NAVAL POSTGRADUATE SCHOOL
Monterey, California
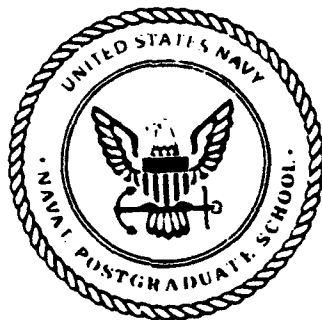
THESIS

DTIC
ELECTE
APR 1 2 1994
S **D**
G

LEAST SQUARES APPROXIMATION
BY G¹ PIECEWISE PARAMETRIC CUBICS

by

Marion R. Holmes

December 1993

Thesis Advisor: Richard Franke

Approved for public release; distributed is unlimited.

DTIC QUALITY INSPECTED 3

94 4 11 070

REPORT DOCUMENTATION PAGE

Form Approved OMB Np. 0704

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE December 1993	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE LEAST SQUARES APPROXIMATION BY G¹ PIECEWISE PARAMETRIC CUBICS	5. FUNDING NUMBERS
6. AUTHOR(S) HOLMES, Marion R.	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

Parametric piecewise cubic polynomials are used throughout the computer graphics industry to represent geometric curved shapes. The exploration of the use of parametric curves and surfaces can be viewed as the birth of Computer Aided Geometric Design (CAGD). In this thesis, least squares approximation is used for fitting a geometrically continuous (G^1) piecewise parametric cubic polynomial to a sequence of ordered points in the plane. Cubic Bézier curves are used as a basis. The parameterization, the control points, the number of knots, and their locations are determined as part of the approximation process. A development of the algorithm is given, along with some results for a variety of sets of ordered data.

14. SUBJECT TERMS Parametric continuity, Geometric continuity, Knot point, Control point, Bézier curve, Least squares, Approximation, Piecewise cubic, Interpolation	15. NUMBER OF PAGES 70
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20.LIMITATION OF ABSTRACT UL

LEAST SQUARES APPROXIMATION
BY G¹ PIECEWISE PARAMETRIC CUBICS

Marion R. Holmes
Lieutenant, United States Navy
B.S., Morehouse College, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS
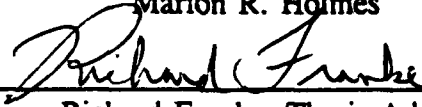
from the

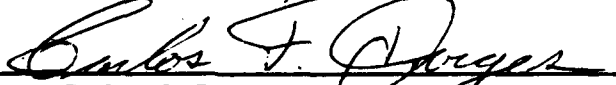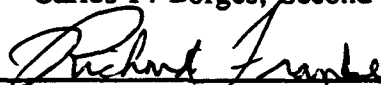## NAVAL POSTGRADUATE SCHOOL
December 1993

Author: _____
Marion R. Holmes

Approved by: _____
Richard Franke, Thesis Advisor

_____
Carlos F. Borges, Second Reader

_____
Richard Franke, Chairman,
Department of Mathematics

ii

ABSTRACT

Parametric piecewise cubic polynomials are used throughout the computer graphics industry to represent geometric curved shapes. The exploration of the use of parametric curves and surfaces can be viewed as the birth of Computer Aided Geometric Design (CAGD). In this thesis, least squares approximation is used for fitting a geometrically continuous (G^1) piecewise parametric cubic polynomial to a sequence of ordered points in the plane. Cubic Bézier curves are used as a basis. The parameterization, the control points, the number of knots, and their locations are determined as part of the approximation process. A development of the algorithm is given, along with some results for a variety of sets of ordered data.

Accesion For		
NTIS CRA&I	☒	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

TABLE OF CONTENTS

ACKNOWLEDGEMENT

This thesis is dedicated to my parents, Mr. and Mrs. Robert L. Holmes, who encouraged higher education, and to my thesis advisor and second reader, who inspired me greatly and without which their efforts this document would not have been possible.

I. INTRODUCTION

A. GENERAL

In many CAGD applications, a user wishes to produce a smooth curve from a given ordered set of data points such that the results captures the general shape of the data. This is no easy problem as many numerical analysts have discovered. There are several types of approximations that can be used to solve this problem. In simple cases polynomial interpolation is generally used. In advanced cases, piecewise polynomials are used. These methods are incorporated into least-squares data fitting in approximating curved shapes from a set of discrete data points. When parametric curves are used, an appropriate parameterization is essential in obtaining a good representation of the original shape.

