UNITED STATES DEPARTMENT OF THE INTERIOR

GEOLOGICAL SURVEY

Review of Three Cubic Spline Methods in Graphics Applications

by

Gerald I. Evenden[1]

Open-File Report 89–19

[1]Woods Hole, Mass.

1989

CONTENTS

1. Abstract **1**
2. Introduction **1**
3. Basic concepts **1**
4. Behavior of spline curves **2**
5. Determination of spline coefficients **4**
   5.1 Osculatory spline knot derivatives **5**
   5.2 Akima spline knot derivatives **5**
   5.3 Standard spline knot derivatives **5**
6. Parametric splines **6**
7. Bezier splines **6**
8. C spine procedures **8**
   8.1 Usage of spline procedures **8**
9. Conclusions **9**
10. References **10**
11. Appendix **11**

FIGURES

1. Nonparametric spline curves for a step function **2**
2. Effects of various end conditions at the beginning of a curve for a nonperiodic standard spline **3**
3. Nonparametric nonperiodic and periodic spline curves for a square wave function **4**
4. Parametric spline curves for an open and closed box **4**
5. Representative sets of Bezier control points and their representative interpolated curves **7**
6. Bezier control points determined for a standard parametric spline curve determined for the circled knots **8**

TABLE

1. Knot derivatives for test data set **9**

LISTINGS

1. Example C test program executing spline procedures **11**
2. C procedure to determine knot derivatives by Akima's method **12**
3. C procedure to determine knot derivatives by the osculatory method **12**
4. C procedure to determine knot derivatives by the standard nonperiodic spline **13**
5. C procedure to determine knot derivatives by the standard periodic spline **14**

Review of Three Cubic Spline Methods in Graphics Applications

by Gerald I. Evenden

1. Abstract

Three basic cubic spline methods—Akima's, osculatory, and standard—are reviewed with respect to graphics applications. Basic concepts are discussed and performance comparisons are made as well as a complete, practical mathematical formulary for determination of spline curves. The Bezier method is also discussed as a general means for description of cubic line segments. C language source code for determination of spline derivatives, test program, and results are included.

2. Introduction.

In graphic operations associated with presentation of scientific and engineering information, it is often desirable to interpolate a smooth, continuous line through a set of data points. A single function can usually be determined for a data set of limited size and evaluated for intermediate smoothing values, but determinating the function for larger data sets can be difficult and (or) time consuming. The results, especially for the purposes of graphics, are often less than desirable. As an alternative, a relatively simple, easily determined set of functions for each interdata point interval can be employed that establishes a piecewise description of the line.

The purpose of this report is to review a common form of piecewise interpolation—the cubic spline function—and to define associated terminology, methods of computation, and examples of performance in graphic operations. The cubic Bezier function is also discussed as a practical mechanism for communicating the properties of spline curves in the graphics environment.

Readers involved only in the application of existing software employing splines such as the Unix utility program **spline** or the author's expanded version **xspline** (Evenden, 1988) will be primarily interested in the preliminary sections dealing with basic concepts and behavior of splines. The formulary for the determination of the cubic spline coefficients is included for programmers who need to develop non-FORTRAN versions of the discussed splines. Ancillary to the formulary, C compiler language procedures are included as well as a test program and resultant output.

3. Basic concepts

The original spline is a draftsman's tool, consisting of a thin flexible bar held in place by weighted pins, called ducks, that is used to draft smooth lines created by the natural curvature of the bar flexed between the constraining pins. The mathematical equivalent of the mechanical spline is described by Ahlberg et al. (1967) and consists of a set of polynomials between each pair of adjacent knots (ducks) that not only generates a curve that passes through the knots but has continuous first and higher derivatives. For the purposes of this report, this mathematical definition of the spline will be referred to as the standard spline.

In graphics applications, the standard spline polynomials are generally third degree where the first and second derivatives are continuous. Continuity of the first derivative is the spline property of principle interest in graphics applications because it determines the smoothness of the curve passing through the knots and thus enhances the visual appearance of the graphic. Nongraphic spline applications, such as numerical integration or differentiation, are often concerned with continuity of the higher derivatives, but they have no effect on the visual appearance of the curve.

One undesirable aspect of determining the polynomial coefficients for a standard spline is that it is computationally complex and for graphics purposes, other, less computationally intensive, methods may produce acceptable results. In addition, the minimum curvature property of standard splines also produces curves with "wiggles" which may be deemed unnatural in appearance and unacceptable for certain graphic presentations.

Akima (1970, 1972) developed a cubic spline interpolation method that reduces the wiggles of the standard spline, producing a curve that tends to be more natural in appearance or, at least, a curve that approximates one which might be expected by manual drafting. Akima also includes in the same paper the osculatory interpolation method that produces curves that are often intermediate to the standard spline and his method. In both of these methods, the polynomial coefficients are readily determined from local knot values.

