1989

Parametric splines in tension

Surendra K. Gupta

Parametric Splines in Tension

by

Surendra K. Gupta

Submitted in Partial Fulfillment
of the
Requirement for the Degree
of

MASTER OF SCIENCE

Approved by:

1. _____ Professor Guy Johnson

2. _____ Professor Peter G. Anderson

3. _____ Professor Peter H. Lutz

Department of Computer Science
Rochester Institute of Technology
Rochester, New York

July, 1989

Parametric Splines in Tension

Reproduction Agreement

I, **Surendra Kumar Gupta** prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

Dr. Surendra K. Gupta
61 Parkridge Drive
Pittsford, NY 14534

Dated: **July 7, 1989**

Abstract

A brief review of curve fitting terminology is presented, and the cubic spline interpolation scheme is outlined. Parametric and non-parametric curve fitting techniques are compared. The technique to fit parametric cubic splines is derived using the Euler-Lagrange formulation. Previous work on splines in tension is identified.

Employing the notion of splines in tension, a method is proposed to fit a parametric curve to a set of $(n + 1)$ points in d-dimension space satisfying a specified set of boundary conditions. The curve fitted will not have any inflection points within any span and will be invariant with respect to coordinate translation and rotation. Using Euler-Lagrange formulation, a system of linear equations in terms of the unknown second derivatives at knots is developed. Three kinds of boundary conditions are investigated. Software is developed in VAX Fortran to fit both parametric splines in tension and parametric cubic splines.

Applications where splines in tension may find use are identified. Some examples of such applications are presented and comparison to cubic spline made. Splines in tension offer a better alternative than Fourier transform in describing boundary of shape in digital image processing application. Possible extensions to the numerical scheme developed and related investigations by other workers in this field are also listed.

Acknowledgements

A number of individuals have facilitated the preparation of this monograph. My sincerest thanks are extended to:

- Guy Johnson, who provided superb guidance during research, painstakingly read the entire manuscript, and suggested several improvements;
- Peter Anderson, who always made himself available, help me set my methodology and goaded me to reach for the highest level of excellence;
- David J. Medvedeff, who helped me with the LaTeX system to typeset and print this monograph;
- Sandhya Varshney Gupta, who sustained me during all the stressful days, weeks, and months when this research was in progress.

Contents

List of Figures

Chapter 1

Introduction

In modeling of physical systems, there is often a need for drawing a smooth curve passing through a number of data points which typically are result of some scientific experiment or simulation. Such applications employ curve fitting techniques to interpolate between the data points and to compute both magnitude and direction of the tangent and curvature vector at each data point. For example, in the study of plastic deformation of crystalline materials, dislocation type *line* defects need to be modeled[19]. Since a dislocation assumes complex configurations as it glides through the crystal under applied stresses, smooth curves describing these configurations are required to study its glide. The curves can not be expressed analytically, and numerical curve fitting techniques become essential to simulate dislocation motion. To simulate such motion, knowledge of both tangent and curvature vectors at

the data points describing the dislocation shape is needed[18]. Influenced by author's experience in physical sciences, this investigation concentrates on curve fitting techniques in a non-interactive environment rather than interactive design of curves. In interactive design of curves, the control points can be moved to obtain a curve of desired shape. For example, B-splines[6, 10] are very useful and powerful in computer aided design. However, in many technical applications, for a given set of interpolation points, the shape of the fitted curve must be controlled by modifying shape parameters instead of altering some or all interpolation points. Consequently, in the past five years, interest in the applying splines to curve synthesis has resurged.

Consider fitting a smooth curve through $(n+1)$ data points $\{x_i, y_i\}_{i=0}^{n}$ in the x–y plane. Two distinct cases exist:

- Drawing a curve $y = f(x)$ when it is known that $f(x)$ underlying the data is single-valued, *i.e.*, for every x in the range $[x_0, x_n]$, $y(x)$ has a unique value. For example, when data represents measurement of temperature y_i at each time x_i, the drawn curve $y = f(x)$ should be **single-valued**.

- Drawing a curve through the data points which represent contour maps or shapes or trajectories. Now the data (x_i, y_i) is simply a collection of data points in the x–y plane, and there is no notion that y is a function of x. Thus, a given x in the range $[x_0, x_n]$ may correspond to several y values requiring a **multi-valued** curve. Such curves may be **open**, *i.e.*, point (x_0, y_0) does not coincide with point (x_n, y_n), or may be **closed**, *i.e.*, the two end-points (x_0, y_0) and (x_n, y_n) do coincide, for example, when data points describe a shape or a contour map. Some applications

may impose further restrictions, *e.g.*, contour maps require that several closed curves must not intersect one another.

In single-valued cases involving curves where y is implicitly dependent on x and the data points are closely related to the particular axis system in which they are defined, *e.g.*, in simple graph plotting applications, the need to rotate the coordinate system does not exist. In many other applications, however, there is no notion that y is a function of x and more importantly, axis rotation may be desired. In such cases, the single-valued curves are too restrictive, and not only the possibility of a curve being multi-valued must be allowed but also the curve must be invariant when the axes are rotated. For instance, the trajectory of a rocket described by a curve passing through a sequence of points in two-dimensional space may *appear* single-valued when the points are defined in a particular coordinate system x–y but will become multi-valued when the same points are redefined in another coordinate system x'–y'. Thus, the shape of the curve should depend only on the relative positions of the points and not on their positions relative to a particular axis system.

1.1 Single-valued Curve Fitting

Most common and popular technique for fitting a single-valued composite curve $y = f(x)$ through the data points $\{x_i, y_i\}_{i=0}^{n}$ is spline interpolation. The term spline is borrowed from the draftsman's aid—a thin metal, plastic or wooden strip which is elastically bent and constrained to pass through

the data points called **knots**. The physical spline representing the curve minimizes its elastic strain energy. Mathematically, it implies that the mean squared curvature is minimum, and in that sense, it is the smoothest curve passing through the knots. Each subinterval j: $[x_{j-1}, x_j]$ is called a **span** where $x_{j-1} < x_j$ and $j = 0, \ldots, n$. On each span j, a piecewise polynomial $S_j(x)$ of degree m, called a **spline**, is chosen such that it has its first $(m-1)$ derivatives continuous at the knots x_{j-1} and x_j. In practice, cubic splines $(m = 3)$ are used, and their second derivative continuity produces a visually smooth curve. The construction of cubic splines does not, however, assume that the derivatives of the interpolant agree with those of the function $f(x)$, even at the knots[2].

The particular problem of cubic spline interpolation is to find cubic polynomials $S_j(x)$ on span $[x_{j-1}, x_j]$ where $j = 1, \ldots, n$ such that

1. $S_1(x_0) = y_0$ and $S_j(x_j) = y_j$ for $j = 1, \ldots, n$; *i.e.*, composite curve passes through each data point;

2. $S_{j+1}(x_j) = S_j(x_j)$ for $j = 1, \ldots, n-1$; *i.e.*, successive splines meet at their common knot;

3. $S'_{j+1}(x_j) = S'_j(x_j)$ for $j = 1, \ldots, n-1$; *i.e.*, successive splines have first derivative continuity at their common knot;

4. $S''_{j+1}(x_j) = S''_j(x_j)$ for $j = 1, \ldots, n-1$; *i.e.*, successive splines have second derivative continuity at their common knot;

5. one of the following sets of boundary conditions is satisfied:

Type I: $S_1'(x_0) = y_0'$ and $S_n'(x_n) = y_n'$ where y_0' and y_n' are specified; *i.e.*, built-in boundary giving **clamped spline**, or

Type II: $S_1''(x_0) = S_n''(x_n) = 0$; *i.e.*, free boundary giving **natural spline**.

Cubic splines can be defined with other sets of boundary conditions.

The equations above describe a system of $4n$ linear algebraic equations involving the $4n$ unknown polynomial coefficients which can be solved for the value of each unknown. Readers requiring more details than the brief outline above can find good references on cubic spline interpolation in papers by Cox[11, 12] and on boundary (or end) conditions in the paper by Adams[1]. An alternate and more general formulation, employing the Euler-Lagrange equation and its integration, is presented in section 1.4. This formulation forms the basis for a variety of extensions to control the shape of fitted splines. Instead of describing each cubic spline by four polynomial coefficients (resulting in $4n$ unknowns), the curve is expressed in terms of the second derivative at each interpolation point. Thus, at most $(n+1)$ unknown second derivatives need to be found.

1.2 Multi-valued Curve Fitting

The cubic spline technique to fit single-valued curves extends quite naturally to multi-valued curves. Each data point (x_i, y_i) is characterized by a unique **parameter** t_i. Employing the method of cubic splines, single-valued inter-

polants $x = f(t)$ and $y = g(t)$ can be generated separately, and combined to give a parametric curve $(x(t), y(t))$. Even though each data point is assigned a unique parameter value t, a multi-valued curve can be fitted, in the sense that a value of x may correspond to several values of y and vice versa, *e.g.*, a curve which describes the shape of a two-dimensional physical object. But parametrization introduces a major new problem—how best to chose the value of t_i. Additionally, one may want parametric curves to be invariant under rotation and translation of coordinate system, *i.e.*, when the points are redefined with respect to another coordinate system on the same x–y plane. Otherwise, a solid object whose shape is being described by fitting the parametric curve will change shape as it is rotated.

The choice of parametrization is also crucial to the smoothness of the curve[7]. A simple parametrization, *e.g.*, $\{t_i = i\}_{i=0}^{n}$ can be used. Figure 1.1 shows a parametric cubic spline drawn through a set of data points in the form of a letter S with $t_i = i$. Ahlberg, Nilson and Walsh[2] suggest defining

$$\begin{aligned} t_0 &= 0 \\ t_i &= t_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \quad \text{for } i = 1, \ldots, n \end{aligned} \tag{1.1}$$

Thus, t becomes the accumulated chord length between successive data points. This parametrization when used on the data of figure 1.1 produces a curve shown in figure 1.2 which is visually smoother than the curve of figure 1.1. Choosing accumulated chord length as parameter t is useful even when the data points are not spaced equally. Theoretical arguments in its favor are presented by Epstein[13]. The resulting parametric curve is invariant under rotation and translation of the coordinate system.

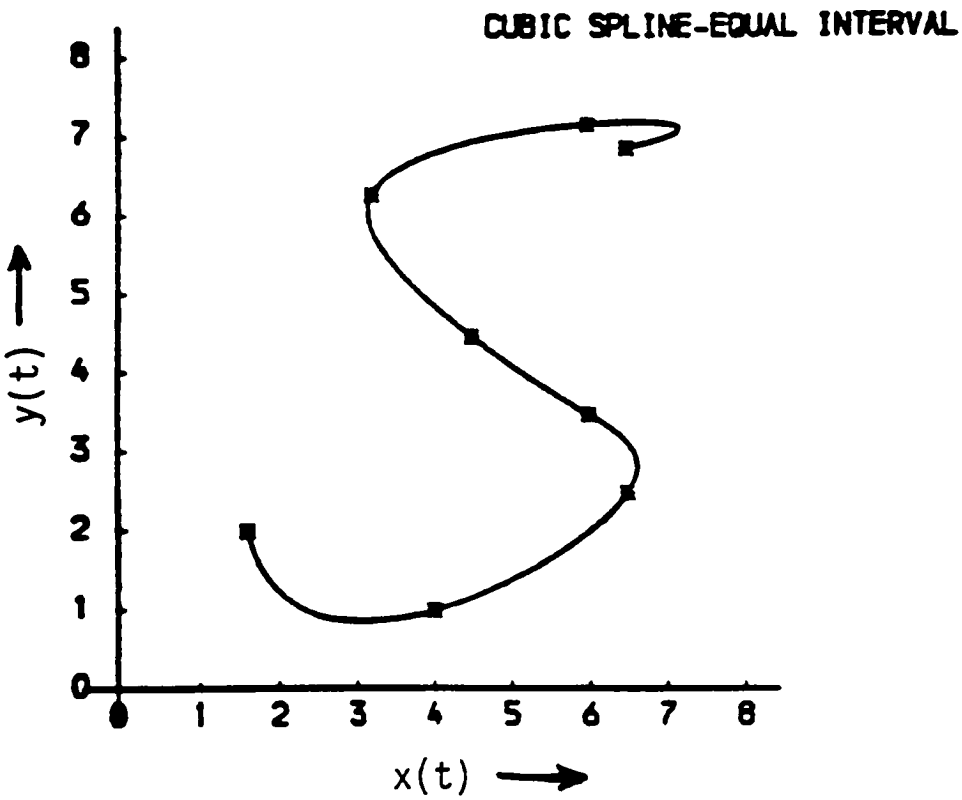

Figure 1.1: Parametric cubic spline[7] using $t_i = i$.

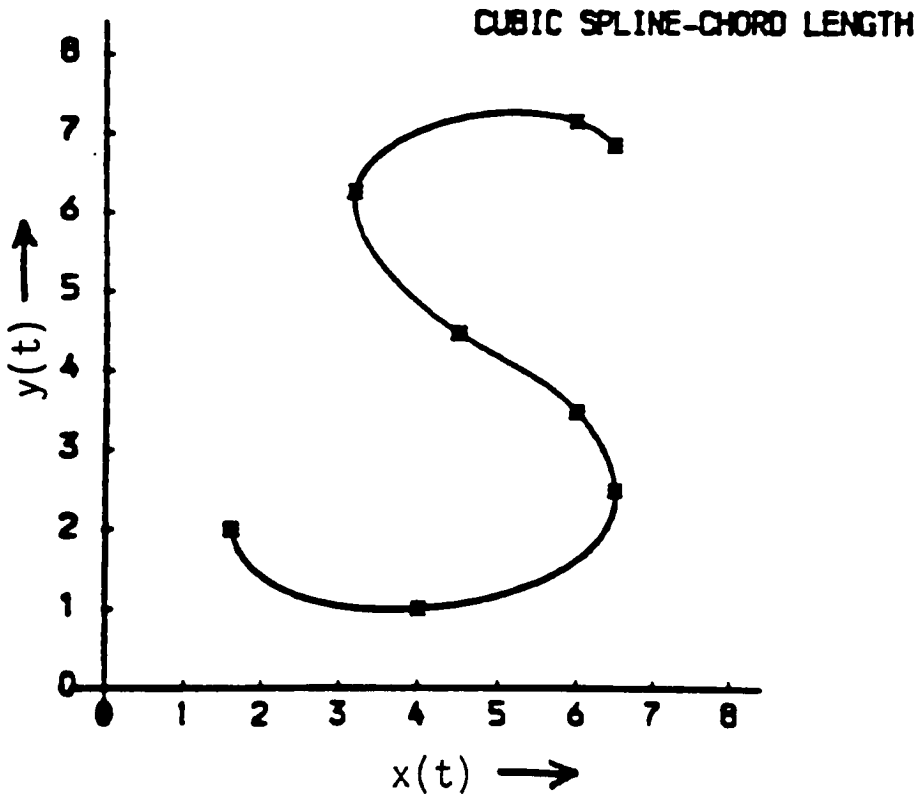

Figure 1.2: Parametric cubic spline[7] using accumulated chord length as the parameter t.