The method of fitting given data by a polynomial which coincides with the data at certain specified points is in some cases a rather inefficient one. In particular, when the function is known at all points in an interval, it seems undesirable to select only a relatively small set of arbitrarily chosen points at which to "match", Weeg [Ref. 1: p. 126]. Obviously, using a large number of points results in a prohibitively high degree polynomial for numerical computations. Another undesirable case is one in which the degree of reliability of such values is not clearly established, as is often the case for experimental data.

The method of least squares is designed to treat these two classes of problems. The usual criterion of least-squares approximation is to minimize the sum of the squares of the errors within a specified domain. Detailed explanations of least

1

squares data-fitting can be found in most books on "Numerical Analysis", [Ref. 1-5].

Polynomial interpolation is the most fundamental of all approximation concepts. The algebraic polynomials $p_n(x)$ are by far the most important and popular approximating functions, Carnahan [Ref. 2: p. 2]. The theory of polynomial interpolation is well developed and fairly simple. Polynomials are easy to evaluate and their sums, products, and differences are also polynomials. Polynomials can be differentiated and integrated with little difficulty. In addition, if the origin of the coordinate system is shifted or if the scale of the independent variable is changed, the transformed polynomials remain polynomials, that is, if $p_n(x)$ is a polynomial, so are $p_n(x+a)$ and $p_n(ax)$, Ralston [Ref. 3: p. 34].

All these advantages of the polynomial would be of little value if there were no analytical justification for believing that polynomials can, in fact, yield good (implies the error can be made arbitrarily small) approximations for a given function $f(x)$. This theoretical justification does exist; any continuous function $f(x)$ can be approximated to any desired degree of accuracy on a specified closed interval by some polynomial $p_n(x)$. This follows from the Weierstrass approximation theorem stated here:

If $f(x)$ is continuous in the closed interval [a,b], then given any $\epsilon > 0$, there is some polynomial $p_n(x)$ of degree $n = n(\epsilon)$ such that

$$|f(x)-p_n(x)| < \epsilon, \quad x \in [a,b].$$

It is important to note that the usual criteria for generating interpolating polynomials in no way guarantees that the polynomial found is the one which the Weierstrass

2

theorem shows must exist. This is why polynomial interpolation is mostly of theoretical value, while faster and more accurate methods, piecewise polynomials, are used extensively in approximation analysis.

Piecewise polynomials are more widely used in least-squares data fitting since they are more flexible than simple polynomials. If we wish to approximate to a function f on [a,b], we may divide [a,b] into N sub-intervals, by introducing points of sub-division

$$a = x_0 < x_1 < \ldots < x_n = b,$$

and use some form of approximating or interpolating polynomial on each sub-interval $[x_{i-1}, x_i]$, $1 \leq i \leq N$. This is called piecewise polynomial approximation. Phillips [Ref. 4: p. 113].

Generally, the approximating polynomials used on neighboring intervals $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$ will not take the same value at the common point x_i. It is natural to seek piecewise approximations which are continuous and whose first few derivatives are also continuous at the nodes of subdivision, so that the connection appears smooth. These are called spline approximations. The most common piecewise polynomial approximation using cubic polynomials between each successive pair of nodes is cubic spline interpolation. A piecewise cubic polynomial approximation to a curved shape consists of a number of single cubic segments connected end-to-end to form a continuous curve. A general cubic polynomial involves four constants; so there is sufficient flexibility in the cubic spline procedure to ensure not only that the interpolant is continuous on the interval, but also that it has

a continuous first and second derivative on the interval. But, as stated by Burden [Ref. 5: p. 132] the construction of the cubic spline does not assume that the derivatives of the interpolant agree with those of the function, even at the nodes.

If x and y are given as functions

$$x = f(t) \qquad y = g(t)$$

over an interval of t-values, then the set of points $(x,y) = (f(t),g(t))$ defined by these equations is called a curve in the coordinate plane. The equations are parametric equations for the curve. The variable t is called the parameter of the curve. When we write parametric equations for the curve in the plane, we say that we have parameterized the curve. For example, a parametric equation of the unit circle is as follows:

$$x = \cos t \quad y = \sin t \quad 0 \le t \le 2\pi .$$

The spline interpolation problem is usually stated as "given data points q_i and parameter values $u_i,...$". The simplest way to determine the u_i is to set $u_i = i$. This is called uniform or equidistant parameterization, Farin [Ref. 6: p. 130]. This method is not useful in most practical situations. The reason is that uniform parameterization does not take into account the geometry of the data points, for an heuristic explanation see Farin [Ref. 6: p. 130]. There are exceptions in which uniform parameterization fares better than other methods, according to Foley [Ref. 7: p. 86].