It should be noted that Cline (1974) developed a tensioned spline system that provides a continuous degree of control over the degree of wiggle of the curve. But this method requires hyperbolic functions and not simple cubics.

Before proceeding, two additional terms related to spline functions should be mentioned: periodic and parametric. Periodic splines assume that the curve is cyclically repeated at both ends of the given set of knot points and that the first and last ordinal values are identical. For example, signal wave forms are often periodic in nature. Parametric splines assume that the ordinate and abscissa values are functions of a third variable or *parameter*. Contour lines are typical examples of parametric curves where open and closed contour lines can be respectively considered nonperiodic and periodic in nature.

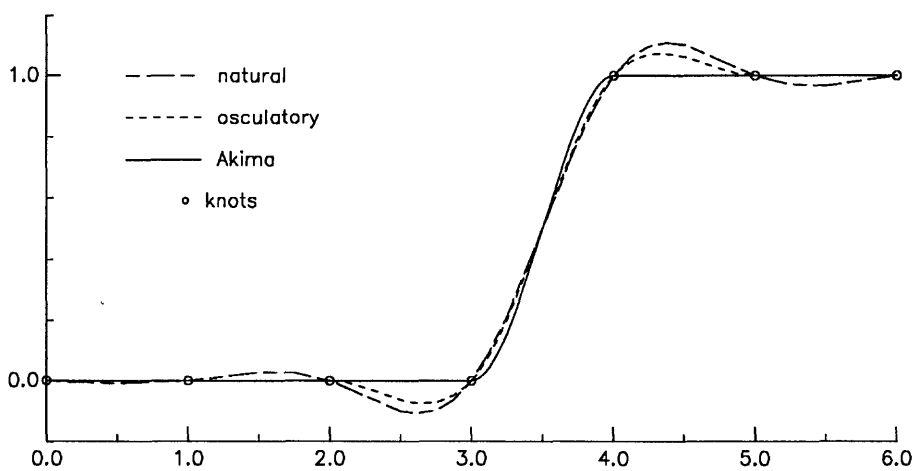

Figure 1. Nonparametric spline curves for a step function.

4. Behavior of spline curves

One of the most descriptive means of demonstrating the action of a nonparametric interpolation method is with a simple step function as shown in Figure 1. The wiggles or overshoot and damped oscillation of the standard spline are quite

evident, whereas overshoot of the osculatory spline only occurs within the neighboring points of the step. Akima's method is a straight line on either side of the step interval and only exhibits curvature within the step by creating curve sections that have much smaller radii of curvature than either the standard or osculatory splines.

Whether the wiggles of the standard spline or osculatory curves are as unnatural as Akima claims depends a great deal upon the nature of the portrayed data. In many situations the data may have a minimum curvature attribute and are more properly represented by standard splines. Indeed, manual smoothing performed with an a priori knowledge of the data will often introduce appropriate wiggles and curvature. But without any special knowledge of the data, manual drafting will probably more closely match the Akima method.

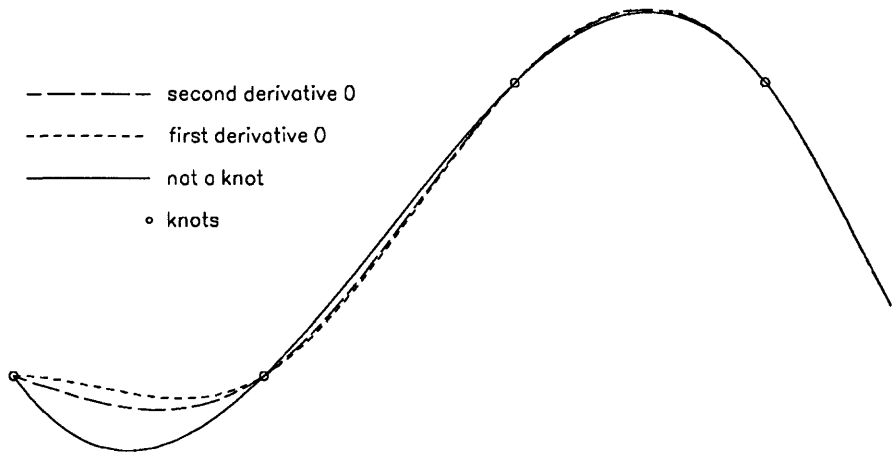

Figure 2. Effects of various end conditions at the beginning of a curve for a non-periodic standard spline.

The standard nonperiodic spline also requires that end point conditions must be specified. This is commonly performed by one of the following methods: specification of either the first or second derivative values or the "not-a-knot" condition. The latter method is recommended by de Boor (1978, p. 55–56) when no other information about the end conditions is available, but the zero second derivative method is employed in most of the illustrations in this report. Figure 2 shows the effect at the beginning of a curve segment for each of these methods. Zero second derivatives for both ends of the curve create what is termed the natural spline; the curve most closely approximates the mechanical spline with unconstrained ends.

