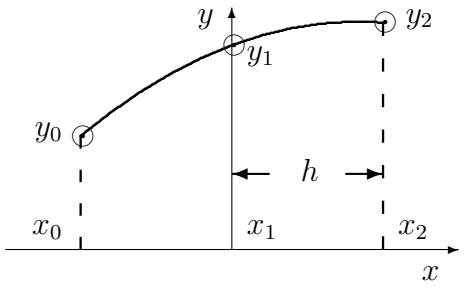


Figure 65: Arrangement of base points for Simpson's Rule

consider that the three base points are at $x = -h$, $x = 0$ and $x = h$ as shown in figure 65

The polynomial passing through the three base points can again be expressed in terms of Lagrange interpolation polynomials as

$$y(x) \approx L_0(x)y_0 + L_1(x)y_1 + L_2(x)y_2 \tag{112}$$

where

$$L_0(x) = \frac{1}{2h^2}(x^2 - hx) \quad L_1(x) = \frac{-1}{h^2}(x^2 - h^2) \quad L_2(x) = \frac{1}{2h^2}(x^2 + hx) \tag{113}$$

Integrating the Lagrange polynomials given in equation (113) over the interval $(-h, h)$ gives the desired weight functions,

$$W^0_{0,2} = \frac{h}{3} \quad W^0_{1,2} = \frac{4h}{3} \quad W^0_{2,2} = \frac{h}{3} \tag{114}$$

so that the equation for Simpson's rule is

$$\int_{-h}^{h} y(x)dx \approx \frac{h}{3}(y_0 + 4y_1 + y_2) \tag{115}$$

For example, we can evaluate the integral of $\sin(x)$ as before. In this case $h = 1/2$ and

$$\int_0^1 \sin(x)dx \approx \frac{1}{6}\left[\sin(0) + 4\sin(0.5) + \sin(1)\right] = 0.45986 \tag{116}$$

Since the correct answer is 0.45970 so that Simpson's rule overestimates the integral, but only by 0.00016. Comparing this with the table of results for the Trapezoidal rule, we see that sixteen intervals would be required to obtain the same accuracy as is obtained with one application of Simpson's rule. The former required seventeen evaluations of the sine function, while the latter required only three. Simpson's rule would therefore be about six times as fast in this case.

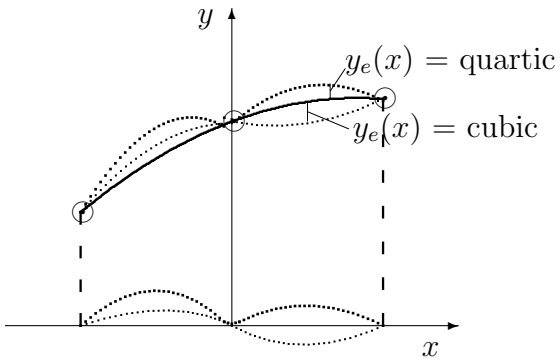


Figure 66: Error in Simpson's Rule

10.6 SIMPSON'S RULE WITH MULTIPLE INTERVALS

We can obtain a multiple interval rule as before by simply adding up the contributions of each sub interval,

$$\int_a^b y(x)dx \approx \frac{h}{3}\left[(y_0 + 4y_1 + y_2) + (y_2 + 4y_3 + y_4) + \ldots (y_{N-2} + 4y_{N-1} + y_N)\right] \quad (117)$$

This can again be simplified to give the final result

$$\int_a^b y(x)dx \approx \frac{h}{3}\left[y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots 2y_{N-2} + 4y_{N-1} + y_N\right] \quad (118)$$

The interval is $h = (b-a)/N$ as before, but since each application of Simpson's rule spans two intervals, N must be an even integer, or equivalently, the total number of base points must be odd.

10.7 ERROR IN SIMPSON'S RULE

An error formula can be derived in exactly the same was as for the Trapezoidal rule, except for one surprise. The procedure is illustrated in figure 66.

Since the approximating function is a polynomial of degree two, we would expect that the error in the integral could be represented by the integral of a cubic function which is zero at the three base points.

$$y_e(x) = cx(x-h)(x+h) \quad (119)$$

but clearly, the integral of this function is zero. In other words, even though we are using a polynomial of degree two as the approximating function, Simpson's rule is

exact for any polynomial of degree up to three. The lowest degree polynomial which produces an error is four,

$$y_e(x) = cx^2(x-h)(x+h) \tag{120}$$

Differentiating equation (120) four times lets us express the unknown constant c in terms of the fourth derivative,

$$y_e^{'} = 4cx^3 - 2h^2x \quad y_e^{''} = 12cx^2 - 2h^2 \quad y_e^{'''} = 24cx \quad y_e^{iv} = 24c \tag{121}$$

We can then obtain the desired expression for the error

$$E = \int_{-h}^{h} \frac{y^{iv}}{24}(x^4 - h^2x^2)dx = -\frac{h^5 y^{iv}}{90} \tag{122}$$

Thus we see that compared with the Trapezoidal rule, the error is smaller, and decreases much faster with h, provided that the higher derivatives do not do something unpredictable.