1.3 Parametric & Non-parametric Splines

Just as single-valued curves can not offer the flexibility required in some situations, parametric curves which are invariant under axis redefinition can not guarantee to produce single-valued curve from *single-valued data.* For instance, the two graphics packages popular in Great Britain: GHOST[15] and GINO-F[16], only offer the axis-invariant parametric curve drawing routines. These are disastrous for users who wish to ensure that single-valued curves are generated for single-valued data. Figure 1.3 shows the curve drawn by GHOST through a set of data points representing measurements of particle speed y at uniformly spaced instants of time x. Figure 1.4 shows the curve drawn by GINO-F through the same set of data points. Both curves show several speed values y at a given time x, *e.g.*, at $x \approx 5$, which does not make any physical sense because at any given time the particle must have a particular speed. *In parametric case, if parametrization $t = x$ is chosen, it degenerates to the single-valued case.*

1.4 Euler-Lagrange Formulation for Splines

The problem of interpolating a given set of points using cubic splines can be formulated in form of a second-order ordinary differential equation. Given a set of $(n+1)$ points $\{t_i, \vec{P}_i\}_{i=0}^{n}$ where t_i represents the parameter value and $\vec{P}_i$ the position vector of the point in a global coordinate system, the object is to find a set of $(n+1)$ vectors $\{\vec{S}''(t_i)\}_{i=0}^{n}$ representing the second derivative

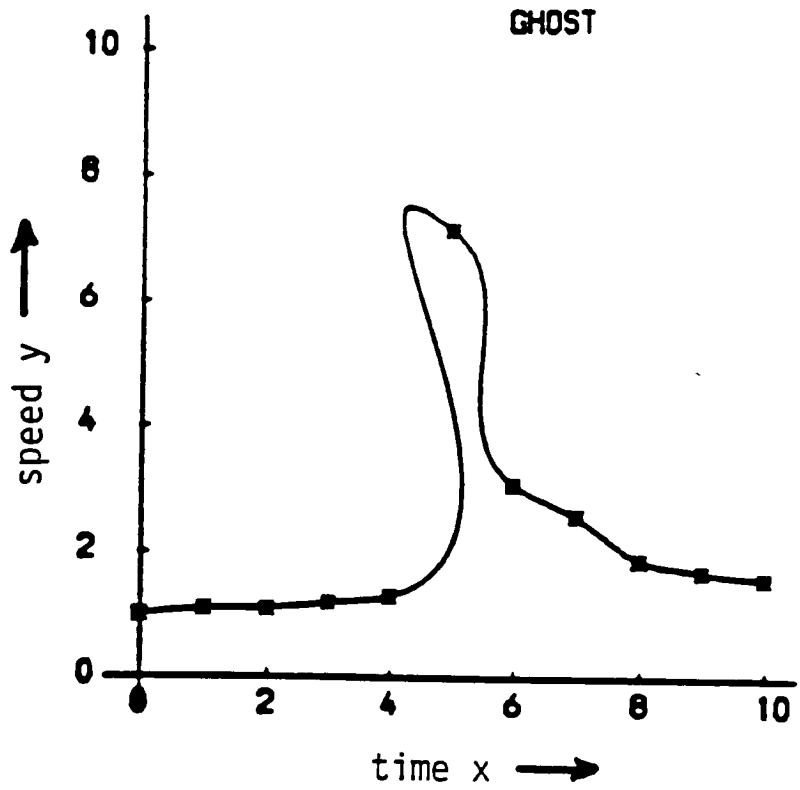

Figure 1.3: Parametric cubic spline[7] using GHOST software.

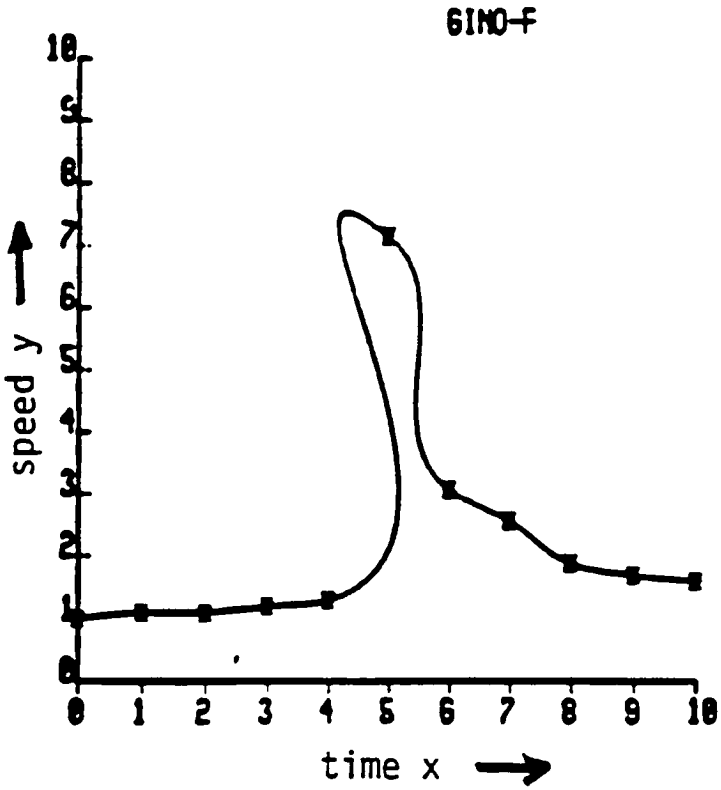

Figure 1.4: Parametric cubic spline[7] using GINO-F software.

of the curve at each point. The curve is described by a set of cubic splines $\{\vec{S}_j\}_{j=1}^{n}$. Each spline $\vec{S}_j$ is fitted over the span $[t_{j-1}, t_j]$. Assuming that $\vec{S}''(t)$ is linear on each span yields a method to construct the desired cubic splines, *i.e.*,

$$\begin{aligned} \vec{S}_j''(t) &= \vec{S}''(t_{j-1})\left[\frac{t_j - t}{h_j}\right] + \vec{S}''(t_j)\left[\frac{t - t_{j-1}}{h_j}\right] \\ &\quad \text{for } j = 1, \ldots, n \end{aligned} \tag{1.2}$$

where $h_j = t_j - t_{j-1}$ represents the parametric length of span j. Integrating the differential equation twice with respect to parameter t yields:

$$\vec{S}_j'(t) = \frac{\vec{S}''(t_{j-1})}{h_j}\left[t t_j - \frac{1}{2}t^2\right] + \frac{\vec{S}''(t_j)}{h_j}\left[\frac{1}{2}t^2 - t t_{j-1}\right] + \vec{\delta}_j \tag{1.3}$$

$$\begin{aligned} \vec{S}_j(t) &= \frac{\vec{S}''(t_{j-1})}{h_j}\left[\frac{1}{2}t^2 t_j - \frac{1}{6}t^3\right] + \frac{\vec{S}''(t_j)}{h_j}\left[\frac{1}{6}t^3 - \frac{1}{2}t^2 t_{j-1}\right] + \vec{\delta}_j t + \vec{\epsilon}_j \\ &\quad \text{for } j = 1, \ldots, n \end{aligned} \tag{1.4}$$

where $\vec{\delta}_j$ and $\vec{\epsilon}_j$ are the two constants of integration. The constants can be determined by invoking the two constraints: at $t = t_{j-1}$, $\vec{S}_j(t) = \vec{P}_{j-1}$ and at $t = t_j$, $\vec{S}_j(t) = \vec{P}_j$. Thus,

$$\begin{aligned} \vec{\delta}_j &= \left[\frac{\vec{P}_j - \vec{P}_{j-1}}{h_j}\right] \\ &\quad - \frac{\vec{S}''(t_{j-1})}{6h_j^2}\left[2t_j^3 - 3t_{j-1}^2 t_j + t_{j-1}^3\right] \\ &\quad - \frac{\vec{S}''(t_j)}{6h_j^2}\left[t_j^3 - 3t_{j-1}t_j^2 + 2t_{j-1}^3\right] \\ \vec{\epsilon}_j &= \left[\frac{t_j \vec{P}_{j-1} - t_{j-1}\vec{P}_j}{h_j}\right] \end{aligned} \tag{1.5}$$

$$+ \quad \frac{t_{j-1}t_j}{6h_j^2}\left[t_{j-1}^2 - 3t_{j-1}t_j + t_j^2\right]\left[\vec{S}''(t_{j-1}) + \vec{S}''(t_j)\right]$$
$$+ \quad \frac{t_{j-1}t_j}{6h_j^2}\left[t_j^2\vec{S}''(t_{j-1}) + t_{j-1}^2\vec{S}''(t_j)\right] \tag{1.6}$$

For the curve to have continuity in first-order derivative, two neighboring spans must have the same derivative vector at their common knot, *i.e.*,

$$\vec{S}_j'(t_j) = \vec{S}_{j+1}'(t_j) \qquad \text{for } j = 1, \ldots, n-1 \tag{1.7}$$

Thus, the continuity condition implies that

$$\begin{aligned} & \frac{t_j^2}{2h_j}\vec{S}''(t_{j-1}) + \frac{t_j^2 - 2t_{j-1}t_j}{2h_j}\vec{S}''(t_j) + \vec{\delta}_j \\ = \quad & \frac{2t_jt_{j+1} - t_j^2}{2h_j}\vec{S}''(t_j) - \frac{t_j^2}{2h_{j+1}}\vec{S}''(t_{j+1}) + \vec{\delta}_{j+1} \end{aligned}$$

The above equation is combined with the expression for constant $\vec{\delta}$ and re-arranged to provide

$$h_j\vec{S}''_{j-1} + 2(h_j + h_{j+1})\vec{S}''_j + h_{j+1}\vec{S}''_{j+1} = 6\left[\frac{\vec{P}_{j+1} - \vec{P}_j}{h_{j+1}} - \frac{\vec{P}_j - \vec{P}_{j-1}}{h_j}\right] \tag{1.8}$$
$$\text{for } j = 1, \ldots, n-1$$

For notational convenience, $\vec{S}''_i$ represents $\vec{S}''(t = t_i)$. Thus, $n-1$ equations are available in terms $n+1$ unknown second derivative vectors, *i.e.*, $\{\vec{S}''_i\}_{i=0}^{n}$. These equations can be combined with the two boundary conditions specified to yield the values for the unknown vectors. For example, if Type II boundary conditions are specified, *i.e.*,

$$\vec{P}_0'' = \vec{S}_0'' = 0 \qquad \text{and} \qquad \vec{P}_n'' = \vec{S}_n'' = 0$$

then the system of equations developed can be represented in a matrix form:

$$\mathbf{A}_{(n-1)\times(n-1)}\vec{\mathbf{S}}''_{(n-1)\times 1} = \vec{\mathbf{B}}_{(n-1)\times 1}$$

where the square matrix $\mathbf{A}$ contains known coefficients involving the parametric span lengths:

$$a_{i,i} = 2(h_i + h_{i+1}) \qquad \text{for } i = 1, \ldots, n-1$$

$$a_{i,i-1} = h_i \qquad \text{for } i = 2, \ldots, n-1$$

$$a_{i,i+1} = h_{i+1} \qquad \text{for } i = 1, \ldots, n-2$$

Matrix $\vec{\mathbf{B}}$ contains known residuals involving the parametric span lengths and the coordinate vectors of the knots:

$$b_i = 6\left[\frac{\vec{P}_{i+1} - \vec{P}_i}{h_{i+1}} - \frac{\vec{P}_i - \vec{P}_{i-1}}{h_i}\right] \qquad \text{for } i = 1, \ldots, n-1$$

Matrix $\vec{\mathbf{S}}''$ contains the unknown second derivative vectors:

$$s_i = \vec{S}_i'' \qquad \text{for } i = 1, \ldots, n-1$$

which need to be determined.

The coefficient matrix A is symmetric, tridiagonal and diagonally dominant which guarantees that the solution vector S'' will be unique and the solution technique will be stable with respect to round-off errors. Similar systems of equations can be developed for other types of boundary conditions.

1.5 Disadvantages of Cubic Splines

Cubic spline interpolation has two major disadvantages:

- It produces unwanted points of inflection on the curve. Figure 1.5 shows the cubic spline interpolant for a set of points describing cost y versus effectiveness x data. For theoretical reasons, the curve is known to be concave everywhere and free from inflection points. So, the spurious inflection point at $x \approx 10$ introduced by the interpolant is misleading.
- It produces undesirable oscillations in the curve. Figure 1.6 shows an example of cubic spline interpolant for a set of points describing x-ray intensity y versus crystal orientation x data. The overshoots at either side of the main peak are undesirable.

The unwanted inflection points and the undesirable oscillations can be removed by pulling on the ends of the spline curve. If the tension characterizing the pulling force at the two ends is very large, the data points will

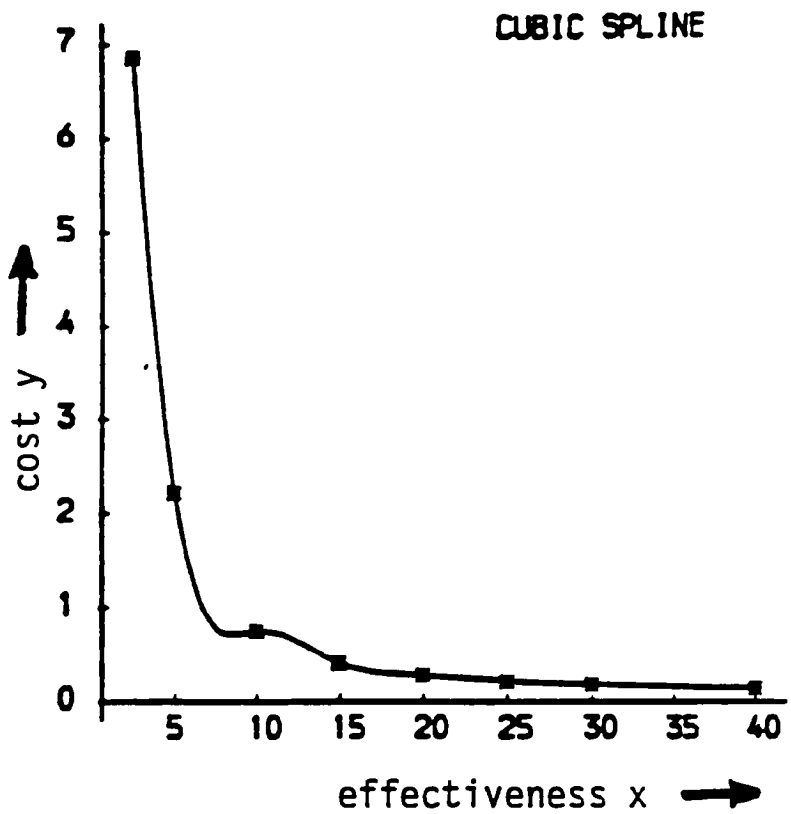

Figure 1.5: Cubic spline[7].