The most commonly used parameterization or knot spacing is chord length spacing, where $h_i = |q_{i+1} - q_i|$ (Euclidean distance). Some authors have

4

suggested using chord length because it approximates the arc length of a parametric curve, for example Ahlberg [Ref. 8: pp. 254-258]. Another advantage for chord length parameterization, as proved by Epstein [Ref. 9: pp. 261-268], is that it guarantees there will be no cusps for the case of a closed periodic curve. When the data is poorly scaled or when the direction of the data changes abruptly, the chord length knot spacing often produces visually poor results, as seen in Foley and Nielson [Ref. 10: pp. 261-271]. A curve may be parametrized in many ways and the choice of parametization is important as discussed by de Boor [Ref. 11: pp. 315-318]. There is probably no "best" parametrization since most known methods can be defeated by a suitably chosen data set. For our algorithm, chord length parametrization was chosen as an initial guess.

The problem presented here is the following:

Given an ordered set m of data points $Q_i = (x_i, y_i)$ for $i = 1, ..., m$, we wish to fit a piecewise Bézier cubic polynomial $B(t) = (x(t), y(t))$ to Q_i that minimizes the sum of squares of distances from the data points to the nearest point on the piecewise Bézier curve. The optimization routine "fmins" minimizes this sum by optimizing the free parameters (i.e., knot points, control points, unit tangent vectors). In particular we wish to fit geometrically continuous (G^1) curves, rather than parametrically continuous (C^1) curves, generally because the former is a less restrictive condition.

B.   CONTINUITY/PARAMETRIC AND GEOMETRIC

Parametric continuity (C^r) of a curve means that the component functions are r times continuously differentiable in a given interval [a,b]. Therefore, C^1 continuity implies that the component functions are one time continuously differentiable. Furthermore, C^1 continuity is based on the interplay between domain and range configurations. Without any information on the domain of a curve, we cannot make any statements concerning differentiability. In another case, zero tangent vectors may give rise to corners or cusps in curves (which are C^1), a fact that intuitively contradicts the concept of differentiability. **Smoothness and differentiability only agree for functional curves, the connection between them is lost in the parametric case.** Differentiable curves may not be smooth and smooth curves may not be differentiable. We emphasize that by "smooth", we mean "perceptually smooth".

Geometric continuity of order r for a curve implies that the curve is r times differentiable with respect to arc length, Farin [Ref. 6]. It also concerns how parametric curves and surfaces can be pieced together in a smooth way. If a curve has continuous curvature, it is G^2 (second order geometrically continuous). Curves with a zero tangent vector under the given parametrization, can be G^2, for example the parametric curve $x = t^3$, $y = t^3$ is G^2 at $t = 0$. A tutorial view of the concepts of geometric continuity within the subject of CAGD is presented by Gregory [Ref. 12: pp. 353-371]. The following examples illustrate the difference between these two concepts:

(1) Consider the parametric curve defined by

$$x = t^3, \quad y = |t^3|, \quad t \in [-1,1].$$

$$\frac{dy}{dt} = \frac{d(|t|^3)}{dt} = 3|t|^2 \frac{d(|t|)}{dt} = 3|t^2| \cdot \frac{t}{|t|} = 3|t|t.$$

Note that $y = |x|$. This curve is C^1 but not G^1, see Figure 1.

(2) Consider the Bézier parametric curves for the control points $(0,0)$, $(0,3)$, $(3,3)$,

and $(3,3)$, $(6,3)$, $(6,0)$ respectively.

<div style="margin-left:2em">

(a) $x(t) = 3t^2$ $y(t) = 6t-3t^2$ $t \in [0,1]$; and

(b) $x(t) = 12t-3t^2-6$ $y(t) = 6t-3t^2$ $t \in [1,2]$.