Nonperiodic splines can be computed as periodic functions provided the first and last ordinates are equal and the endpoint derivatives are identical. Figure 3 compares the periodic splines for one cycle of a square wave defined by a 9 knot data set with the nonperiodic splines. The symmetry of the peak and trough of the periodic curve versus the asymmetric character of the nonperiodic curve are clearly what would be desired if the waveform represented a single cycle sampling of a continuous signal.

Since the Akima and osculatory methods determine coefficients from data value local to the knots, the periodic curves are achieved by simple replication of the data from either end of the data set to the alternate end. Standard splines, however, require a significantly different algorithm for periodic data because the entire data set is analyzed in coefficient determination.

Figure 4 shows an open and closed square box, demonstrating the three types of splines in parametric form for both open and closed curves. It is interesting to note that the closed standard spline for the square box closely approximates a circle. Again, the Akima method minimizes overshoot at the expense of smaller radii of curvature at the knots. In both cases the osculatory spline curve is virtually identical to, and consequently hidden by, the Akima method curve.

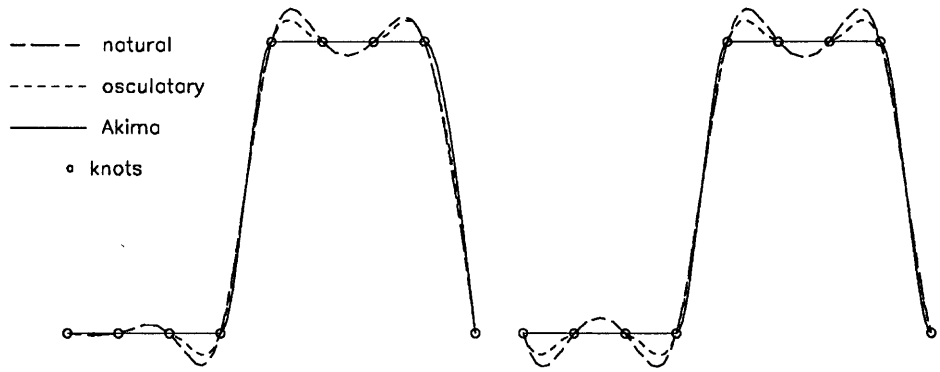

Figure 3. Nonparametric nonperiodic and periodic spline curves for a square wave function.

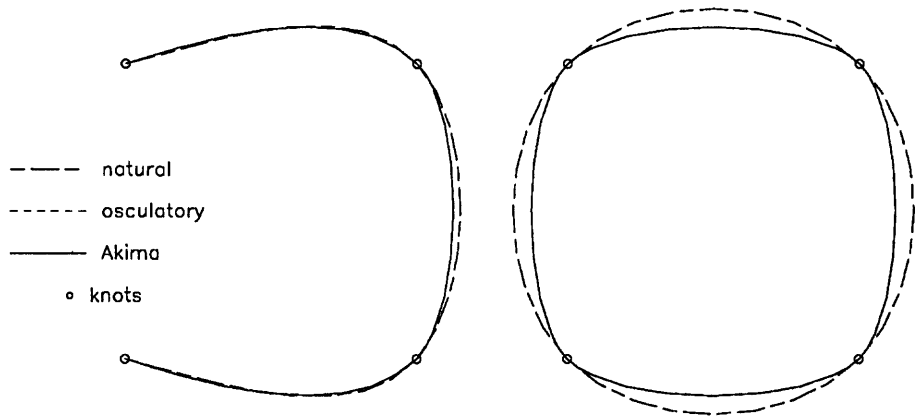

Figure 4. Parametric spline curves for an open and closed box.

The open and closed parametric splines employ the same algorithms as the respective nonparametric nonperiodic and periodic splines where the knot coordinate pairs are both considered ordinates and the abscissa for each axis is a summation of the internode hypotenuse. Units for each axis are also considered identical.

5. Determination of spline coefficients

Nonparametric splines consist of a set of $N-1$ cubic polynomial equations determined for a set of N knots defined by the coordinate pairs $x_1\ y_1, x_2\ y_2, \cdots, x_N\ y_N$ where $x_1<x_2<\cdots x_{N-1}<x_N$ and where the derivatives are continuous across the knots. The value of y for any x in the interval x_k, x_{k+1} can be determined by:

$$y(x) = \sum_{i=0}^{3} c_{ki}(x-x_k)^i$$

where $1 \le k < N-1$. The polynomial coefficients for each knot interval can be determined from the relations:

$$c_{k0}=y_k$$
$$c_{k1}=y'_k$$
$$c_{k2}=[3(m_k-y'_k)+y'_{k+1}-y'_k]/h_k$$
$$c_{k3}=[(y'_{k+1}-y'_k)/h_k-2(m_k-y'_k)]/h_k$$

where y'_k is the first derivative at each knot, $h_k=x_{k+1}-x_k$, and $m_k=(y_{k+1}-y_k)/h_k$.