10.8 Romberg Integration

Romberg integration is unlike any of the previous methods covered. One does not fit a polynomial of degree $N-1$ through N base points. Instead, one obtains a succession of approximations to the integral, which can be continued indefinitely until a prescribed level of accuracy is obtained. Each approximation is derived from the preceding one, with the addition of a prescribed number of new evaluations of the function to be integrated. Romberg integration requires that we be able to evaluate the integrand at more and more points until convergence is achieved. The method is therefore only effective in a computer based environment. However, the method can be illustrated with simple examples which can be computed manually.

The first approximation to the integral is obtained simply by applying the trapezoidal rule with one interval,

$$\int_{x_l}^{x_r} y(x)dx \approx \frac{h}{2}(y_0 + y_1) = A_{0,1} \tag{123}$$

where $h = x_r - x_l$, $y_0 = y(x_l)$ and $y_1 = y(x_r)$. Successive approximations to the integral will be generated from a matrix $A_{n,m}$, where $A_{0,1}$ is the leading element. The first index of each matrix element indicates the number of sub-intervals (expressed as a power of two). In this case, we have $2^0 = 1$ sub-intervals. The second index denotes, in some loose sense, the degree of the approximation to the integral. Here we are using a first degree polynomial, so the second index is 1.

So far, we have only succeeded in making a simple thing complicated. We will now make it even more complicated. We can double the number of sub-intervals by adding a third point, y_2 at the mid-point $x_2 = x_l + h/2$. This gives us a new approximation to the integral

$$\int_{x_l}^{x_r} y(x)dx \approx \frac{h}{4}(y_0 + 2y_2 + y_1) = A_{1,1} \tag{124}$$

where the first index of the matrix element $A_{n,m}$ is now $2^1 = 2$, denoting two sub-intervals. We can compute $A_{1,1}$ more efficiently if we make use of the value of $A_{0,1}$ which was computed previously,

$$A_{1,1} = \frac{1}{2}A_{0,1} + \frac{h}{2}y_2 \tag{125}$$

This doesn't seem like much of a gain, but it makes a big difference as the iterations progress with more sub-intervals. Now comes the clever part. We know that the error in the trapezoidal rule should be (approximately) inversely proportional to the square of the spacing. Thus the error in the second estimate should be one quarter that of the first estimate. Suppose that the exact answer is A and that the error with one interval is e. We can then write

$$\begin{aligned} A &= A_{0,1} + e \\ A &\approx A_{1,1} + e/4 \end{aligned}$$

Eliminating the error, e, gives the following result,

$$A \approx \frac{4A_{1,1} - A_{0,1}}{3} \equiv A_{0,2} \tag{126}$$

So far we have three different approximations to the integral, with the third one presumably the best. These can be arranged in matrix form,

$A_{0,1}$	$A_{0,2}$
$A_{1,1}$	

The next step is to double the number of intervals again by adding two new points, $x_3 = x_l + h/4$ and at $x_4 = x_r - h/4$. The trapezoidal rule approximation to the integral is,

$$A_{2,1} = \frac{h}{8}(y_0 + 2y_3 + 2y_2 + 2y_4 + y_1) = \frac{1}{2}A_{1,1} + \frac{h}{4}(y_3 + y_4) \tag{127}$$

Applying the same error formula (equation (126)) to the latest result of doubling the number of intervals provides us with a fifth estimate of the integral,

$$\frac{4A_{2,1} - A_{1,1}}{3} \equiv A_{1,2} \tag{128}$$

and we can add this to our collection,

$A_{0,1}$	$A_{0,2}$
$A_{1,1}$	$A_{1,2}$
$A_{2,1}$	

Now the second column represents values of the integral which would both be exact if the integrand were a second degree polynomial. From the error analysis for Simpson's rule, we saw that halving the interval would reduce the estimated error by $2^4 = 16$. We can use this result to obtain an even better estimate,

$$\begin{aligned} A &= A_{0,2} + e \\ A &\approx A_{1,2} + e/16 \end{aligned}$$

Eliminating the error, e, gives the following result,

$$A \approx \frac{16A_{1,2} - A_{0,2}}{15} \equiv A_{0,3} \tag{129}$$

which we can add as the third element in the first row of the matrix,

$A_{0,1}$	$A_{0,2}$	$A_{0,3}$
$A_{1,1}$	$A_{1,2}$	
$A_{2,1}$		

Without writing down any more steps, the pattern should now be evident. If we double the number of intervals again, we will need to introduce four new points. The trapezoidal rule estimate of the integral based on the total of eight points will be $A_{3,1}$. Applying the equivalent of equation (126) will give us $A_{2,2}$, and the equivalent of equation (129) will give us $A_{1,3}$. The two entries in the third column can then be used to derive an even more exact value, $A_{0,4}$.