</div>

For curve (a),

$$\frac{dx}{dt} = 6t \qquad \frac{dy}{dt} = 6-6t,$$

$$\frac{d^2x}{dt^2} = 6 \qquad \frac{d^2y}{dt} = -6$$

For curve (b),

$$\frac{dx}{dt} = 12-6t, \quad \frac{d^2x}{dt^2} = -6, \quad \frac{dy}{dt} \text{ and } \frac{d^2y}{dt^2} \text{ is same as (a)}.$$

Thus, $y(t)$ is infinitely differentiable, but $x(t)$ is not twice differentiable at $t=1$; as

shown above $x''(1) = 6$ for (a) and $x''(1) = -6$ for (b). However, the curve does

have continuous curvature at $t=1$, as shown below. The curvature of a plane curve is
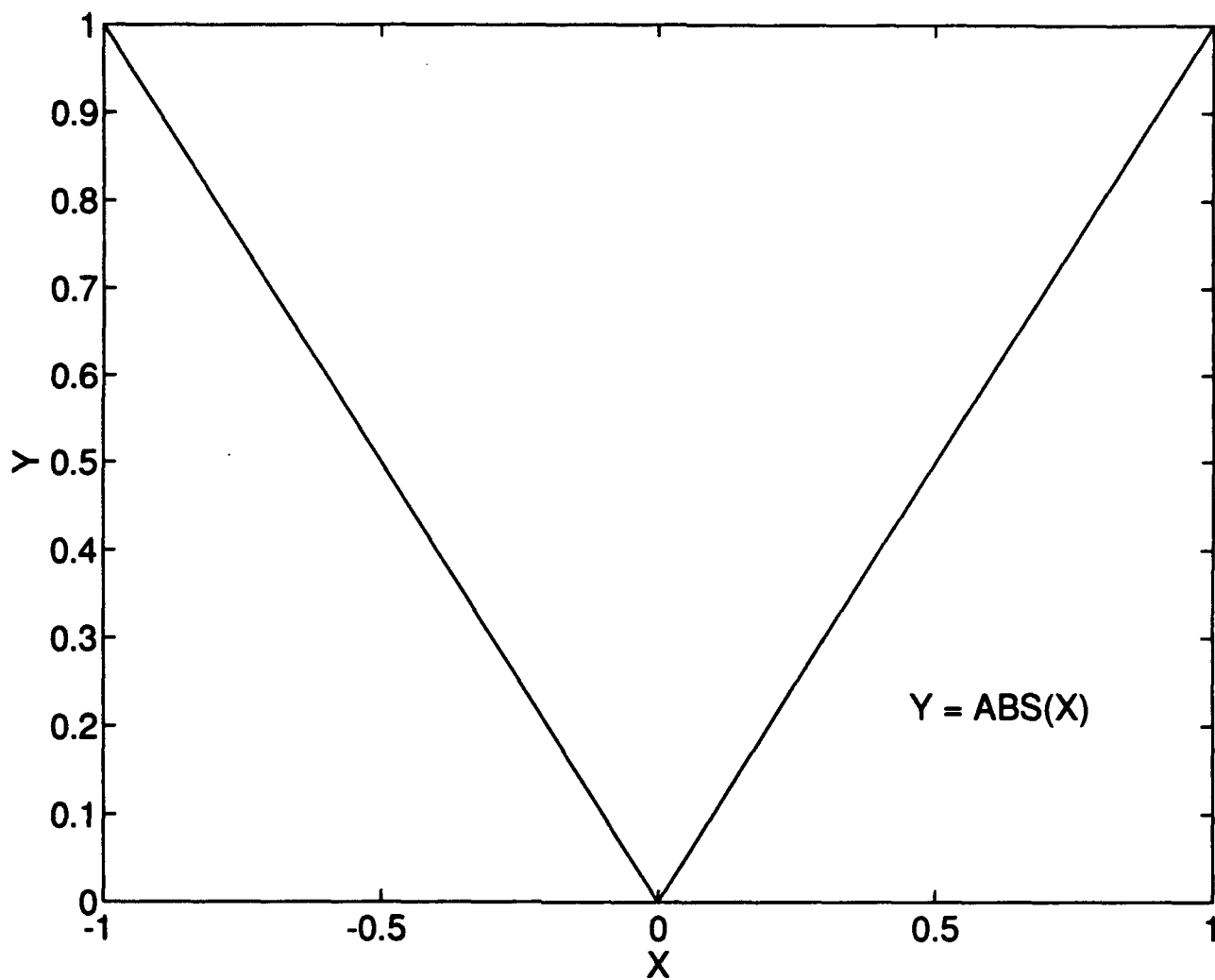
Figure 1. $x = t^3$, $y = |t^3|$. This curve is C^1, thus parametrically continuous but not G^1.