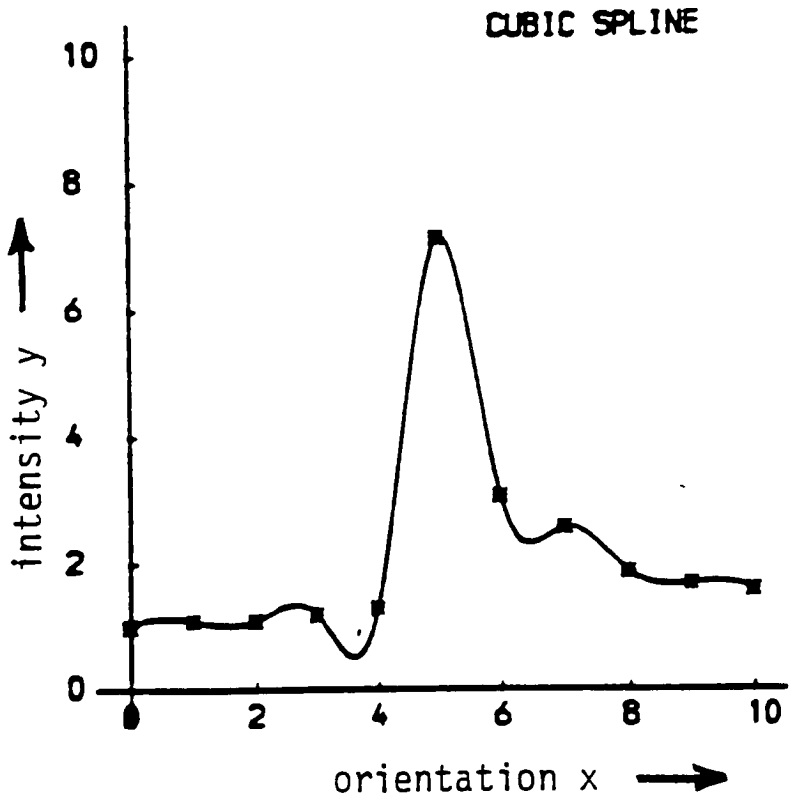

Figure 1.6: Cubic spline[7].

simply be connected by straight line segments, and the curve would be piecewise linear. Thus, spline in tension is the mathematical analogue of the mechanical spline subject to a uniform tension between its ends.

1.6 Splines in Tension

Schweikert[22] was the first person to investigate analytically the notion of splines in tension. Instead of the standard cubic interpolant: $f(x) = a + b \cdot x + c \cdot x^2 + d \cdot x^3$, he used the form: $f(x) = a + b \cdot x + c \cdot \sinh(\alpha \cdot x) + d \cdot \cosh(\alpha \cdot x)$ on each span where α is the tension parameter, and imposed first and second derivative continuity conditions at each knot. The limiting interpolant $\alpha \rightarrow 0$ becomes the standard cubic spline, whereas the other limiting interpolant $\alpha \rightarrow \infty$ degenerates the curve to become piecewise linear, *i.e.*, polygonal segments.

Cline[9] also investigated the splines in tension but with a different assumption, and his treatment yields results similar to those of Schweikert. He assumed that $\{f''(x) - \alpha^2 \cdot f(x)\}$ is linear on each span, again α is the tension parameter. This leads to a set of ordinary differential equations which can be integrated analytically.

Nielson[20] generalized the notion of splines in tension by characterizing them with a certain minimum property, and created a class of interpolants which he called ν splines. Splines in tension share many properties with ν splines.

The curves produced by splines in tension for the data of figures 1.5 and 1.6 are shown in figures 1.7 and 1.8. These curves are much tighter than those of figures 1.5 and 1.6. The unwanted inflection point and the undesirable oscillations no longer exist. In using splines in tension, however, the computational simplicity of piecewise cubic polynomials is lost since they now involve hyperbolic functions. Also, the tension parameter α needs to be varied until an acceptable curve is generated.

1.7 Problem Description

The treatments of both Schweikert and Nielson assume a $y = f(x)$ form thus restricting the curve fit to a set of knots in two-dimensional space (x–y plane) where the x-coordinates must monotonically increase in value. Cline also makes the same restriction, but suggests that his scheme can be generalized to fit a continuous curve through a sequence of points characterized by x-y coordinate pairs in the plane, which will allow approximating open, closed and multi-valued curves. This investigation extends Cline's scheme to parametric curves in d-dimension space.

Specifically, given $(n+1)$ point coordinate vectors in d-dimension $(d > 1)$ space $\{\vec{P}_i\}_{i=0}^{n}$ and two boundary conditions involving the first two derivatives $\vec{P}'(t)$ and(or) $\vec{P}''(t)$ at the two end-points $\vec{P}_0$ and $\vec{P}_n$ where parameter t is uniquely defined for each point vector, the object is to construct a set of piecewise parametric interpolatory splines $\{\vec{S}_j(t)\}_{j=1}^{n}$ defining a smooth curve which has first and second order derivative continuity at each interior knot

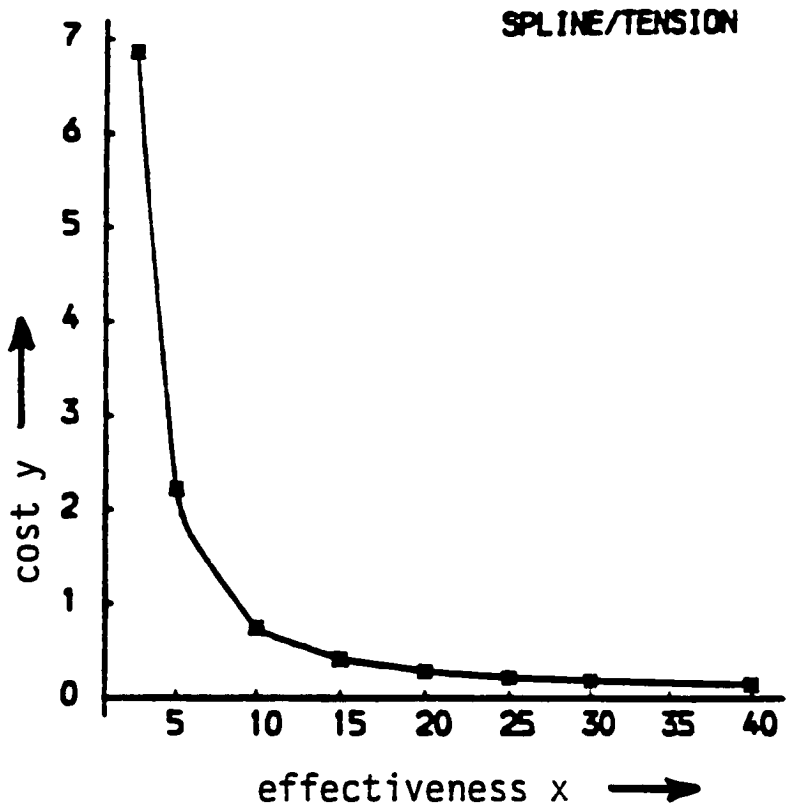

Figure 1.7: Spline in tension[7] using the data of figure 1.5.

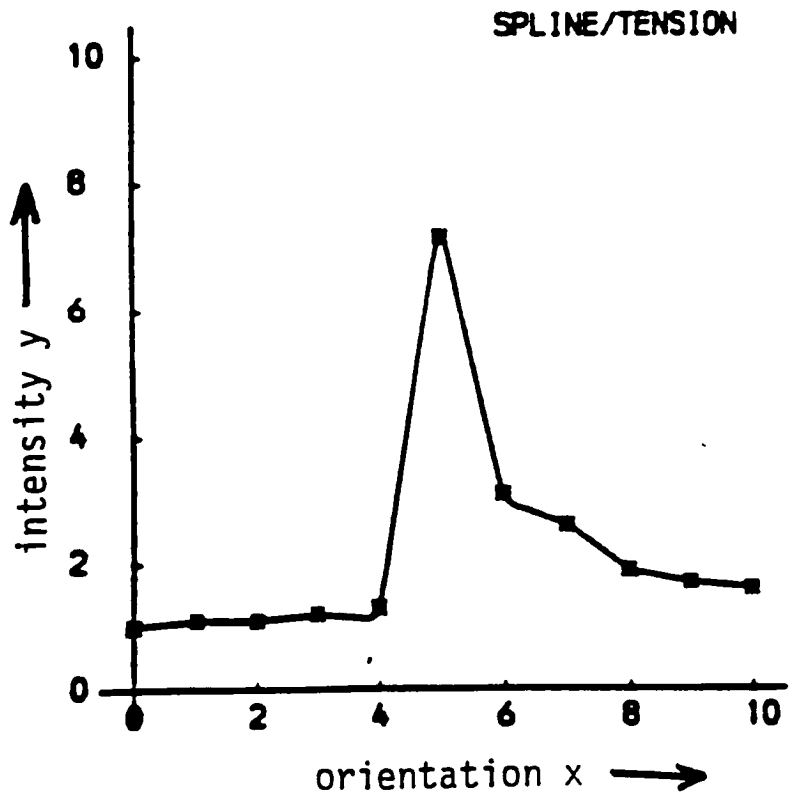

Figure 1.8: Spline in tension[7] using the data of figure 1.6.

$\{\vec{P}_i\}_{i=1}^{n-1}$. Thus, spline $\vec{S}_j(t)$ must have two continuous derivatives on its span $[t_{j-1}, t_j]$. Additionally, each spline $\vec{S}_j(t)$ must not have any inflection points[1] within the span $[t_{j-1}, t_j]$. The parameter t must be defined in a suitable manner to ensure that the drawn curve is invariant when the coordinate system is translated and rotated in the d-dimension space.

[1]inflection points are physically meaningful in two-dimensional space; in three-dimensional or higher space, their meaning becomes tenuous. For parametric curves, $\vec{P}''(t) = 0$ is assumed to represent an inflection point.

Chapter 2

Mathematical Development

Cline's assumption that $\{S''(t)-\alpha^2 \cdot S(t)\}$ is linear on each span where α is the tension parameter, can be incorporated in the Euler-Lagrange formulation of section 1.4. Such a formulation will yield a method to construct the splines in tension. In the following sections, the mathematics of the method to fit a parametric spline in tension is developed.

2.1 Constraints

The notion of accumulated chord length is extended to d-dimension space by using the Euclidean norm $\| \bullet \|_2$. Thus, parameter t_i characterizing point i

can be defined as

$$t_0 = 0$$

and

$$t_i = t_{i-1} + \|\vec{P}_i - \vec{P}_{i-1}\|_2 \quad \text{for } i = 1, \ldots, n \tag{2.1}$$

where $\|\vec{P}_i - \vec{P}_{i-1}\|_2$ represents the notion of the distance between point $(i-1)$ and point i.

The splines $\vec{S}_j(t)$ are chosen to satisfy the following constraints:

$$\vec{S}_1(t_0) = \vec{P}_0 \quad \text{and}$$

$$\vec{S}_j(t_j) = \vec{P}_j \quad \text{for } j = 1, \ldots, n \tag{2.2}$$

$$\vec{S}_{j+1}(t_j) = \vec{S}_j(t_j) \quad \text{for } j = 1, \ldots, n-1 \tag{2.3}$$

$$\vec{S}'_{j+1}(t_j) = \vec{S}'_j(t_j) \quad \text{for } j = 1, \ldots, n-1 \tag{2.4}$$

$$\vec{S}''_{j+1}(t_j) = \vec{S}''_j(t_j) \quad \text{for } j = 1, \ldots, n-1 \tag{2.5}$$

$$\begin{aligned} \vec{S}''_j(t) - \alpha^2 \cdot \vec{S}_j(t) &= \left[\vec{S}''(t_{j-1}) - \alpha^2 \cdot \vec{S}(t_{j-1})\right] \left[\frac{t_j - t}{t_j - t_{j-1}}\right] \\ &+ \left[\vec{S}''(t_j) - \alpha^2 \cdot \vec{S}(t_j)\right] \left[\frac{t - t_{j-1}}{t_j - t_{j-1}}\right] \\ &\text{for } j = 1, \ldots, n \end{aligned} \tag{2.6}$$

and a set of boundary conditions chosen from the next section.

2.2 Boundary Conditions

Many choices exist for specifying the end boundary conditions for a piecewise curve, but only the following three sets will be treated in this thesis:

Type I: Specify the two end tangent vectors, *i.e.*, the values of $\vec{S}_1'(t_0)$ and $\vec{S}_n'(t_n)$. This type of boundary condition results in **clamped splines**, also called **encastered splines**[21].

Type II: Set the second derivative at the two ends to zero, *i.e.*, $\vec{S}_1''(t_0) = \vec{S}_n''(t_n) = 0$. This type of boundary condition results in **natural splines**, also called **relaxed splines**[23].

Type III: Cyclic end conditions, *i.e.*, $\vec{S}_1'(t_0) = \vec{S}_n'(t_n)$ and $\vec{S}_1''(t_0) = \vec{S}_n''(t_n)$, can be used to produce a curve which repeats itself at intervals or a closed curve if $\vec{P}_0 = \vec{P}_n$. This type of boundary condition results in **cyclic splines**, sometimes also called **periodic splines**.