This procedure can be tested by evaluating the now familiar

$$\int_0^1 \sin(x)dx = 0.45970 \tag{130}$$

Here are the calculations for each of the elements of $A_{n,m}$

$$\begin{aligned}
A_{0,1} &= \frac{1}{2}(\sin(0) + \sin(1)) = 0.42074 \\
A_{1,1} &= \frac{1}{2}A_{0,1} + \frac{1}{2}\sin(0.5) = 0.45008 \\
A_{0,2} &= \frac{4A_{1,1} - A_{0,1}}{3} = 0.45986 \\
A_{2,1} &= \frac{1}{2}A_{1,1} + \frac{1}{4}(\sin(0.25) + \sin(0.75)) = 0.45730 \\
A_{1,2} &= \frac{4A_{2,1} - A_{1,1}}{3} = 0.45971 \\
A_{0,3} &= \frac{16A_{1,1} - A_{0,2}}{15} = 0.45970
\end{aligned}$$

0.42074	0.45986	0.45970
0.45008	0.45971	
0.45730		

The result is exact to five significant figures, and only required five evaluations of the sine function.

This procedure can be readily programmed. Shown below is a C++ code which contains the general Romberg algorithm, but is specialized to treat the example of the sine function. In real life, the procedure can be set up to continue iterations until a specified accuracy is obtained. Here the program is set up just to do six iterations, and then to stop. Note that ten significant figure accuracy is achieved by the fifth iteration. The first three iterations agree with the hand calculation shown above.

```
//*************** ROMBERG INTEGRATION OF SIN(X) ***********************
#include <fstream.h>
#include <math.h>
#include "dmcdebug.h"

void main()
{
  int n, m, nk, mk, iz, nz;
  double xleft, xright, h, ba, sum, x, f;
  CArray2d<double> A(6,6);

  for (n=0; n<6; n++) {
    for (m=0; m<6; m++) A[n][m] = 0.0;
  }
  xleft   = 0.0;
  xright  = 1.0;
  h       = xright - xleft;
  ba      = 0.5 * h;
  A[0][0] = ba * (sin(xleft) + sin(xright));

  mk = 1;
  for (nk=0; nk<5; nk++) {
    sum    = 0.0;
    x      = xright - ba;
    for (m=0; m<mk; m++) {
      sum += sin(x);
      x   -= h;
    }

    A[nk+1][0] = 0.5 * A[nk][0] + sum*ba;
    h          = ba;
    ba         = 0.5 * ba;
    f          = 1.0;
    for (iz=0; iz<=nk; iz++) {
      nz          = nk - iz;
      f          *= 4.0;
      A[nz][iz+1] = (f*A[nz+1][iz] - A[nz][iz])/(f-1.0);
    }
    mk += mk;
  }
  ofstream ouf("romberg.out");
  ouf.setf(ios::fixed | ios::showpoint);
  ouf.precision(10);
  for (n=0; n<6; n++) {
    for (m=0; m<6; m++) {ouf << setw(13) << A[n][m];}
```

```
    ouf << endl;
  }
}
```

```
Output from Romberg.cpp

 0.4207354924 0.4598621899 0.4596974486 0.4596976942 0.4596976941 0.4596976941
 0.4500805155 0.4597077449 0.4596976904 0.4596976941 0.4596976941 0.0000000000
 0.4573009376 0.4596983188 0.4596976941 0.4596976941 0.0000000000 0.0000000000
 0.4590989735 0.4596977331 0.4596976941 0.0000000000 0.0000000000 0.0000000000
 0.4595480432 0.4596976966 0.0000000000 0.0000000000 0.0000000000 0.0000000000
 0.4596602832 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
```

10.9 Newton-Cotes Integration

One can derive integration formulas based on polynomials of any order, with Trapezoidal and Simpson's rule being two examples. The family of formulas with equally spaced ordinates is known as Newton-Cotes. The higher order formulas are, in principle, more accurate. However, keeping in mind the problems of polynomial interpolation of functions containing slope discontinuities, a high order rule might not be the best for some applications.