Determination of the knot derivatives is the purpose of algorithms associated with the osculatory, Akima and standard splines. The following is a summary of the methods for determining the spline derivatives which are also presented as C language procedures in Appendix 1 along with tabulation of the results from a test data set.

5.1 Osculatory spline knot derivatives.

The osculatory derivatives are simply based upon the derivative of a quadratic passing through three adjacent knots, which can be expressed as:

$$y'_k \leftarrow (h_k m_{k-1} - h_{k-1} m_k)/(h_{k-1} + h_k) \text{ for } k \leftarrow 2, 3, \cdots, N-1,$$
$$y'_1 \leftarrow 2m_1 - y'_2 \text{ and}$$
$$y'_N \leftarrow 2m_{N-1} - y'_{N-1}$$

Note that in each of the methods only the interval size, h, and the divided differences, m, are required in the computations.

5.2 Akima spline knot derivatives.

The internal derivatives for Akima's method are obtained from:

$$y'_k \leftarrow \frac{|m_{k+1} - m_k|\, m_{k-1} + |m_{k-1} - m_{k-2}|\, m_k}{|m_{k+1} - m_k| + |m_{k-1} - m_{k-2}|} \text{ for } k \leftarrow 1, 2, \cdots, N.$$

For the indeterminate case represented by a zero denominator the derivative can be determined by:

$$y'_k \leftarrow (m_{k-1} + m_k)/2.$$

For the end conditions $k \leftarrow 1, 2, N-1$, and N, it is assumed that:

$$m_{-1} \leftarrow 3m_1 - 2m_2,$$
$$m_0 \leftarrow 2m_1 - m_2,$$
$$m_N \leftarrow 2m_{N-1} - m_{N-2}, \text{ and}$$
$$m_{N+1} \leftarrow 3m_{N-1} - 2m_{N-2}.$$

5.3 Standard spline knot derivatives.

The standard spline requires the solution of a set of simultaneous equations which fortunately involves a tridiagonal matrix with a computationally efficient algorithm for solution. In addition, the nonparametric form requires specification of either the first or second derivatives or the not-a-knot condition at the end knots in order to completely define the system of equations.

To determine the knot derivatives for a nonperiodic standard spline, one of the following initialization steps must be selected:

second derivative	first derivative	not-a-knot
$n \leftarrow 1$	$n \leftarrow 2$	$n \leftarrow 1$
$w_1 \leftarrow 1/2$	$w_1 \leftarrow 0$	$w_1 \leftarrow (h_1+h_2)/h_1$
$y'_1 \leftarrow (6m_1 - h_1 y''_1)/4$		$y'_1 \leftarrow \dfrac{[h_1+2(h_1+h_2)]h_2 m_1 + h_1^2 m_2}{(h_1+h_2)h_2}$

Then perform the following sequence:

$$\left.\begin{aligned} w_k &\leftarrow 1/[2 + (2 - w_{k-1})h_k/h_{k-1}] \\ y'_k &\leftarrow [3m_k + (3m_{k-1} - y'_{k-1})h_k/h_{k-1}]w_k \end{aligned}\right\} \text{ for } k \leftarrow 2, 3, \cdots, N-1$$

As with the initial knot specifications, the ending knot method must be selected. If the last knot's second derivative is known, then:

$$y'_N \leftarrow (3m_N + y''_{N-1}/h_{N-1} - y'_{N-1})/(2 - w_{N-1})$$

Otherwise, if a not-a-knot end condition is required then:

$$y'_N \leftarrow \frac{(h_{N-1}^2 m_{N-2} + (2r + h_{N-1})h_{N-2}m_{N-1})/3r - y'_{N-1}r}{h_{N-2} - w_{N-1}r}$$

where $r \leftarrow h_{N-1}/h_{N-2}$. No action is required if the last knot's first derivative is known. Finally, determine the remaining knot derivatives with:

$$y'_k \leftarrow y'_k - w_k y'_{k+1} \text{ for } k \leftarrow N-1, N-2, \cdots, n$$

End conditions of the nonperiodic spline are not required by the periodic spline but the method of solution is more complex and it is also important to remember that the ordinate value of the last knot is the same as the first. After setting $y'_1 \leftarrow w_1 \leftarrow 0$ and $v_1 \leftarrow 1$ the first pass is similar to the nonperiodic method:

$$\left.\begin{aligned} w_k &\leftarrow 1/[2 + (2 - w_{k-1})h_k/h_{k-1}] \\ v_k &\leftarrow -v_{k-1}w_k h_k/h_{k-1} \\ y'_k &\leftarrow [3m_k + (3m_{k-1} - y'_{k-1})h_k/h_{k-1}]w_k \end{aligned}\right\} \text{ for } k \leftarrow 2, 3, \cdots, N-1$$