2.3 Integration of Differential Equation

To derive a governing equation similar to equation 1.8 in section 1.4, the linear differential equation 2.6 needs to be integrated twice, followed by substitution of constraint 2.2 and imposition of the continuity conditions. For

convenience, a modified equation 2.6 is reproduced below:

$$\begin{aligned}\vec{S}_j''(t) - \alpha^2 \cdot \vec{S}_j(t) &= \left[\vec{S}_{j-1}'' - \alpha^2 \cdot \vec{P}_{j-1}\right]\left[\frac{t_j - t}{h_j}\right] \\ &+ \left[\vec{S}_j'' - \alpha^2 \cdot \vec{P}_j\right]\left[\frac{t - t_{j-1}}{h_j}\right] \\ &\text{for } j = 1, \ldots, n\end{aligned}$$

where $h_j = t_j - t_{j-1}$ represents the notional parametric span length. Other modifications include substitution of $\vec{S}''(t_i)$ by $\vec{S}_i''$ and of $\vec{S}(t_i)$ by $\vec{P}_i$ for notational convenience and compactness. The homogeneous solution of the differential equation is

$$\vec{S}_{j,h}(t) = \vec{a}_j e^{\alpha \cdot t} + \vec{b}_j e^{-\alpha \cdot t} \tag{2.7}$$

where $\vec{a}_j$ and $\vec{b}_j$ are constant coefficients to be determined later, e is the exponential function and the subscript h denotes that it is the homogeneous solution. The particular solution is of the form

$$\vec{S}_{j,p}(t) = \vec{c}_j \cdot t + \vec{d}_j \tag{2.8}$$

where $\vec{c}_j$ and $\vec{d}_j$ are again constants to be determined later, and the subscript p denotes that it is the particular solution. Thus, the complete solution will be the sum of homogeneous and particular solution, *i.e.*,

$$\vec{S}_j(t) = \vec{a}_j e^{\alpha \cdot t} + \vec{b}_j e^{-\alpha \cdot t} + \vec{c}_j \cdot t + \vec{d}_j \tag{2.9}$$

This can be differentiated twice with respect to parameter t, and substituted in the differential equation above to obtain the constants $\vec{c}_j$ and $\vec{d}_j$. The particular solution becomes

$$\begin{aligned}\vec{S}_{j,p}(t) &= \left[-\alpha^{-2}\cdot\vec{S}''_{j-1}+\vec{P}_{j-1}\right]\left[\frac{t_j-t}{h_j}\right] \\ &+ \left[-\alpha^{-2}\cdot\vec{S}''_{j}+\vec{P}_{j}\right]\left[\frac{t-t_{j-1}}{h_j}\right]\end{aligned} \tag{2.10}$$

The constants $\vec{a}_j$ and $\vec{b}_j$ are determined by setting $\vec{S}_j(t)=\vec{P}_{j-1}$ at $t=t_{j-1}$ and $\vec{S}_j(t)=\vec{P}_j$ at $t=t_j$. Then,the complete solution becomes

$$\begin{aligned}\vec{S}_j(t) &= \frac{\vec{S}''_{j-1}}{\alpha^2}\left[\frac{\sinh\{\alpha(t_j-t)\}}{\sinh(\alpha h_j)}-\frac{(t_j-t)}{h_j}\right] \\ &+ \frac{\vec{S}''_{j}}{\alpha^2}\left[\frac{\sinh\{\alpha(t-t_{j-1})\}}{\sinh(\alpha h_j)}-\frac{(t-t_{j-1})}{h_j}\right] \\ &+ \vec{P}_{j-1}\frac{(t_j-t)}{h_j}+\vec{P}_j\frac{(t-t_{j-1})}{h_j}\end{aligned} \tag{2.11}$$

Thus, interpolation within a span j requires knowledge of the parametric span length h_j, the two end-points $\vec{P}_{j-1}$ and $\vec{P}_j$, second derivatives at the two end-points $\vec{S}''_{j-1}$ and $\vec{S}''_j$, and the tension parameter α.

The continuity conditions require that two neighboring spans must have the same first derivative at their common knot, *i.e.*,

$$\vec{S}'_j(t_j)=\vec{S}'_{j+1}(t_j) \qquad \text{for } j=1,\ldots,n-1$$

Hence, when the first derivative of the interpolant with proper indices is substituted in the continuity condition above, it yields

$$\begin{aligned}
& \frac{\vec{S}''_{j-1}}{\alpha^2}\left[\frac{1}{h_j} - \frac{\alpha}{\sinh(\alpha h_j)}\right] \\
+ \; & \frac{\vec{S}''_{j}}{\alpha^2}\left[\alpha\coth(\alpha h_j) - \frac{1}{h_j} + \alpha\coth(\alpha h_{j+1}) - \frac{1}{h_{j+1}}\right] \\
+ \; & \frac{\vec{S}''_{j+1}}{\alpha^2}\left[\frac{1}{h_{j+1}} - \frac{\alpha}{\sinh(\alpha h_{j+1})}\right] \\
= \; & \left[\frac{\vec{P}_{j+1} - \vec{P}_j}{h_{j+1}} - \frac{\vec{P}_j - \vec{P}_{j-1}}{h_j}\right] \qquad (2.12) \\
& \text{for } j = 1, \ldots, n-1
\end{aligned}$$

Obviously, this equation is very similar to that derived in section 1.4. But it now involves hyperbolic functions which are computationally time-intensive. Now, a system of $(n-1)$ equations can be developed involving $(n+1)$ unknown second derivative vectors, $\{\vec{S}''_i\}_{i=0}^{n}$. These equations when combined with the two boundary conditions will yield the solution matrix containing the unknown vectors.

2.4 Normalization of Tension Parameter α

Unfortunately, if the parameter t is scaled with tension parameter α remaining the same, a different curve will be generated. It is easily seen that equation 2.12 is altered in a non-linear manner if the parameter t (and hence span length h) is multiplied by a scalar. To remove this non-linear behav-

ior, the tension parameter α can be normalized by the average span length. Thus, the normalized tension parameter γ can be defined as

$$\gamma = \alpha \cdot \frac{t_n - t_0}{n} = \alpha \bar{h} \tag{2.13}$$

When appropriate substitutions of α by γ are made in equation 2.12, the same spline will result even if parameter t is linearly scaled, as long as the same normalized tension parameter γ is employed. For convenience, equations 2.11 and 2.12 are restated below using γ instead of α

$$\begin{aligned}
\vec{S}_j(t) &= \frac{\bar{h}^2 \vec{S}''_{j-1}}{\gamma^2} \left[\frac{\sinh\{\gamma(t_j - t)/\bar{h}\}}{\sinh(\gamma h_j/\bar{h})} - \frac{(t_j - t)}{h_j} \right] \\
&+ \frac{\bar{h}^2 \vec{S}''_j}{\gamma^2} \left[\frac{\sinh\{\gamma(t - t_{j-1})/\bar{h}\}}{\sinh(\gamma h_j/\bar{h})} - \frac{(t - t_{j-1})}{h_j} \right] \\
&+ \vec{P}_{j-1} \frac{(t_j - t)}{h_j} + \vec{P}_j \frac{(t - t_{j-1})}{h_j}
\end{aligned} \tag{2.14}$$

$$\begin{aligned}
&\frac{\bar{h}^2 \vec{S}''_{j-1}}{\gamma^2} \left[\frac{1}{h_j} - \frac{\gamma}{\bar{h} \sinh(\gamma h_j/\bar{h})} \right] \\
+ &\frac{\bar{h}^2 \vec{S}''_j}{\gamma^2} \left[\frac{\gamma}{\bar{h}} \coth(\gamma h_j/\bar{h}) - \frac{1}{h_j} + \frac{\gamma}{\bar{h}} \coth(\gamma h_{j+1}/\bar{h}) - \frac{1}{h_{j+1}} \right] \\
+ &\frac{\bar{h}^2 \vec{S}''_{j+1}}{\gamma^2} \left[\frac{1}{h_{j+1}} - \frac{\gamma}{\bar{h} \sinh(\gamma h_{j+1}/\bar{h})} \right] \\
= &\left[\frac{\vec{P}_{j+1} - \vec{P}_j}{h_{j+1}} - \frac{\vec{P}_j - \vec{P}_{j-1}}{h_j} \right] \\
&\text{for } j = 1, \ldots, n-1
\end{aligned} \tag{2.15}$$

2.5 Substitution of Boundary Conditions

Substituting any one the three types of boundary conditions being considered into the expression derived in the section above, will provide a non-homogeneous system of linear algebraic equations involving the unknowns $\vec{S}''(t_i)$. In each, the system of equations can be represented in a matrix form $\mathbf{A}\vec{\mathbf{S}}'' = \vec{\mathbf{B}}$, where the square matrix $\mathbf{A}$ contains known coefficients, matrix $\vec{\mathbf{B}}$ contains known residuals, and matrix $\vec{\mathbf{S}}''$ contains the unknown second derivatives which need to be determined. Both matrices $\vec{\mathbf{B}}$ and $\vec{\mathbf{S}}''$ will have d columns, each representing a coordinate direction of the d-dimension space.

2.5.1 Type I Boundary Conditions

To fit **clamped splines**, the values of $\vec{S}'(t_0)$ and $\vec{S}'(t_n)$ are specified by the user:

$$\vec{S}'(t_0) = \vec{\phi}_0$$

and

$$\vec{S}'(t_n) = \vec{\phi}_n$$

which allows $\vec{S}''_0$ to be expressed in terms of $\vec{S}''_1$ and $\vec{\phi}_0$, and $\vec{S}''_n$ in terms of $\vec{S}''_{n-1}$ and $\vec{\phi}_n$.

Equation 2.11 can be differentiated once with respect to t yielding:

$$\begin{aligned}\vec{S}'_j(t) &= \frac{\vec{S}''_{j-1}}{\alpha^2}\left[-\frac{\alpha\cosh\{\alpha(t_j-t)\}}{\sinh(\alpha h_j)}+\frac{1}{h_j}\right] \\ &+ \frac{\vec{S}''_j}{\alpha^2}\left[\frac{\alpha\cosh\{\alpha(t-t_{j-1})\}}{\sinh(\alpha h_j)}-\frac{1}{h_j}\right] \\ &+ \frac{(\vec{P}_j-\vec{P}_{j-1})}{h_j} \qquad (2.16)\end{aligned}$$

Substituting $\vec{S}'_1(t_0)=\vec{\phi}_0$ in the equation above provides

$$\begin{aligned}\frac{\vec{S}''_0}{\alpha^2} &= \frac{\vec{S}''_1}{\alpha^2}\left[\frac{\alpha h_1-\sinh(\alpha h_1)}{\alpha h_1\cosh(\alpha h_1)-\sinh(\alpha h_1)}\right] \\ &+ \sinh(\alpha h_1)\left[\frac{(\vec{P}_1-\vec{P}_0)-h_1\vec{\phi}_0}{\alpha h_1\cosh(\alpha h_1)-\sinh(\alpha h_1)}\right] \qquad (2.17)\end{aligned}$$

and substituting $\vec{S}'_n(t_n)=\vec{\phi}_n$ provides

$$\begin{aligned}\frac{\vec{S}''_n}{\alpha^2} &= \frac{\vec{S}''_{n-1}}{\alpha^2}\left[\frac{\alpha h_n-\sinh(\alpha h_n)}{\alpha h_n\cosh(\alpha h_n)-\sinh(\alpha h_n)}\right] \\ &+ \sinh(\alpha h_n)\left[\frac{h_n\vec{\phi}_n-(\vec{P}_n-\vec{P}_{n-1})}{\alpha h_n\cosh(\alpha h_n)-\sinh(\alpha h_n)}\right] \qquad (2.18)\end{aligned}$$

The two equations above in conjunction with equation 2.12 yield a system of $(n-1)$ equations in terms of $(n-1)$ unknown second derivatives, *i.e.*, $\{\vec{S}''_i\}_{i=1}^{n-1}$. Such a system can be represented in a matrix form: $\mathbf{A}\vec{\mathbf{S}}''=\vec{\mathbf{B}}$ where

$$a_{i,i}=\alpha\coth(\alpha h_i)-\frac{1}{h_i}+\alpha\coth(\alpha h_{i+1})-\frac{1}{h_{i+1}} \quad \text{for } i=2,\ldots,n-2$$

$$a_{i,i-1} = \frac{1}{h_i} - \frac{\alpha}{\sinh(\alpha h_i)} \qquad \text{for } i = 2, \ldots, n-1$$

$$a_{i,i+1} = \frac{1}{h_{i+1}} - \frac{\alpha}{\sinh(\alpha h_{i+1})} \qquad \text{for } i = 1, \ldots, n-2$$

$$\vec{s}_i = \frac{\vec{S}_i''}{\alpha^2} \qquad \text{for } i = 1, \ldots, n-1$$

$$\vec{b}_i = \left[\frac{\vec{P}_{i+1} - \vec{P}_i}{h_{i+1}} - \frac{\vec{P}_i - \vec{P}_{i-1}}{h_i} \right] \qquad \text{for } i = 2, \ldots, n-2$$

and

$$\begin{aligned} a_{1,1} &= \alpha \coth(\alpha h_1) - \frac{1}{h_1} + \alpha \coth(\alpha h_2) - \frac{1}{h_2} \\ &+ \frac{\sinh(\alpha h_1) - \alpha h_1}{\sinh(\alpha h_1) - \alpha h_1 \cosh(\alpha h_1)} \left[\frac{1}{h_1} - \frac{\alpha}{\sinh(\alpha h_1)} \right] \end{aligned}$$

$$\begin{aligned} \vec{b}_1 &= \frac{\vec{P}_2 - \vec{P}_1}{h_2} \\ &- \frac{\vec{P}_1 - \vec{P}_0}{h_1} \left[\frac{\alpha h_1 - \alpha h_1 \cosh(\alpha h_1)}{\sinh(\alpha h_1) - \alpha h_1 \cosh(\alpha h_1)} \right] \\ &- \vec{\phi}_0 \left[\frac{\sinh(\alpha h_1) - \alpha h_1}{\sinh(\alpha h_1) - \alpha h_1 \cosh(\alpha h_1)} \right] \end{aligned}$$

$$\begin{aligned} a_{n-1,n-1} &= \alpha \coth(\alpha h_{n-1}) - \frac{1}{h_{n-1}} + \alpha \coth(\alpha h_n) - \frac{1}{h_n} \\ &+ \left[\frac{\sinh(\alpha h_n) - \alpha h_n}{\sinh(\alpha h_n) - \alpha h_n \cosh(\alpha h_n)} \right) \left(\frac{1}{h_n} - \frac{\alpha}{\sinh(\alpha h_n)} \right] \end{aligned}$$

$$\vec{b}_{n-1} = \frac{\vec{P}_n - \vec{P}_{n-1}}{h_n}\left[\frac{\alpha h_n - \alpha h_n \cosh(\alpha h_n)}{\sinh(\alpha h_n) - \alpha h_n \cosh(\alpha h_n)}\right]$$
$$- \frac{\vec{P}_{n-1} - \vec{P}_{n-2}}{h_{n-1}} + \vec{\phi}_n\left[\frac{\sinh(\alpha h_n) - \alpha h_n}{\sinh(\alpha h_n) - \alpha h_n \cosh(\alpha h_n)}\right]$$

Matrix **A** is symmetric, tridiagonal and diagonally dominant. Thus, a unique and non-trivial solution to the system of equations mentioned above exists. Furthermore, numerical techniques employed to solve such a system will be stable with respect to round-off errors, and will not require any special pivoting or scaling strategies.