defined as the following:

$$K = \frac{\left(\frac{dx}{dt}\right)\left(\frac{d^2y}{dt^2}\right) - \left(\frac{dy}{dt}\right)\left(\frac{d^2x}{dt^2}\right)}{\left[\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2\right]^{\frac{3}{2}}} \ .$$

Therefore, for (a),

$$K_{t=1} = \frac{(6)(-6) - (6-6)(6)}{[(6)^2 + (6-6)^2]^{\frac{3}{2}}} = \frac{-36}{216} = \frac{-1}{6} \ ,$$

(b),

$$K_{t=1} = \frac{(12-6)(-6) - (6-6)(6)}{[(12-6)^2 + (6-6)^2]^{\frac{3}{2}}} = \frac{-36}{216} = \frac{-1}{6} \ ,$$

Both curves have curvature equal to -1/6. Figure 2 depicts a graph of these two

curves that coincide at point (3,3).

Figure 2. This curve consists of two piecewise parabolic arcs, that coincide at P2. This curve is not twice differentiable at P2, but it has continuous curvature, and is therefore G^2.
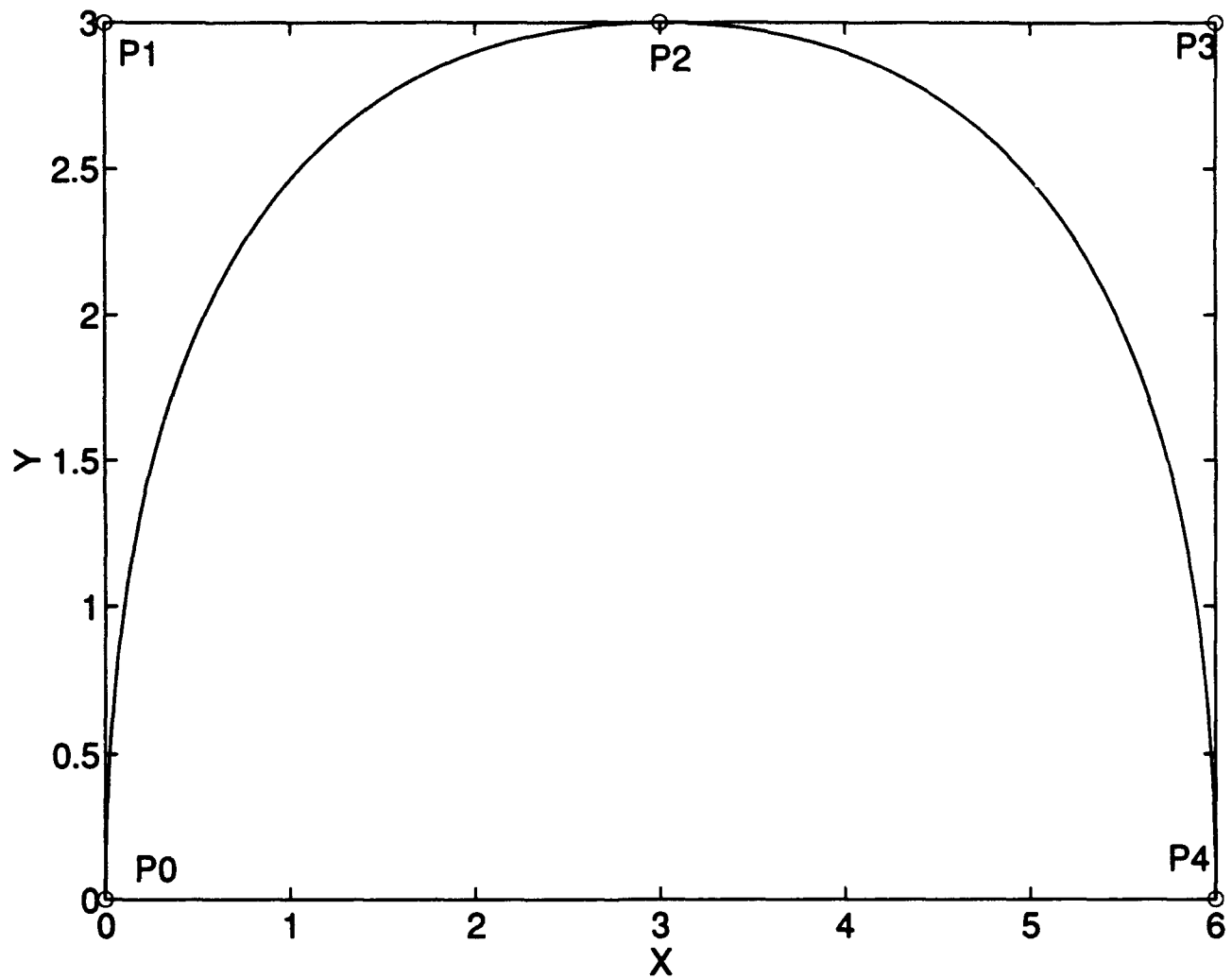
C.   BERNSTEIN POLYNOMIALS