After setting $v_{N-1} \leftarrow w_{N-1}$:

$$\left.\begin{aligned} v_k &\leftarrow v_k - w_k v_{k+1} \\ y'_k &\leftarrow y'_k - w_k y'_{k+1} \end{aligned}\right\} \text{ for } k \leftarrow N-1, N-2, \cdots, 2$$

The end knot derivatives are now determined from:

$$y'_1 \leftarrow y'_N \leftarrow \frac{(3m_{N-1} - y'_{N-1})h_1 + (3m_1 - y'_1)h_{N-1}}{(v_{N-1} + 2)h_1 + (v_1 + 2)h_{N-1}}$$

and the intermediate derivatives by:

$$y'_k \leftarrow y'_k + v_k y'_1 \text{ for } k \leftarrow 2, 3, \cdots, N-1.$$

6. Parametric splines.

The parametric spline is basically the same as the nonparametric form except that there are two functions of the parametric parameter t:

$$y(t) = \sum_{i=0}^{3} c_{ki} t^i \quad \text{and} \quad x(t) = \sum_{i=0}^{3} d_{ki} t^i$$

where $0 \le t \le [(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2]^{1/2}$. Determination of the polynomial coefficients is similar to that of the nonparametric form except that $y' = dy/dt$, $x' = dx/dt$, and h_k is the inter-node hypotenuse.

7. Bezier coordinates.

The Bezier system is often employed in CAD packages as a method to describe complex curves because the user can visualize the expected curve based upon discrete points that are interactively entered. Figure 5 shows several representative curves generated by Bezier interpolation based upon the points digitized. What is of interest here is the use of the Bezier system as a mechanism for describing curves such as splines without having to transmit information that is not in the same unit space as the knot coordinates or interpolate what might be a large number of intermediate values. For example, the spline knot coordinates and the first derivatives are impossible to handle by software without specialized coding and entry of other specialized descriptor information. If intermediate Bezier coordinates are introduced into the original data knot stream, then the entire set can be translated, scaled and (or) rotated in an identical manner and final conversion to a drafted curve can be deferred to the primitive device software or (in some cases) to the graphics hardware.

In the Bezier system, each axis (u) of a parametric curve segment is defined by the general polynomial expression (Newman and Sproull, 1979, p. 315):

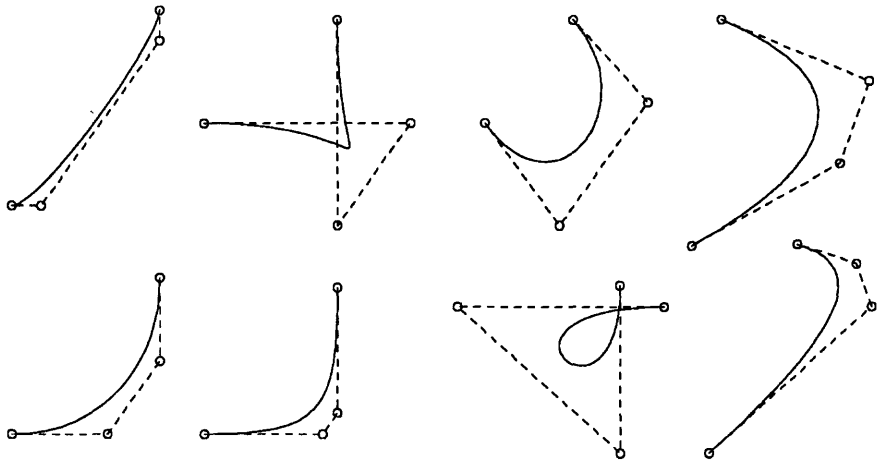

Figure 5. Representative sets of Bezier control points and their representative interpolated curves.

$$u(t) = \sum_{i=0}^{n} u_i \binom{n}{i} t^i (1-t)^{n-i}$$

where $0 \le t \le 1$. For two dimensional, third degree ($n=3$) parametric polynomials, the following coefficients can be obtained:

$$d_0 = X_0 \qquad c_0 = Y_0$$
$$d_1 = 3(X_1 - X_0) \qquad c_1 = 3(Y_1 - Y_0)$$
$$d_2 = 3(X_0 - 2X_1 + X_2) \qquad c_2 = 3(Y_0 - 2Y_1 + Y_2)$$
$$d_3 = X_3 - X_0 + 3(X_1 - X_2) \qquad c_3 = Y_3 - Y_0 + 3(Y_1 - Y_2)$$

Note that when $t=0$ the curve passes through X_0Y_0 and when $t=1$ it passes though X_3Y_3. The intermediate Bezier coordinates X_1Y_1 and X_2Y_2 act as control for the shape of the curve, which does not pass through these points except in a straight line situation.

Another property of interest in graphics windowing or clipping operations or in data retrieval is that the interpolated curve is always within the convex hull formed by the Bezier coordinates. Consequently, a Bezier set can be rejected if all of its points are outside a region of interest because there is no possibility that evaluation of intermediate values will produce curve segments inside the region.