2.5.2 Type II Boundary Conditions

Natural splines require the second derivative $\vec{S}''(t)$ to be zero at $t = t_0$ and at $t = t_n$. Thus,

$$\vec{S}_0'' = 0 \text{ and } \vec{S}_n'' = 0$$

The equations above in conjunction with equation 2.12 again yield a system of $(n-1)$ equations in terms of $(n-1)$ unknown second derivatives, *i.e.*, $\{\vec{S}_i''\}_{i=1}^{n-1}$. Such a system can be represented in matrix form: $\mathbf{A}\vec{\mathbf{S}}'' = \vec{\mathbf{B}}$ where the only non-zero entries are

$$a_{i,i} = \alpha \coth(\alpha h_i) - \frac{1}{h_i} + \alpha \coth(\alpha h_{i+1}) - \frac{1}{h_{i+1}} \quad \text{for } i = 1, \ldots, n-1$$

$$a_{i,i-1} = \frac{1}{h_i} - \frac{\alpha}{\sinh(\alpha h_i)} \qquad \text{for } i = 2, \ldots, n-1$$

$$a_{i,i+1} = \frac{1}{h_{i+1}} - \frac{\alpha}{\sinh(\alpha h_{i+1})} \qquad \text{for } i = 1, \ldots, n-2$$

$$\vec{s}_i = \frac{\vec{S}_i''}{\alpha^2} \qquad \text{for } i = 1, \ldots, n-1$$

$$\vec{b}_i = \left[\frac{\vec{P}_{i+1} - \vec{P}_i}{h_{i+1}} - \frac{\vec{P}_i - \vec{P}_{i-1}}{h_i} \right] \qquad \text{for } i = 1, \ldots, n-1$$

Again, matrix **A** is symmetric, tridiagonal and diagonally dominant. Thus, a unique and non-trivial solution to the system of equations mentioned above exists. Furthermore, numerical techniques employed to solve such a system will be stable with respect to round-off errors, and will not require any special pivoting or scaling strategies.

2.5.3 Type III Boundary Conditions

Cyclic splines require that the first derivative $\vec{S}'$ be same at $t = t_0$ and $t = t_n$, and the second derivative $\vec{S}''$ at $t = t_0$ be equal to that at $t = t_n$. Thus,

$$\vec{S}_0' = \vec{S}_n' \qquad \text{and} \qquad \vec{S}_0'' = \vec{S}_n''$$

Equation 2.16 can be algebraically manipulated to yield the needed equation

$$\begin{aligned}
& \frac{\vec{S}_1''}{\alpha^2}\left[\frac{\alpha}{\sinh(\alpha h_1)} - \frac{1}{h_1}\right] \\
+ \quad & \frac{\vec{S}_{n-1}''}{\alpha^2}\left[\frac{\alpha}{\sinh(\alpha h_n)} - \frac{1}{h_n}\right] \\
+ \quad & \frac{\vec{S}_n''}{\alpha^2}\left[-\alpha\coth(\alpha h_1) + \frac{1}{h_1} - \alpha\coth(\alpha h_n) + \frac{1}{h_n}\right] \\
= \quad & \frac{\vec{P}_n - \vec{P}_{n-1}}{h_n} - \frac{\vec{P}_1 - \vec{P}_0}{h_1}
\end{aligned} \tag{2.19}$$

Now, the equations above in conjunction with equation 2.12 yield a system of n equations in terms of n unknown second derivatives, *i.e.*, $\{\vec{S}_i''\}_{i=1}^n$. The system of equations can still be represented in a matrix form: $\mathbf{A}\vec{\mathbf{S}}'' = \vec{\mathbf{B}}$ where the non-zero entries are

$$a_{i,i} = \alpha\coth(\alpha h_i) - \frac{1}{h_i} + \alpha\coth(\alpha h_{i+1}) - \frac{1}{h_{i+1}} \quad \text{for } i = 1, \ldots, n-1$$

$$a_{i,i-1} = \frac{1}{h_i} - \frac{\alpha}{\sinh(\alpha h_i)} \qquad \text{for } i = 2, \ldots, n$$

$$a_{i,i+1} = \frac{1}{h_{i+1}} - \frac{\alpha}{\sinh(\alpha h_{i+1})} \qquad \text{for } i = 1, \ldots, n-1$$

$$\vec{s}_i = \frac{\vec{S}_i''}{\alpha^2} \qquad \text{for } i = 1, \ldots, n$$

$$\vec{b}_i = \left[\frac{\vec{P}_{i+1} - \vec{P}_i}{h_{i+1}} - \frac{\vec{P}_i - \vec{P}_{i-1}}{h_i}\right] \qquad \text{for } i = 1, \ldots, n-1$$

and

$$a_{1,n} = \frac{1}{h_1} - \frac{\alpha}{\sinh(\alpha h_1)}$$

$$a_{n,1} = \frac{1}{h_1} - \frac{\alpha}{\sinh(\alpha h_1)}$$

$$a_{n,n} = \alpha \coth(\alpha h_1) - \frac{1}{h_1} + \alpha \coth(\alpha h_n) - \frac{1}{h_n}$$

$$\vec{b}_n = \frac{\vec{P}_1 - \vec{P}_0}{h_1} - \frac{\vec{P}_n - \vec{P}_{n-1}}{h_n}$$

The matrix **A** is still symmetric, banded and diagonally dominant although not tridiagonal. Thus, numerical techniques employed to solve the system will be stable with respect to round-off errors and will yield the unique and non-trivial solution. However, due to the corner elements $a_{1,n}$ and $a_{n,1}$, the numerical techniques will no longer be as computationally efficient as those for tridiagonal matrices.

2.6 Solution of System of Linear Equations

Many methods exist to solve such systems of linear equations, for example, elimination methods, LU decomposition methods and iterative methods[8].

For a system of equations of degree n, when the coefficient matrix is diagonally dominant and tridiagonal, a modified Gaussian elimination technique can be used. Pivoting or scaling strategies are not required, and the technique is $O(nd)$. If the coefficient matrix is diagonally dominant but not tridiagonal, the same technique can be employed but is no longer $O(nd)$. In this study, the technique is implemented as a separate routine without any special provisions for storing efficiently the elements of the coefficient matrix.

2.7 Implementation

The scheme to fit parametric splines in tension as outlined in this chapter can be implemented simply using few routines. Detailed algorithms are not presented in this section since each routine is simple being merely a higher-language translation of the equations derived in this chapter. Compiler-produced listings of the routines can be found in appendix A. The main program **TENSION** requires standard input and produces standard output. The input data is

n: the number of data points;

d: the number of components in each data point coordinate vector;

P: coordinate vector for each data point;

γ: the normalized tension parameter;

ptype: the type of parametrization desired,

- type **A** implies that the parameter t equals the first component of coordinate vector **P** of each data point, *i.e.*, $t = x$ for single-valued curve fitting;
- type **B** implies that the parameter t equals the index of each data point, *i.e.*, $t = i$. This parametrization is not satisfactory and is implemented to allow comparison with published data; and
- type **C** implies that the parameter t equals the accumulated chord length between successive data points (see equation 2.1). This parametrization is most appropriate for many-valued curve fitting.

btype: the type of boundary conditions desired,

- type 1 implies **clamped splines**, in which case two vectors, the first derivative at each end, must be supplied by the user;
- type 2 implies **natural splines**, no additional information is needed; and
- type 3 implies **cyclic splines**, again, no additional information is needed.

The standard output is the list of second derivative vectors **SPP** $= \vec{S}''$.

The main program first invokes a subprogram called **PARAMETRIZE** to assign a parameter value to each data point depending on **ptype** specified by the user. It then invokes another subprogram called **TSPLINES** to fit parametric splines in tension with the specified type of boundary conditions. The subprogram **TSPLINES**, in turn, invokes another subprogram called **GAUSS** to solve a system of equations by modified Gaussian elimination method.

A similar program package to fit parametric cubic splines can be found in appendix B.

Chapter 3

Some Applications & Future Extensions

The tension parameter α can be considered as a shape factor under user-control which the user can vary to alter the shape of a curve fitted to a given set of data points. This is in contrast to computer-aided design of curves where for example, using B-splines, a user can modify the curve by increasing or decreasing the number of control points, even perhaps altering a few control points. In most scientific applications, however, a user does not have the flexibility to modify the given set of data points, and can only alter the fitted curve by manipulating shape factors. In this chapter, a few applications are presented which demonstrate the use of splines in tension. Since splines in tension are developed as an alternative to cubic splines, in

the first two examples, both types of curves are presented for comparison.

3.1 Tension versus Cubic Splines

To illustrate the dramatic effect of tension applied to a closed curve, a simple shape composed of a semicircle over a rectangle is constructed. To represent the shape, knots are chosen at every 15° interval on the semicircle of unit radius, at the corners and the midpoint of each side of the rectangle. Thus, a total of 18 points are chosen with the 0^{th} point coinciding with the 18^{th} point. Coordinates of each point are listed below.

i	0	1	2	3	4
x_i	0.0	0.03407417	0.133975	0.2928932	0.5
y_i	0.0	0.258819	0.5	0.7071068	0.866025
i	5	6	7	8	9
x_i	0.741181	1.0	1.258819	1.5	1.707107
y_i	0.9659258	1.0	0.9659258	0.866025	0.7071068
i	10	11	12	13	14
x_i	1.866025	1.965926	2.0	2.0	2.0
y_i	0.5	0.258819	0.0	-0.5	-1.0
i	15	16	17	18	
x_i	1.0	0.0	0.0	0.0	
y_i	-1.0	-1.0	-0.5	0.0	

Accumulated chord length is chosen for curve parameter and cyclic boundary conditions (*i.e.*, first and second derivatives are matched at the first and the last point) are imposed. The curve that results when a cubic spline is fitted to the data points is shown in figure 3.1. The semicircular part of the shape is reproduced accurately by the cubic spline. This is to be expected because the semicircle has a constant curvature over its span and cubic spline interpolates linearly on each span the curvature at the knots. Thus, the cubic spline fit results in a circular arc. However, cubic spline poorly represents the rectangular part of the shape. It has extraneous curls at the corners of the rectangle.

A tension spline with normalized tension parameter $\gamma = 0.001$ with cyclic boundary conditions is fitted to the same set of data points and is shown in figure 3.2. When compared to cubic spline, the semicircular portion of the shape is not fitted well by tension spline. However, the fit to rectangular portion is excellent. It should be noted that the curve tightens considerably with such a small tension (*i.e.*, $\gamma = 0.001$).

3.2 Shape with Fillets

To eliminate stress concentrations that arise at sharp corners, engineering components are often rounded at corners (this rounding is referred as fillets). The shape in the last section is rounded at the bottom two rectangular corners with a circular arc of radius = 0.125, resulting in a shape composed of three circular arcs and three straight line segments. The fillets are described

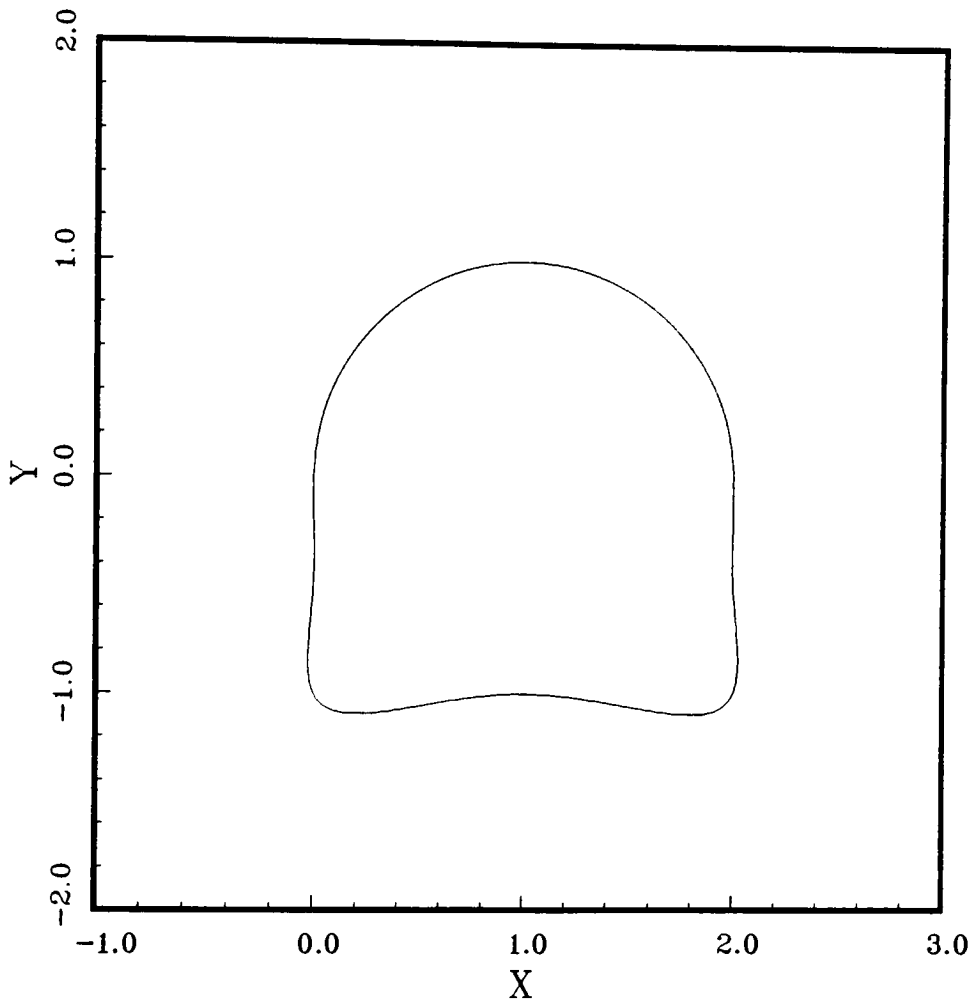

Figure 3.1: Cubic spline with cyclic boundary conditions fitted to a shape composed of a semicircle and a rectangle