The four point Newton-Cotes rule is also known as Simpson's second rule,

$$\int_{x_l}^{x_r} y(x)dx = \frac{3h}{8}\left(y_0 + 3y_1 + 3y_2 + y_3\right) \tag{131}$$

where $h = (x_r - x_l)/3$. The error can be shown to be

$$E = \frac{3h^5}{80}y^{iv} \tag{132}$$

which is not much better than Simpson's first rule. Its principal advantage is in getting you out of the predicament of having an odd number of intervals in a multiple application of Simpson's first rule. By throwing in one four point (three interval) segment, the remaining number of intervals will be even.

The approximation to the integral of the sine function according to Simpson's second rule is,

$$\int_0^1 \sin(x)dx \approx \frac{1}{8}\left[\sin(0) + 3\sin(0.33333) + 3\sin(0.66667) + \sin(1)\right] = 0.45977 \tag{133}$$

10.10 Tchebycheff Rules

We now move on to the case where the true function is still approximated by a polynomial through a set of base points, but that the positions of the base points may not necessarily be evenly spaced, nor need they include the limits of the region to be integrated. This extra flexibility can be used for two purposes. The first is to achieve a simplification of the rule by requiring that all the weights be equal. The second is to increase the accuracy of the rule.

Specifying the former leads to Tchebycheff's rule, the latter leads to Gauss' rule, which we will cover briefly in the next section.

Having equal weights in an integration rule used to be a convenience in carrying out volume calculations from traditional lines drawings. One simply calculated the area of each station, possibly with a mechanical device known as a planimeter, and added them up and multiplied by a single constant to get the volume. As a more extreme example, one could obtain the centroid of a volume by cutting out the Tchebycheff spaced sections out of cardboard, and the balancing them on a pin!

Of course, this required the preparation of a lines drawing with stations which were not uniformly spaced, but spaced in a particular way depending upon the order of the Tchebycheff rule which was to be used.

The three point Tchebycheff rule, for example, is as follows:

$$\int_{-h}^{h} y(x)dx = \frac{2h}{3}\left[y\left(\frac{-h}{\sqrt{2}}\right) + y(0) + y\left(\frac{h}{\sqrt{2}}\right)\right] \tag{134}$$

While it is difficult to find the necessary positions of the ordinates to obtain constant weights, (particularly as the number of base points becomes large), it is easy to find the weights and to verify that they are constant using the Lagrange interpolation procedure developed in the preceding sections. If the number of points is eight, or is greater than ten, there is no solution possible for the ordinates which would produce constant weights. This is of little practical concern, since one would generally not chose to use any integration rule of such a high order.

We can illustrate the use of the three point formula using the same example as before,

$$\int_0^1 \sin(x)dx \approx \frac{1}{3}\left[\sin(0.14645) + \sin(0.5) + \sin(0.85355)\right] = 0.45966 \tag{135}$$

Note that the positions of the ordinates had to be scaled from the interval $(-h, h)$ to $(0, 1)$. The error of 0.00004 which is even better than with Simpson's rule.

10.11 Gauss' Rule

If we allow both the weights and the positions of the ordinates to be adjusted for maximum accuracy, an even better set of integration formulas can be derived. We saw in the derivation of the error for Simpson's rule that the integral of any third degree polynomial with zero's at the three equally spaced base points was zero. This made Simpson's rule one degree more accurate than expected. The idea in Gauss integration is to locate the base points in such a way that the integral of even higher degree polynomials with zeros at the base points vanishes. These turn out to be the zero's of the Legendre polynomial of a particular order which depend on the number of points. The derivation, needless to say, is too lengthy to be included here, but may be found in many numerical analysis texts.

The three point Gauss rule is,

$$\begin{aligned}\int_{-h}^{h} y(x)dx &= h[0.55555y(-0.77460h) + 0.88889y(0) \\ &\quad +0.55555y(0.77460h)]\end{aligned}$$

This looks almost the same as the three point Tchebycheff rule, except that the ordinates are slightly closer to the ends of the interval, and that the three weights are different. The difference in computing effort between the two rules (and for Simpson's rule, for that matter) is negligible. Here we go again with the sine function example,

$$\begin{aligned}\int_{0}^{1} \sin(x)dx &\approx \frac{1}{2}[0.55555\sin(0.11270) + 0.88889\sin(0.5) \\ &\quad +0.55555\sin(0.88730)] = 0.45970\end{aligned}$$

This is obviously the time to quit, since the error is now zero (at least to five significant figures)! The same accuracy required 128 intervals with trapezoidal rule, so the accuracy of Gauss' rule in this example is certainly impressive.