Any spline segment arrived at by the previously discussed methods can be now expressed by adding two intermediate Bezier points to the data set using the following expressions:

	x–axis	y–axis
nonparametric	parametric	(both forms)
$X_{k1} = x_k + h_k/3$	$X_{k1} = x_k + h_k d_{k1}/3$	$Y_{k1} = y_k + h_k c_{k1}/3$
$X_{k2} = X_{k1} + h_k/3$	$X_{k2} = X_{k1} + (h_k d_{k1} + h_k^2 d_{k2})/3$	$Y_{k2} = Y_{k1} + (h_k c_{k1} + h_k^2 c_{k2})/3$

Regardless of how complex the shape of the original curves are, the Bezier method of defining this complexity only increases the size of the original set data from N to $N+2(N-1)$ points. Figure 6 shows a parametric standard spline curve where the the Bezier intermediate coordinates have been determined and employed in the graphics operation that generated the interpolated vectors.

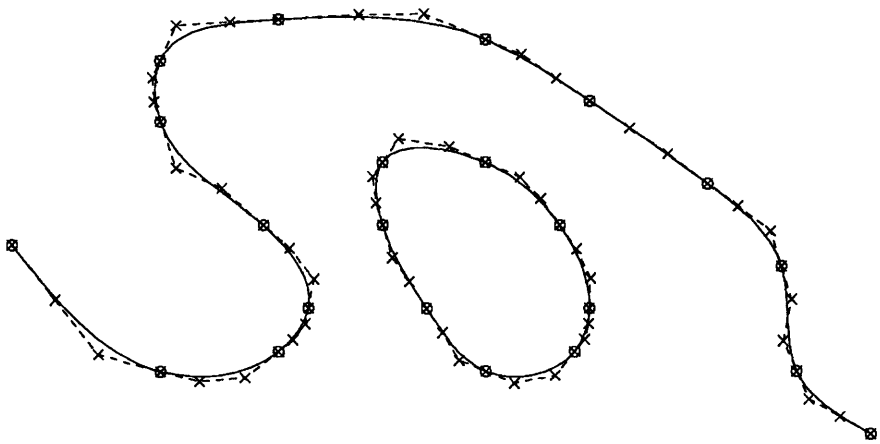

Figure 6. Bezier control points (crosses and dashed line) determined for a standard parametric spline curve determined for the circled knots.

8. C spline procedures

The FORTRAN programmer will have no difficulty locating source code or libraries for spline procedures discussed in this paper, but routines for the C programming language tend to be difficult to locate. A great deal of this problem is due to the relative newness of the C language and the somewhat slow adoption of the language by the scientific and engineering community. But there has recently been published a nonperiodic standard spline version (Press et al., 1988, p. 94–98) which is part of a significant collection of scientific C procedures.

One difficulty in developing general computer procedures for spline functions is that the potential usage is sufficiently varied so that any single method developed for one application will be less than optimal for another. To avoid the undue complexity that would be associated with code designed to cover a wide variety of possible applications, the following procedures are presented as reasonably optimal algorithms for basic determination of knot derivatives. In addition, the pre- and post-processing of the data is omitted as well as various checks on the veracity of the data because these are more properly a function of the application program rather than of the algorithms themselves.

8.1 Usage of spline procedures.

The synopsis of the basic spline functions is:

```
#include "spline.h"

void osc(C, n)
void aki(C, n)
void spl(C, n, der0, dern)
void pspl(C, n)

COEF *C;
int n, der0, dern;
```

where the functions `osc`, `aki`, `spl` and `pspl` are the respective osculatory, Akima, standard nonperiodic and standard periodic splines. `C` is a structure array defined in the header file `spline.h`:

```
typedef float REAL;
typedef struct { REAL h, m, c1; } COEF;
```

The `typedef` of `REAL` is a convenient means to change the procedures from single to double precision for nongraphic applications (especially those involving the standard splines) such as numerical approximation, integration or differentiation. The structure element `h` is the positive interknot interval size, `m` is the divided differences, and `c1` is the knot first derivative determined by the procedure. The number of knots, `n`, should be greater than 2 for all procedures.

In the case of the standard nonperiodic spline, `spl`, two flagging values, `der0` and `dern`, are used to indicate conditions to be applied to the respective first and last knot. A value of 1 or 2 indicates that the respective first or second derivative is specified, otherwise the not-a-knot condition is to be applied. When the derivative is specificed, the appropriate `c1`'s must be initialized by the user before execution. For example, for the natural spline where the second derivatives of the end points are zero, then `der0=dern=0` and `C[0].c1=C[n-1].c1=0` (granted, `c1` is the first derivative, but this is a practical expediency).

The standard spline procedures require a user-supplied routine `bomb` that is called with a string pointer argument in case temporary allocation of work memory fails. Do not return from this routine! `Spl` and `pspl` require `n REAL` and `2n REAL` words respectively of temporary work space.