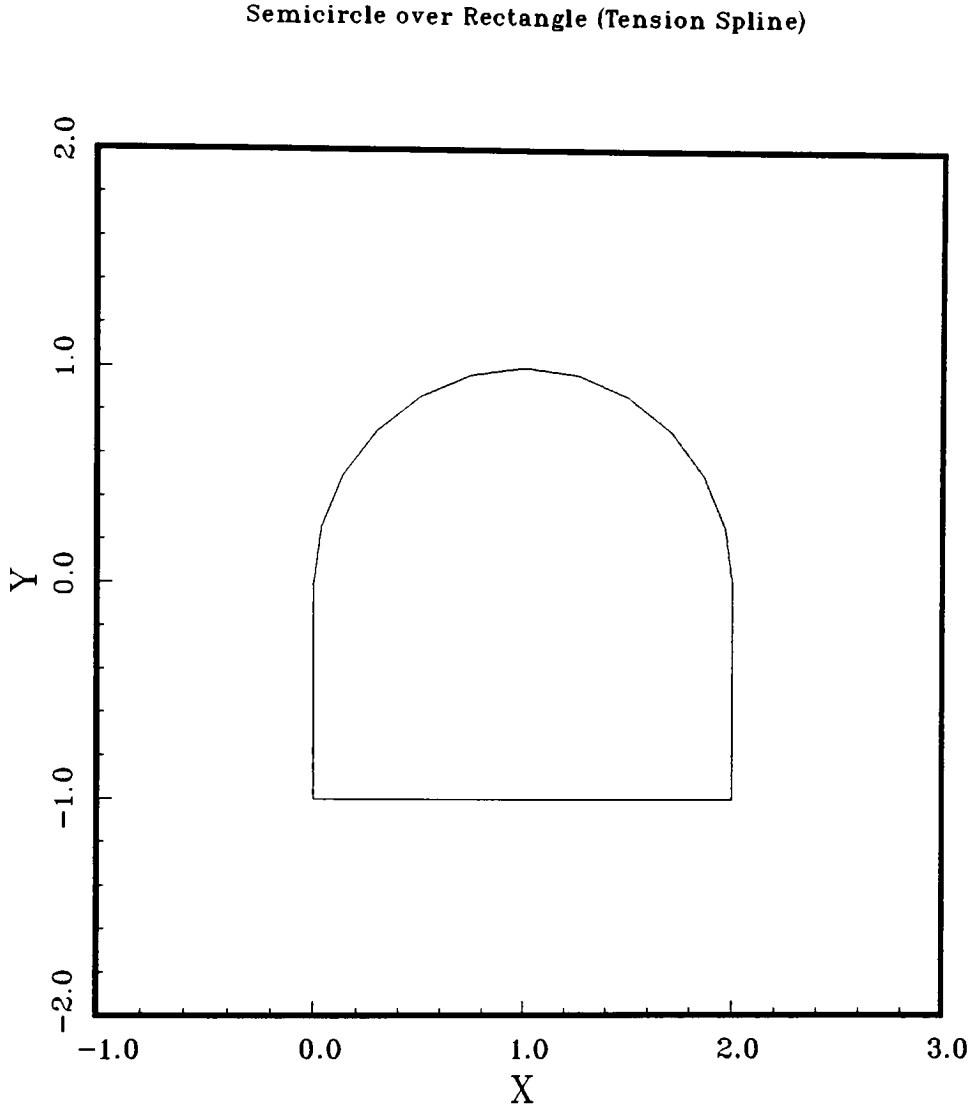

Figure 3.2: Tension spline with cyclic boundary conditions and normalized tension parameter $\gamma = 0.001$ fitted to a shape consisting of a semicircle and a rectangle

by points at 30° intervals resulting in a total of 24 points listed below:

i	0	1	2	3	4
x_i	0.0	0.03407417	0.133975	0.2928932	0.5
y_i	0.0	0.258819	0.5	0.7071068	0.866025
i	5	6	7	8	9
x_i	0.741181	1.0	1.258819	1.5	1.707107
y_i	0.9659258	1.0	0.9659258	0.866025	0.7071068
i	10	11	12	13	14
x_i	1.866025	1.965926	2.0	2.0	2.0
y_i	0.5	0.258819	0.0	-0.5	-0.875
i	15	16	17	18	19
x_i	1.983253	1.9375	1.875	1.0	0.125
y_i	-0.9375	-0.9832532	-1.0	-1.0	-1.0
i	20	21	22	23	24
x_i	0.0625	0.01674682	0.0	0.0	0.0
y_i	-0.9832532	-0.9375	-0.875	-0.5	0.0

As in the previous section, accumulated chord length is chosen for parameterization, and cyclic boundary conditions are employed. Curve resulting from fitting a cubic spline is shown in figure 3.3. Straight segments of the rectangle are not represented accurately by such a curve. The curve, in comparison to that of figure 3.1, is tighter.

When a spline in tension with normalized tension parameter $\gamma = 0.01$ is fitted to the same set of data points, the curve that results is much tighter (shown in figure 3.4), and perhaps a closer representation of the shape being modeled

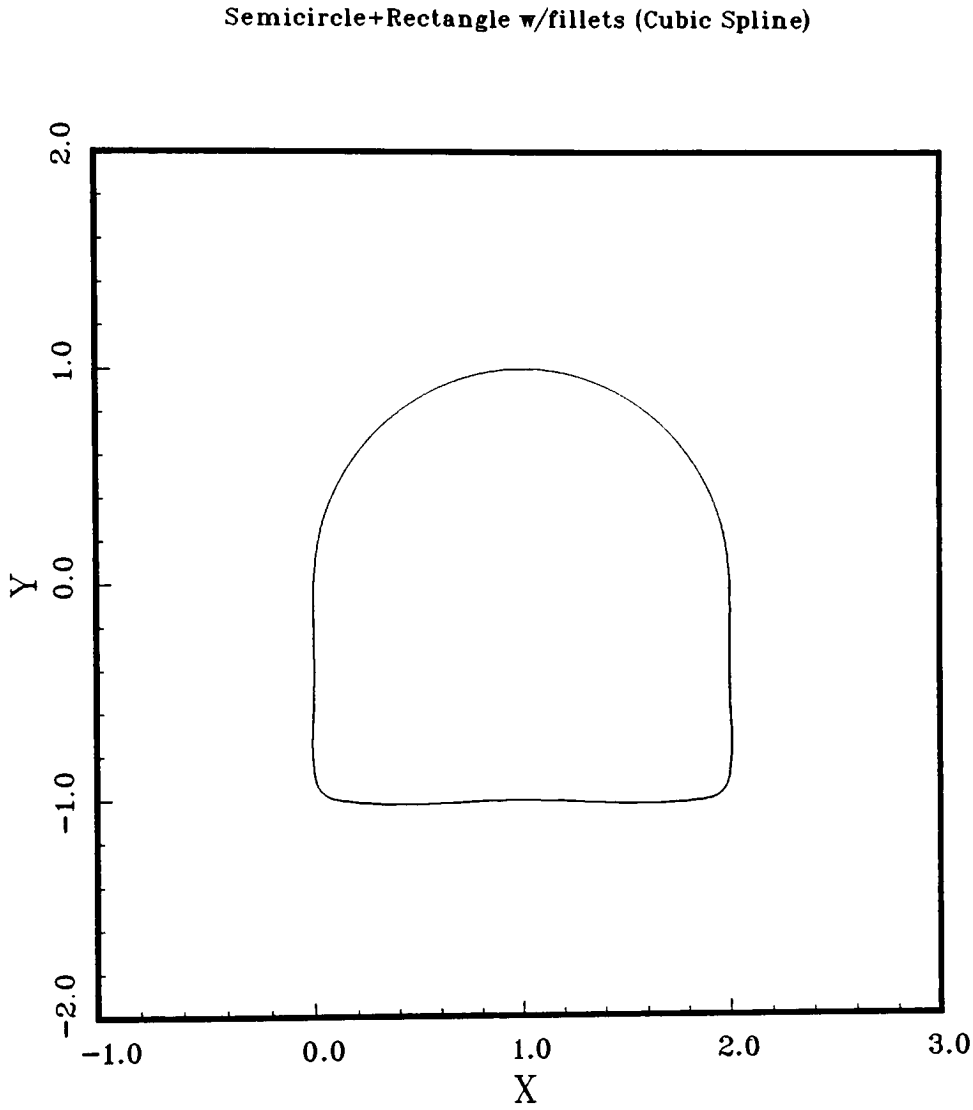

Figure 3.3: Cubic spline with cyclic boundary conditions fitted to data points representing a shape composed of a semicircle and a rectangle with fillets of radii = 0.125

considering that only a few data points have been chosen to represent the semicircular portion and the fillets at rectangular corners.

With shapes such as those discussed in sections 3.1 and 3.2, it is difficult to chose if splines in tension result in better representation of the shape than cubic splines. These static shapes are in any case better described by a collection of circular arcs and straight line segments. Splines in tension are much more useful in describing dynamic shapes in numerical simulation of scientific phenomenon. Such an example is presented in the next section.

3.3 Dislocation Dynamics

In the study of plastic deformation of crystalline materials, dislocation type *line* defects need to be modeled. Since a dislocation (defined as the boundary between slipped and unslipped regions) assumes complex configurations as it glides through the crystal under applied stresses, smooth curves describing these configurations are required to study its glide. The curves can not be expressed analytically, and numerical curve fitting techniques become essential to simulate the motion of dislocations. To simulate such motion, knowledge of both tangent and curvature vectors at the data points describing the dislocation shape is needed. In past, investigators fitted a circular arc to three points (the point of interest and its two neighbors on either side) which yields the information about the tangent and curvature vectors. A more satisfactory method is to fit parametric splines in tension. In fact, since a dislocation experiences self-tension to minimize its energy (*i.e.*, prefers min-

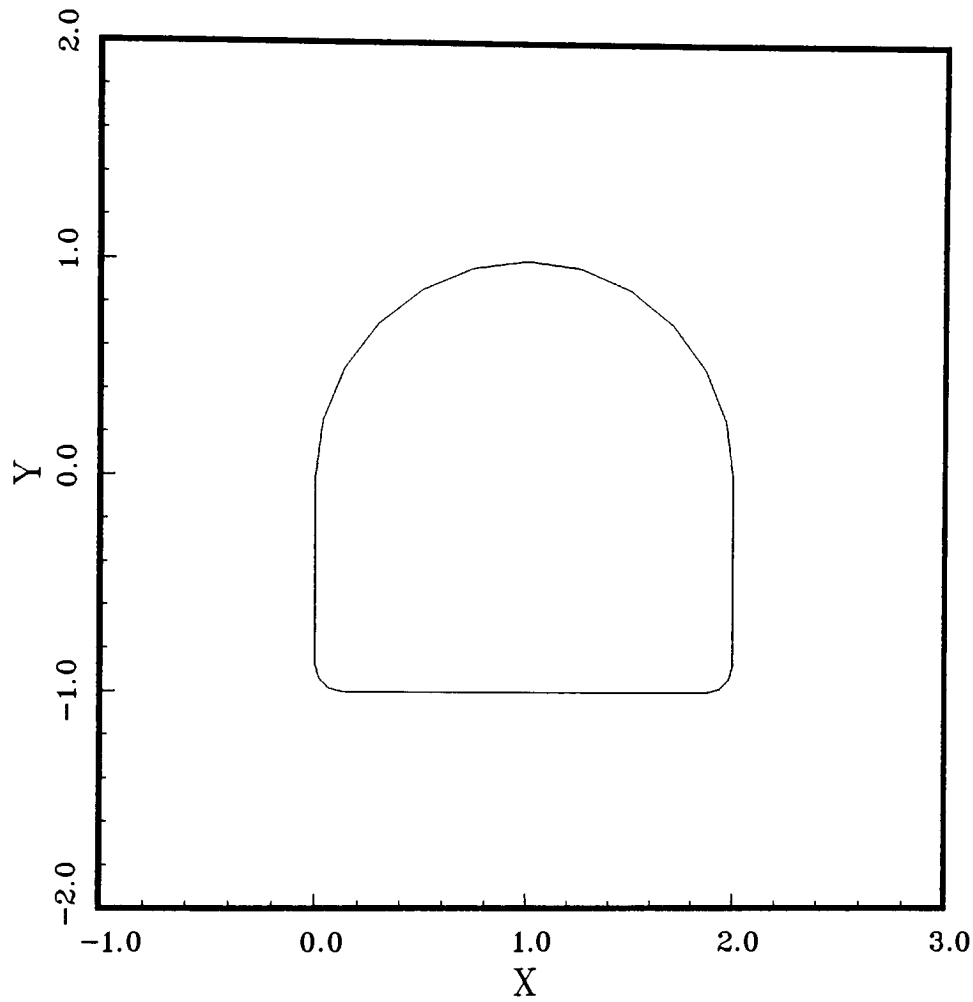

Figure 3.4: Spline in tension with cyclic boundary conditions and normalized tension parameter $\gamma = 0.01$ fitted to data points representing a shape composed of a semicircle and a rectangle with fillets of radii = 0.125

imize its curvature), parametric splines in tension can be considered to be the mathematical analogue of a curved dislocation.

Consider modeling of a double-ended Frank-Read source[18]. Fitting a parametric spline, be it cubic or in tension, poses a problem since two boundary conditions need to be specified to approximate the open curve. For arbitrarily shaped source dislocation, tangent or curvature information at its two ends is unavailable. If open curves are assumed to have constant curvature end spans, parametric splines in tension with constant curvature end conditions approximate such Frank-Read sources well. In figure 3.5, the dynamic shape of such a source at four instants in time is shown. The results of the dynamic simulation correlate well with those determined experimentally[18].

3.4 Shape Descriptions for Machine Vision

Another application of parametric splines in tension is to describe shapes in digital image processing for machine vision applications. In such applications, shape descriptors that are invariant with respect to size, translation and rotation are required. Size invariance is achieved by normalizing each shape dimension by the length of its outer boundary. Barski, Maddigan and Robichaux[3] developed a technique where they compute the angular change from one pixel to the next on the boundary of the image, and represent the boundary by a single real function. The cumulative angular bend is a function of the accumulated arc length. The normalized function is expanded by Fourier series yielding Fourier Descriptors (FDs), each described by an

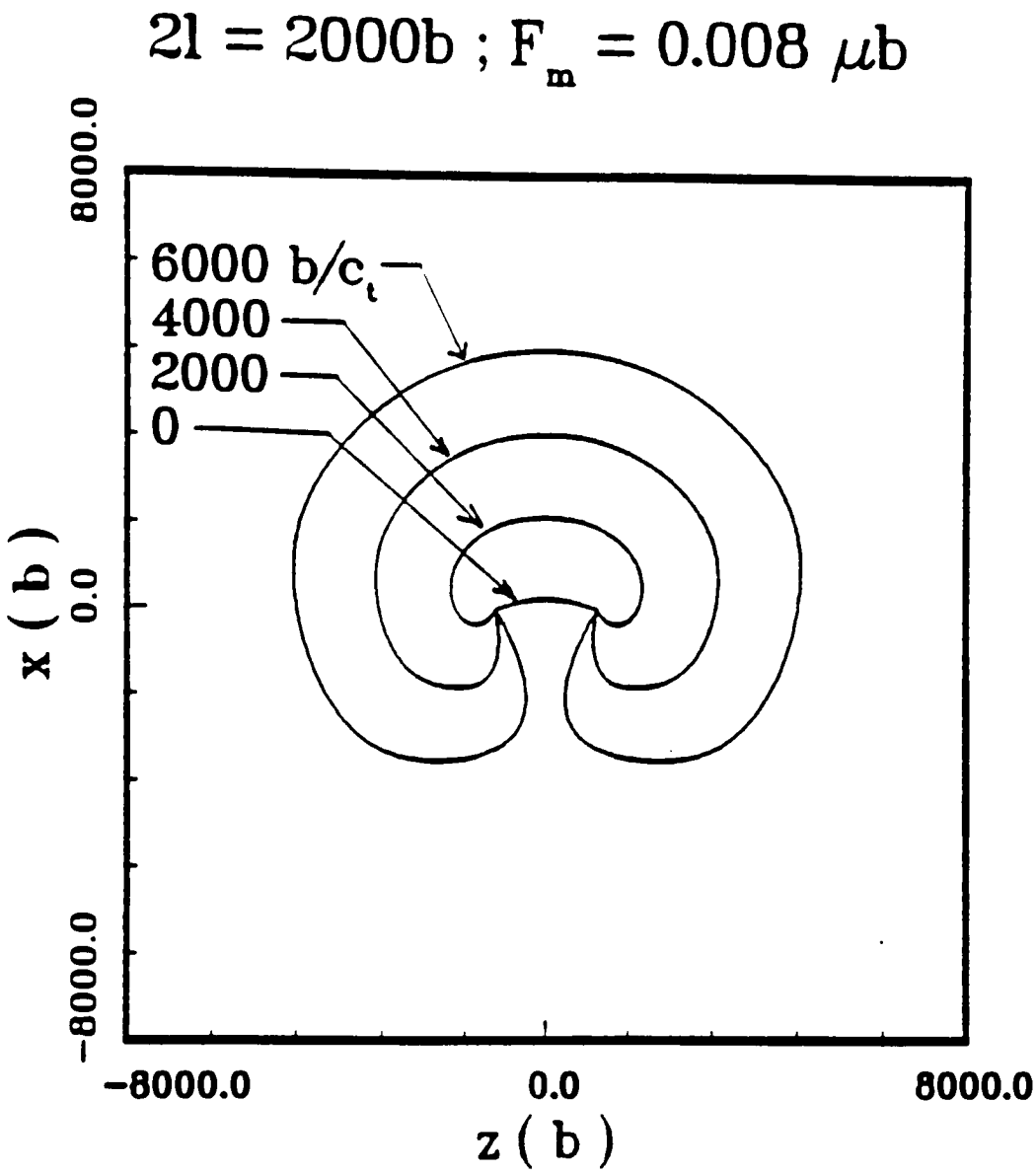

Figure 3.5: Parametric splines in tension with constant curvature end conditions and normalized tension parameter $\gamma = 0.5$ fitted to points describing a Frank-Read source at time $t = 0.\ 2000.\ 4000$ and $6000b/c_t$

amplitude and a phase angle. The amplitudes are independent of rotation, starting point and translation. In principle, a finite number of FDs are sufficient to approximately reconstruct the image. However, they found that the FDs were numerically indistinguishable for similar but different shapes. Also, due to digitizing errors, the same shape rotated more than 30° resulted in a different set of FDs. Since the boundary of a digitized image is fairly represented by a polygon, parametric splines in tension offer a better alternative to compute boundary descriptors. The curvature vector of such a spline at each point (pixel) of the boundary being the descriptor. Furthermore, only a finite number of curvature vectors are required since the boundary has a finite number of pixels. In principle, the difficulty lies in choosing the starting point because two identical shapes with different starting point on the boundary will yield two sets of boundary descriptors differing by a constant phase amount. In practice, these two sets can be reconciled by simple pattern recognition techniques.

3.5 Other Extensions

Several improvements can be made to the technique presented in this thesis. Some of these are minor from both mathematical and implementational point of view.

3.5.1 Adaptive Parametrization

Accumulated chord length parametrization can be iteratively improved. Initially, the parameter is computed based on length of straight line segments connecting the data points, and the parametric spline in tension fitted. The curve fitting can yield the length of curved segments connecting the data points, and a new set of parameter values can be obtained. Spline in tension can now be fitted based on this new set of parameters. This procedure can be repeated until the change in the total length of curve is negligible from one iteration to the next. If the tension parameter is large, this convergence will be achieved quickly. In the extreme case, when the tension parameter is infinite, parametric spline in tension will be composed of straight line segments, and no improvement will be possible after the first iteration.

3.5.2 Variable Tension

Figures 3.2 and 3.4 suggest that to describe known shapes by parametric splines in tension, a single tension parameter applied to the entire curve does not yield the desired shape. For example, the shape in figure 3.2 will be accurately reproduced if the semicircular portion of the curve was fitted by spline with zero tension (*i.e.*, cubic spline) and the rectangular portion by spline with non-zero tension. Thus, by choosing value of tension parameter local to each span, a wider variety of curves can be generated. The derivations of chapter 2 can easily incorporate a tension parameter α_j local to a span j. In practice, however, how would one choose the tension parameters $\{\alpha_j\}_{j=1}^{n}$ to

yield the "desired" curve in a non-interactive environment? In an interactive environment, the user could increase or decrease the tension value on a span till the desired shape is achieved.

Foley[14] develops a method that generalizes the weighted spline (which has first order derivative continuity) with local control of interval tension. Weaver[24] develops the *non-parametric* splines in tension using Euler-Lagrange formulation with tensions local to each span. Weaver suggests obtaining the tension parameters by a non-linear constrained optimization, and acknowledges that the computation time of the method for large number of data points increases very quickly and that for some examples the procedure is extremely sensitive to the choice of starting values for the tension parameters.

3.5.3 Other Approaches

Barsky[5] develops an alternative to splines in tension by introducing another shape parameter in addition to a tension-*like* parameter resulting in Beta-Splines. The second shape parameter is called the *bias* parameter. He suggests that his scheme can be extended to piecewise surfaces. Gregory[17] develops a rational spline alternative to *non-parametric* splines in tension. His treatment requires that the interpolation data be monotonic and of convex shape.

3.6 Concluding Remarks

In this thesis, parametric splines under tension are developed from the Euler-Lagrange formulation. A variety of boundary conditions are explored. These splines are found attractive in description of geometric shapes and in simulation of scientific phenomenon which require curve fitting techniques that yield reasonable tangent and curvature information. Other applications where splines in tension are preferred include machine vision problems where the image boundary needs to be recognized and matched to those in the database and in drawing contour maps. The technique can be easily extended to include adaptive parametrization. For interactive environments, the technique can also be easily extended by offering the user local control of tension on each span.

Appendix A

Splines in Tension

```
! module          main program TENSION
!
! programmer      Vinnie Gupta; 8/15/88
!
! objective       Standard input and output, and driver for
!                 parametrization and spline fitting routines.
!
! input data
!       n         # of data points (numbered 0 through n)
!       d         dimensionality of vector space (< 4)
!       P         array of point coordinate vectors
!       g         normalized tension parameter
!       ptype     type of parametrization
!       btype     type of boundary conditions
!       SP        end first derivative vectors when btype=1
!
```

```
! output data
!       SPP     array of second derivative vectors
!
! ---------------------------------------------------------------------

        program TENSION
        parameter maxn=100, maxd=3
        character*1 ptype, fname*6
        integer*4   btype
        real*4      P(0:maxn,1:maxd), g, SP(1:2,maxd), SPP(0:maxn,maxd)
        real*4      t(0:maxn), r(1:maxd) !parameters
        real*4      A(maxn,maxn), B(maxn,maxd)  !work space
        real*4      c(maxn), s(maxn), h(maxn)   !work space
        common/WORK/ A, B, c, s, h              !work area
        common/BC/   SP                         !boundary conditions

write(6,*) 'Input Filename ?'
read(5,12) fname
open(unit=15, file=fname//'.in', status='OLD', READONLY)
        read(15,*) n, d                          !standard input
        read(15,*) ((P(i,j), j - 1, d), i = 0, n)
        read(15,*) g, ptype, btype
        if (btype .eq. 1) then
                read(15,*) ((SP(i,j), j = 1, d), i = 1,2)
        endif
close (unit=15)

        call PARAMETRIZE(ptype, maxn, maxd, n, d, P, t)

        call TSPLINES(n, d, t, P, g, btype, SPP)

  12 format(a)
open(unit=16, file=fname//'.TMP', status='NEW')
write(16,*) n, d, g
        do 100 i = 0, n                          !standard output
          write(16,11) t(i), (P(i,j), j = 1, d), (SPP(i,j), j   1, d)
 11   format(' ',5e16.6)
```

```
 100    enddo
close(unit=16)

write(6,*) '# of points on each span?'  !to interpolate
read(5,*) m
q = g * n / (t(n) - t(0)) !denormalized tension
q2 = q * q                              !for interpolation

open (unit=16, file=fname//'.int', status='NEW')
write(16,*) (P(0,j), j = 1, d)
do 400 k = 1, n          !for each span

  hk = t(k) - t(k-1) !span length
  dt = hk/m                             !span subdivision
  sh = sinh(q*hk)                       !hyperbolic sin

  do 300 i = 1, m                       !for each subdivision

tt = t(k-1) + i * dt            !parameter value
tl = tt - t(k-1)                !left segment
tr = t(k) - tt                  !right segment
do 200 j = 1, d                 !for each dimension
  r(j) =   SPP(k-1,j) * ( sinh(q*tr)/sh - tr/hk)
     &  + SPP(k,j)   * ( sinh(q*tl)/sh - tl/hk)
     &  + (P(k-1,j) * tr + P(k,j) * tl)/hk
 200 enddo
       write(16,*) (r(j), j = 1, d)    !standard output for plot

 300   enddo

 400 enddo

close(unit=16)

       stop
       end
```

```
!-----------------------------------------------------------------------
!
! module        subroutine PARAMETRIZE
!
! programmer    Vinnie Gupta, 8/11/88
!
! objective     To parametrize the data points as desired
!
!-----------------------------------------------------------------------

      subroutine PARAMETRIZE(ptype, maxn, maxd, n, d, P, t)
      character*1 ptype
      real*4      P(0:maxn, 1:maxd), t(0:maxn)

      if (ptype .eq. 'A') then                !single-valued case
        do 100 i = 0, n
          t(i) = P(i, 1)
100     enddo
        return
      endif

      if (ptype .eq. 'B') then                !use index values
        do 200 i = 0, n
          t(i) = i
200     enddo
        return
      endif

      if (ptype .eq. 'C') then                !Euclidean norm
        t(0) = 0
        do 300 i = 1, n
          chord = 0
          do 250 j   1, d
            chord = chord + (P(i,j)-P(i-1,j))**2
250       enddo
          t(i) = t(i-1) + sqrt(chord)
300     enddo
```

```
          return
        endif

        end

!-----------------------------------------------------------------------
!
! module        subroutine TSPLINES
!
! programmer    Vinnie Gupta; 08/11/88
!
! objective     To fit parametric splines in tension with the desired
!               type of boundary conditions.
!
!-----------------------------------------------------------------------

        subroutine TSPLINES(n, d, t, P, g, btype, SPP)
parameter maxn - 100, maxd = 3
        integer*4 btype
        real*4    P(0:maxn,1:maxd), g, SPP(0:maxn,1:maxd), t(0:maxn)
        common/WORK/ A(maxn,maxn), B(maxn,maxd), c(maxn), s(maxn),
     & h(maxn)
        common/BC/   SP(1:2,1:maxd)

        q = g * n / (t(n)-t(0))                 !tension "alpha"
        do 100 i = 1, n
          h(i)   t(i) - t(i-1)                  !span length
          c(i)   cosh( q * h(i))                !cosh
          s(i) = sinh( q * h(i))                !sinh
 100    enddo

        do 200 i = 1, n-1                       !principal diagonal
          A(i,i) = q*c(i)/s(i) - 1/h(i)
     &           + q*c(i+1)/s(i+1) - 1/h(i+1)
 200    enddo

        do 300 i = 2, n                         !lower diagonal
```

```
          A(i,i-1)   1/h(i) - q/s(i)
300     enddo

        do 400 i = 1, n-1                          !upper diagonal
          A(i,i+1) = 1/h(i+1) - q/s(i+1)
400     enddo

        do 500 i = 1, n-1                          !residual vector
          do 450 j = 1, d
            B(i,j) - (P(i+1,j)-P(i,j))/h(i+1)
     &             - (P(i,j)-P(i-1,j))/h(i)
450       enddo
500     enddo

        if (btype .eq. 1) then                     !clamped splines

          A(1,1) = A(1,1) + (s(1)-q*h(1))/(s(1)-q*h(1)*c(1))
     &                    *(1/h(1)-q/s(1))
          A(n-1,n-1) = A(n-1,n-1) + (s(n)-q*h(n))/(s(n)-q*h(n)*c(n))
     &                            *(1/h(n)-q/s(n))
          do 600 j = 1, d
            B(1,j) = (P(2,j)-P(1,j))/h(2)
     &             - (P(1,j)-P(0,j)) * q
     &             * (1 - c(1))/(s(1) - q*h(1)*c(1))
     &             - SP(1,j)*(s(1)-q*h(1))/(s(1)-q*h(1)*c(1))
            B(n-1,j)   (P(n,j)-P(n-1,j)) * q
     &               * (1 - c(n))/(s(n) - q*h(n)*c(n))
     &               - (P(n-1,j)-P(n-2,j))/h(n-1)
     &               + SP(2,j)*(s(n)-q*h(n))/(s(n)-q*h(n)*c(n))
600       enddo
          call GAUSS(maxn, maxd, n-1, d, A, B, SPP)
          do 700 j = 1, d
            SPP(0,j) = (SP(1,j) - (P(1,j)-P(0,j))/h(1)
     &               + (1/h(1) - q/s(1)) * SPP(1,j) )
     &               / (1/h(1) - q*c(1)/s(1))
            SPP(n,j) = (SP(2,j) - (P(n,j)-P(n-1,j))/h(n)
     &               - (1/h(n)   q/s(n)) * SPP(n-1,j) )
```

```
     &                / (q*c(n)/s(n) - 1/h(n))
700       enddo

        elseif (btype .eq. 2) then              !natural splines

          call GAUSS(maxn, maxd, n-1, d, A, B, SPP)
          do 800 j = 1, d
            SPP(0,j) = 0
            SPP(n,j) = 0
800       enddo

        elseif (btype .eq. 3) then              !cyclic splines

          A(1,n)   1/h(1) - q/s(1)
          A(n,1) = 1/h(1) - q/s(1)
          A(n,n-1) = 1/h(n) - 1/s(n)
          A(n,n) = q*c(1)/s(1) - 1/h(1)
     &           + q*c(n)/s(n) - 1/h(n)
          do 900 j = 1, d
            B(n,j) = (P(1,j)-P(0,j))/h(1) - (P(n,j)-P(n-1,j))/h(n)
900       enddo
          call GAUSS(maxn, maxd, n, d, A, B, SPP)
          do 1000 j = 1, d
            SPP(0,j) = SPP(n,j)
1000      enddo

        endif

        q2 = q * q                              !square(alpha)
        do 1200 i   0, n                        !denormalize vectors
          do 1100 j = 1, d
            SPP(i,j) = SPP(i,j) * q2
1100      enddo
1200    enddo

        return
        end
```

```
!-------------------------------------------------------------------
!
! module        subroutine GAUSS
!
! programmer    Vinnie Gupta; 08/11/88
!
! objective     To solve a system of equations using the modified
!               Gaussian elimination method.
!
!-------------------------------------------------------------------

        subroutine GAUSS(maxn, maxd, n, d, A, B, S)
        real*4 A(maxn,maxn), B(maxn,maxd), S(0:maxn,maxd)
        real*4 mult

        do 400 k = 1, n-1                          !each step k
          do 300 i = k+1, n                        !each row i
            mult = a(i,k)/a(k,k)
            if (mult .ne. 0) then
              do 100 j = k+1, n                    !each column j
                a(i,j) = a(i,j) - mult * a(k,j)
100           enddo
              do 200 l = 1, d                      !each component l
                  b(i,l) = b(i,l) - mult * b(k,l)
200           enddo
            endif
300       enddo
400     enddo

        do 1000 l = 1, d                           !each component l
          do 900 i   n, 1, -1                      !each unknown i
            do 800 j = i+1, n                      !each term j
              b(i,l) = b(i,l) - a(i,j) * s(j,l)
800         enddo
            s(i,l) = b(i,l)/a(i,i)
900       enddo
```

```
1000    enddo

        return
        end
```