Listing 1 in the Appendix is a test program that demonstrates the execution of the spline procedures as well as the means to verify their proper functioning by comparison of the printed results with Table 1. Appendix Listings 2 to 5 contain the code for the spline procedures.

Table 1. Knot derivatives for test data set.

x	y	Akima	Oscu-latory	Standard splines			Periodic
				Nonperiodic			
				note 1	note 2	note 3	
0	1	0.548611	0.553105	1.19225	0	0.684372	1.69459
0.8	1.5	0.655727	0.696895	0.358526	0.70386	0.506256	0.219145
1.7	2.2	0.799517	1.02603	1.57769	1.45131	1.5212	1.60492
3	4	–1.60689	–0.842658	–0.706826	–0.587738	–0.64403	–0.638622
4.1	1	–2.61642	–2.59569	–3.39218	–3.72494	–3.57624	–3.66705
4.9	–1	–2.28409	–0.681817	–1.21792	–0.15498	–0.627739	–0.317968
6	1	3.97727	4.31818	5.59144	1	3.04114	1.69459

note 1: not-a-knot end conditions
note 2: 0 and 1 beginning and ending first derivative conditions
note 3: zero second derivative end conditions

9. Conclusions.

Cubic spline functions are an important tool in scientific and engineering graphics applications, and application programs and procedures for their determination are widely available. However, there are many spline function variations and it is difficult to find any single source of material which provides more than one or two variants and presents the information in a truly practical manner.

The use of the cubic Bezier appears to be a viable mechanism for the description of any cubic function and suggests that it should be considered as a basic primitive in a comprehensive graphic systems. Use of Bezier coordinates should also be considered as a basic storage mechanism when mass storage is less important than CPU time required for regeneration of smoothed curves.

10. References.

Ahlberg, J.H., Nilson, E.N., Walsh, J.L, 1967, *The Theory of Splines and Their Aplications:* New York, Academic Press, 284 p.

Akima, Hiroshi, 1970, A new method of interpolation and smooth curve fitting based on local procedures: JACM, v. 17, no. 4, p. 589–602.

Akima, Hiroshi, 1972, Algorithm 433—Interpolation and smooth curve fitting based on local procedures: CACM, v. 15, no. 10, p. 914–918.

Cline, A.K., 1974, Scalar- and planar-valued curve fitting using splines under tension: CACM, v. 17, n. 4, p. 218–223.

de Boor, Carl, 1978, *A Practical Guide to Splines:* New York, Springer-Verlag, 392 p.

Evenden, G.I., 1988, Xspline.1—Unix documentation of enhanced spline filter utility: U.S. Geological Survey Administrative report, Woods Hole, Mass., 2 p.

Newman, W.M., Sproull, R.F., 1979, *Principles of Interactive Computer Graphics:* New York, McGraw-Hill, 541 p.

Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., 1988, *Numerical Recipes in C—the Art of Scientific Computing:* Cambridge, Cambridge University Press, 735 p.

11. Appendix

The procedures presented here are written with optimization of the process as a principle guideline and consequently, may lack in clarity for the less experienced C programmer. However, the best efforts of the author may still miss mechanisms which can enhance performance and improvements may be determined by the reader. The code is sufficiently structured so that alterations in execution, parameter testing, etc., can certainly be inserted. For example, the use of the user-supplied `bomb` procedure can be replaced with an appropriate error code return.

Listing 1. Example C test program executing spline procedures.

```
#include <stdio.h>
#include "spline.h"
#define N 7
     static
struct { REAL x, y; } data[N] = {
     0, 1, .8, 1.5, 1.7, 2.2, 3, 4,
     4.1, 1, 4.9, -1, 6, 1};
     void
bomb(s) char *s; {
     fprintf(stderr,"fatal error: %s\n",s);
     exit(1);
}
main() {
     int i, j;
     COEF C[N];
     void aki(), osc(), spl(), pspl();

     for (i = 0; i < N-1; ++i) {
         C[i].h = data[i+1].x - data[i].x;
         C[i].m = (data[i+1].y - data[i].y) / C[i].h;
     }
     for (j = 0; j < 6; ++j) {
         switch (j) {
         case 0: aki(C, N); break;
         case 1: osc(C, N); break;
         case 2: spl(C, N, 0, 0); break;
         case 3: C[0].c1 = 0; C[N-1].c1 = 1;
                 spl(C, N, 1, 1); break;
         case 4: C[0].c1 = C[N-1].c1 = 0;
                 spl(C, N, 2, 2); break;
         case 5: pspl(C, N); break;
         }
         for (i =  0; i < N; ++i)
             printf("%g\t%g\t%g\n",
                 data[i].x, data[i].y, C[i].c1);
     }
     exit(0);
}
```

Listing 2. C procedure to determine knot derivatives by Akima's method.

```
#include "spline.h"
     void
aki(C, n) COEF *C; {
     double fabs();
     REAL m1, m2, m3, m4, b, t1, t2;

     m3 = C->m;
     t1 = m3 - C[1].m;
     m2 = m3 + t1;
     m1 = m2 + t1;
     while ( n-- ) {
         m4 = n > 1 ? C[1].m : m3 + m3 - m2;
         t1 = fabs(m4 - m3);
         t2 = fabs(m2 - m1);
         (C++)->c1 = (b = t1 + t2) ? (t1 * m2 + t2 * m3) / b :
             .5 * (m2 + m3);
         m1 = m2;
         m2 = m3;
         m3 = m4;
     }
}
```

Listing 3. C procedure to determine knot derivatives by the osculatory method.

```
#include "spline.h"
     void
osc(C, n) COEF *C; {
     struct { REAL m, h; } L, N;
     COEF *D = C;

     L.m = C->m; L.h = C->h;
     C->c1 = L.m + L.m;
     --n;
     while (--n) {
         N.m = (++C)->m; N.h = C->h;
         C->c1 = (N.h * L.m + L.h * N.m) / (N.h + L.h);
         L = N;
     }
     D->c1 -= D[1].c1;
     C[1].c1 = L.m + L.m - C->c1;
}
```

Listing 4. C procedure to determine knot derivatives by the standard nonperiodic spline.

```
#include "spline.h"
     void
spl(C, n, der0, dern) COEF *C; {
     int is, i;
     char *malloc();
     void free();
     REAL hl, m3l, r, m3, c1, w, h, *W, *B;

     if (!(B = (REAL *)malloc(n * sizeof(REAL))))
         bomb("spl memory alloc. failure");
     W = B;
     m3 = C->m;
     h = C->h;
     switch (der0) { /* set starting conditions */
     case 2: c1 = C->c1 = 1.5 * C->m - .25 * C->h * C->c1;
         w = .5; is = 1;
         break;
     case 1: c1 = C->c1;
         w = 0.; is = 2;
         break;
     default: hl = C[1].h; r = hl + h;
         c1 = C->c1 = ((h + 2.*r)*hl*m3 + h*h*C[1].m) / (r * hl);
         w = r / hl; is = 1;
         break;
     }
     *W = w; m3 *= 3.;
     for ( i = 2; i < n; ++i) { /* decomp. and forward subst. */
         hl = h; m3l = m3;
         r = (h = (++C)->h) / hl;
         m3 = 3. * C->m;
         w = *++W = 1. / (2. + r * (2. - w));
         c1 = C->c1 = (m3 + r * (m3l - c1)) * w;
     }
     switch (dern) { /* set ending conditions */
     case 2: c1 = C[1].c1 = (m3 + .5 * C[1].c1 * h - c1)/(2. - w);
         break;
     case 1: c1 = C[1].c1;
         break;
     default: r = h + hl;
         c1 = C[1].c1 = ( (h*h*m3l + (2.*r + h)*hl*m3)/(3.*r)
             - r*c1) / (hl - w * r);
         break;
     }
     /* backsubstitution */
     while ( --n >= is ) c1 = (C--)->c1 -= *W-- * c1;
     free((char *)(B));
}
```

Listing 5. C procedure to determine knot derivatives by the standard periodic spline.

```
#include "spline.h"
     void
pspl(C, n) COEF *C; {
     REAL rhl, dylf, dyl, rh, r, dy, fnn, c1, w, v;
     struct DBL { REAL w, v; } *W, *WA;
     char *malloc();
     void free();
     int i;

     if (!(WA = W = (struct DBL *)
         malloc(n * sizeof(struct DBL))))
             bomb("nspl memory alloc. failure");
     c1 = w = C->c1 = W->w = 0.;
     v = W->v = 1.;
     rhl = 1. / C->h;
     dyl = dylf = 3. * C->m;
     n -= 2;
     rh = C[n].h * rhl;
     for (i = 1; i <= n; ++i) {
         r = (dy = (++C)->h) * rhl;
         rhl = 1. / dy;
         dy = 3. * C->m;
         (++W)->w = w = 1. / (2. + r * (2. - w));
         W->v = v = - r * v * w;
         C->c1 = c1 = (dy + r * (dyl - c1)) * w;
         dyl = dy;
     }
     W->v = (v -= w);
     for (i = n - 1; i > 0 ; --i) { /* reverse loop */
         w = (--W)->w;
         v = (W->v -= w * v);
         c1 = ((--C)->c1 -= w * c1);
     }
     --C; --W; /* compute endpoint derivative. */
     C->c1 = fnn = (dyl - C[n].c1 + (dylf - c1) * rh) /
         ((v + 2.) * rh + WA[n].v + 2.);
     for (i = 1; i <= n; ++i) /* second forward loop */
         (++C)->c1 += (++W)->v * fnn;
     (++C)->c1 = fnn;
     free(WA);
}
```