Appendix B

Cubic Splines

```
! module          main program CUBIC
!
! programmer      Vinnie Gupta; 8/15/88
!
! objective       Standard input and output, and driver for
!                 parametrization and spline fitting routines.
!
! input data
!       n         # of data points (numbered 0 through n)
!       d         dimensionality of vector space (< 4)
!       P         array of point coordinate vectors
!       ptype     type of parametrization
!       btype     type of boundary conditions
!       SP        end first derivative vectors when btype=1
!
! output data
```

```
!       SPP     array of second derivative vectors
!
! ---------------------------------------------------------------------

        program CUBIC
        parameter maxn=100, maxd=3
        character*1 ptype, fname*6
        integer*4   btype
        real*4      P(0:maxn,1:maxd), g, SP(1:2,maxd), SPP(0:maxn,maxd)
        real*4      t(0:maxn), r(1:maxd) !parameters
        real*4      A(maxn,maxn), B(maxn,maxd)  !work space
        real*4      c(maxn). s(maxn), h(maxn)   !work space
        common/WORK/ A, B, c, s, h              !work area
        common/BC/   SP                         !boundary conditions

write(6,*) 'Input Filename ?'
read(5,12) fname
open(unit=15, file=fname//'.in', status='OLD', READONLY)
        read(15,*) n, d                          !standard input
        read(15,*) ((P(i,j), j = 1, d), i = 0, n)
        read(15,*) g, ptype, btype
        if (btype .eq. 1) then
                read(15,*) ((SP(i,j), j = 1, d), i   1,2)
        endif
close (unit=15)

        call PARAMETRIZE(ptype, maxn, maxd, n, d, P, t)

        call CSPLINES(n, d, t, P. btype, SPP)

  12 format(a)
open(unit=16, file=fname//'.TMP', status='NEW')
write(16,*) n, d, g
        do 100 i = 0, n                         !standard output
          write(16,11) t(i), (P(i,j), j = 1, d), (SPP(i,j), j = 1, d)
 11   format(' ',5e16.6)
 100    enddo
```

```
close(unit=16)

write(6,*) '# of points on each span?'  !to interpolate
read(5,*) m
q = g * n / (t(n)   t(0)) !denormalized tension
q2 = q * q                              !for interpolation

open (unit=16, file=fname//'.int', status='NEW')
write(16,*) (P(0,j), j = 1, d)
do 400 k = 1, n          !for each span

  hk = t(k) - t(k-1) !span length
  dt = hk/m                             !span subdivision

  do 300 i = 1, m                       !for each subdivision

tt = t(k-1) + i * dt            !parameter value
tl = tt - t(k-1)                !left segment
tr = t(k) - tt                  !right segment
do 200 j = 1, d                 !for each dimension
  r(j) =   (P(k-1,j)*tr + P(k,j)*tl)/hk
     &  - tr * tl / 6 / hk
     &  * (hk * (SPP(k-1,j) + SPP(k,j))
     &  + tr * SPP(k-1,j) + tl * SPP(k,j) )
 200 enddo
       write(16,*) (r(j), j - 1, d)    !standard output for plot

 300   enddo

 400 enddo

close(unit=16)

       stop
       end
```

```
!-----------------------------------------------------------------------
!
! module        subroutine PARAMETRIZE
!
! programmer    Vinnie Gupta, 8/11/88
!
! objective     To parametrize the data points as desired
!
!-----------------------------------------------------------------------

      subroutine PARAMETRIZE(ptype, maxn, maxd, n, d, P, t)
      character*1 ptype
      real*4      P(0:maxn, 1:maxd), t(0:maxn)

      if (ptype .eq. 'A') then                !single-valued case
        do 100 i = 0, n
          t(i) = P(i, 1)
100     enddo
        return
      endif

      if (ptype .eq. 'B') then                !use index values
        do 200 i   0, n
          t(i)   i
200     enddo
        return
      endif

      if (ptype .eq. 'C') then                !Euclidean norm
        t(0) = 0
        do 300 i = 1, n
          chord = 0
          do 250 j = 1, d
            chord = chord + (P(i,j)-P(i-1,j))**2
250       enddo
          t(i)   t(i-1) + sqrt(chord)
300     enddo
```

```
          return
        endif

        end

!-----------------------------------------------------------------------
!
! module        subroutine CSPLINES
!
! programmer    Vinnie Gupta; 08/11/88
!
! objective     To fit parametric cubic splines with the desired type
!               of boundary conditions.
!
!-----------------------------------------------------------------------

        subroutine CSPLINES(n, d, t, P, btype, SPP)
parameter maxn=100, maxd=3
        integer*4 btype
        real*4    P(0:maxn,1:maxd), SPP(0:maxn,1:maxd), t(0:maxn)
        common/WORK/ A(maxn,maxn). B(maxn,maxd), c(maxn), s(maxn),
     &               h(maxn)
        common/BC/   SP(1:2,1:maxd)

        do 100 i = 1, n
          h(i) = t(i)   t(i-1)                  !span length
 100    enddo

        do 200 i = 1, n-1
          A(i,i) = 2 * (h(i) + h(i+1))          !principal diagonal
 200    enddo

        do 300 i = 2, n
          A(i,i-1) = h(i)                       !lower diagonal
 300    enddo

        do 400 i = 1, n-1
```

```
          A(i,i+1) = h(i+1)                          !upper diagonal
400     enddo

        do 500 i = 1, n-1
          do 450 j = 1, d                            !residual vector
            B(i,j) = 6 * ( (P(i+1,j)-P(i,j))/h(i+1)
     &                  - (P(i,j)-P(i-1,j))/h(i) )
450       enddo
500     enddo

        if (btype .eq. 1) then                       !clamped splines

          A(1,1) = A(1,1)   0.5 * h(1)
          A(n-1,n-1) = A(n-1,n-1)   0.5 * h(n)

          do 600 j = 1, d
            B(1,j) = B(1,j) - 3 * ( (P(1,j)-P(0,j))/h(1)
     &                             SP(1,j) )
            B(n-1,j) = B(n-1,j) - 3 * ( SP(2,j)
     &                               - (P(n,j)-P(n-1,j))/h(n) )
600       enddo
          call GAUSS(maxn, maxd, n-1, d, A, B, SPP)
          do 700 j = 1, d
            SPP(0,j) = 3 * ((P(1,j)-P(0,j))/h(1) - SP(1,j))/h(1)
     &                 - SPP(1,j)/2
            SPP(n,j) = 3 * (SP(2,j) - (P(n,j)-P(n-1,j))/h(n))/h(n)
     &                 - SPP(n-1,j)/2
700       enddo

        elseif (btype .eq. 2) then                   !natural splines

          call GAUSS(maxn, maxd, n-1, d, A, B, SPP)
          do 800 j = 1, d
            SPP(0,j) = 0
            SPP(n,j) = 0
800       enddo
```

```
        elseif (btype .eq. 3) then              !cyclic splines

          A(1,n) = h(1)
          A(n,1) = h(1)
          A(n,n-1) = h(n)
          A(n,n) = 2 * (h(1) + h(n))
          do 900 j = 1, d
            B(n,j)   6 * ( (P(1,j)-P(0,j))/h(1)
     &                    - (P(n,j)-P(n-1,j))/h(n) )
 900      enddo
  write(6,*) 'Reached before calling gauss'
  do 910 i = 1, n
  write(6,901) (a(i,j), j   1, n)
  write(6,*) (b(i,j), j   1,d)
 901   format(' ',7f10.4)
 910   enddo

          call GAUSS(maxn, maxd, n, d, A, B, SPP)
  write(6,*) 'Completed Gauss'
          do 1000 j   1, d
            SPP(0,j) = SPP(n,j)
1000      enddo

        endif

        return
        end
!-----------------------------------------------------------------------
!
! module        subroutine GAUSS
!
! programmer    Vinnie Gupta; 08/11/88
!
! objective     To solve a system of equations using the modified
!               Gaussian elimination method.
!
!-----------------------------------------------------------------------
```

```
      subroutine GAUSS(maxn, maxd, n, d, A, B, S)
      real*4 A(maxn,maxn), B(maxn,maxd), S(0:maxn,maxd)
      real*4 mult

      do 400 k = 1, n-1                          !each step k
        do 300 i = k+1, n                        !each row i
          mult = a(i,k)/a(k,k)
          if (mult .ne. 0) then
            do 100 j = k+1, n                    !each column j
              a(i,j) = a(i,j) - mult * a(k,j)
100         enddo
            do 200 l = 1, d                      !each component l
                b(i,l) = b(i,l) - mult * b(k,l)
200         enddo
          endif
300     enddo
400   enddo

      do 1000 l = 1, d                           !each component l
        do 900 i = n, 1, -1                      !each unknown i
          do 800 j = i+1, n                      !each term j
            b(i,l) = b(i,l) - a(i,j) * s(j,l)
800       enddo
          s(i,l) = b(i,l)/a(i,i)
900     enddo
1000  enddo

      return
      end
```

Appendix C

Bibliography

The following books were helpful in researching the topic of curve fitting and solution of a system of linear equations:

1. M. L. James, G. M. Smith and J. C. Wolford, *Applied Numerical Methods for Digital Computation*, Harper & Row, New York, Third Edition (1985).

2. C. F. Gerald, *Applied Numerical Analysis*, Addison Wesley, Second Edition (1980).

3. R. L. Burden and J. D. Faires, *Numerical Analysis*, Prindle, Weber & Schmidt, Boston, Third Edition (1985).

4. D. F. Rogers and J. A. Adams, *Mathematical Elements of Computer Graphics*, McGraw Hill, New York (1976).

5. I. D. Fox and M. J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis Harwood Limited, Chichester (1980).

6. S. H. Chasen, *Geometric Principles and Procedures for Computer Graphic Applications*, Prentice-Hall (1978).

References

[1] J. A. Adams, *Cubic spline curve fitting with controlled end conditions*, **Computer Aided Design**, 6(1974)2–9.

[2] J. H. Ahlberg, E. N. Nilson and J. L. Walsh, *The theory of splines and their applications*, Academic Press, London (1967).

[3] L. Barski, J. Maddigan and J. Robichaux, **Shape Recognition System**, Senior Design Project II, Department of Mechanical Engineering, Rochester Institute of Technology, Spring (1989).

[4] B. A. Barsky, *Exponential and polynomial methods for applying tension to an interpolating spline curve*, **Computer Vision, Graphics and Image Processing**, v27(1984)1–18.

[5] Brian A. Barsky, **Computer Graphics and Geometric Modeling using Beta-Splines**, Springer-Verlag (1988).

[6] W. Boehm, G. Farin and J. Kahmann, *A survey of curve and surface methods in CAGD*, **Computer Aided Geometric Design**, 1(1984)1–60.

[7] K. W. Brodlie, *A review of methods for curve and function drawing* in **Mathematical Methods in Computer Graphics and Design** (ed. K. W. Brodlie), Academic Press, London (1980)1–37.

[8] R. L. Burden and J. D. Faires in **Numerical Analysis**, third edition, Prindle, Weber & Schmidt, Boston, (1985)290–360.

[9] A. K. Cline, *Scalar- and planar-valued curve fitting using splines under tension*, **Comm. ACM**, **17**(1974)218–223.

[10] E. Cohen, T. Lyche and R. F. Riesenfeld, *Discrete B-Splines and subdivision techniques in computer aided geometric design and computer graphics*, **Computer Graphics and Image Processing**, 14(1980)87–111.

[11] M. G. Cox, *An algorithm for spline interpolation*, J. Inst. Maths. Applics., **15**(1975)95–108.

[12] M. G. Cox, *A survey of numerical methods for data and function approximation* in **The state of the art in numerical analysis** (ed. D. A. H. Jacobs), Academic Press, London (1977)627–668.

[13] M. P. Epstein, *On the influence of parametrization in parametric interpolation*, **SIAM J. Numer. Anal.**, **13**(1976)261–268.

[14] T. A. Foley, *Local control of interval tension using weighted splines*, **Computer-aided Geometric Design**, v3(1986)281–294.

[15] *GHOST User Manual*, Report # CLM-R177, Culham Laboratory (1978).

[16] *GINO-F User Manual*, Computer Aided Design Centre, Cambridge (1975).

[17] J. A. Gregory, *Shape preserving spline interpolation*, **Computer-aided Design**, v18#1(1986)53–57.

[18] S. K. Gupta, *Dislocation Dynamics in Frank-Read Source Operation*, PhD Thesis, Department of Mechanical Engineering, University of Rochester (1988).

[19] Z. D. Jaztrezbski, *The Nature and Properties of Engineering Materials*, John Wiley & Sons, (1987)111–118.

[20] G. M. Nielson, *Some piecewise polynomial alternatives to splines under tension* in **Computer Aided Geometric Design** (eds. R. E. Barnhill and R. F. Riesenfeld), (1974)209–235.

[21] A. W. Nutbourne, *A cubic spline package, part 2—the mathematics*, **Computer Aided Design**, 5#1(1973)7–13.

[22] D. G. Schweikert, *An interpolation curve using a spline in tension*, **J. Math. and Phy.**, 45(1966)312–317.

[23] N. E. South and J. P. Kelly, *Analytic Surface Methods*, Ford Company NC Development Unit, Unit Engineering Office, December (1965).

[24] U. Weaver, *Non-negative exponential splines*, **Computer-aided Design**, v20#1(1988)11–16.