Bernstein polynomials are a useful tool for matching a curve to a function f on a closed bounded interval [a,b]. The polynomials are so named because of their central role in Bernstein's constructive proof of the Weierstrass approximation theorem. The essence of this proof is the construction of a sequence of polynomials which can be shown to converge uniformly to f. It also turns out that the derivatives of these approximations converge uniformly to those of f - assuming the latter exists. Buchanan [Ref. 13].

We begin with the definition of the Bernstein polynomials.

Definition: The Bernstein polynomial of degree n associated with the function f on [a,b] is defined by

$$B_n(f;x) = \sum_{i=0}^{n} \binom{n}{i} (x-a)^i (b-x)^{n-i} f(x_i) \tag{1}$$

where the points $x_i = a + ih = a + (i/n)(b-a)$ for $i = 0,1,...,n$. In the special case where the interval is [0,1], equation (1) reduces to

$$B_n(f;x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} f(\frac{i}{n}) = \sum_{i=0}^{n} f(\frac{i}{n}) B_{n,i}(x) \tag{2}$$

where the Bernstein basis polynomials are defined by

$$B_{n,i}(x) = \binom{n}{i} x^i (1-x)^{n-i} \qquad (i = 0,1,...,n; n = 0,1,2,...) \ .$$

Those who have studied probability theory should recognize the Bernstein basis polynomials as the probability density functions for a binomial distribution. Specifically, $B_{n,i}(x)$ is the probability of achieving exactly i successes in a sequence of

n independent trials where the probability of success on any one trial is x. Bernstein worked in probability theory as well as in differential equations and approximation theory.

In view of this, it is clear that

$$\sum_{i=0}^{n} B_{n,i}(x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} = (x+(1-x))^n = 1 \qquad (3)$$

from which it can be deduced that the Bernstein polynomial for the function $f(x) = 1$ is itself identically 1; equation (3) is the "partition of unity property".

As another example, by differentiating (3) we obtain

$$0 = \sum_{i=0}^{n} \binom{n}{i} x^{i-1} (1-x)^{n-i-1} [i(1-x) - (n-i)x]$$
$$= \sum_{i=0}^{n} \binom{n}{i} x^{i-1} (1-x)^{n-i-1} (i-nx)$$

and multiplying by x(1-x) yields

$$\sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} (i-nx) = 0 \quad .$$

Therefore, it follows that

$$\sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} \frac{i}{n} = \sum_{i=0}^{n} \binom{n}{i} x^{i+1} (1-x)^{n-i} = x \qquad (4)$$

using (3). Thus, the Bernstein polynomial for the function $f(x)=x$ reproduces this function exactly.

Continuing in this way, differentiating (4) we have

and multiplying by x(1-x)/n yields

$$1 = \sum_{i=0}^{n} \frac{i}{n}\binom{n}{i} x^{i-1}(1-x)^{n-i-1}(i-nx)$$

$$\frac{x(1-x)}{n} = \sum_{i=0}^{n} \frac{i}{n}\binom{n}{i} x^{i}(1-x)^{n-i}\left(\frac{i}{n}-x\right)$$

$$= \sum_{i=0}^{n} \frac{i^2}{n^2}\binom{n}{i} x^{i}(1-x)^{n-1} - x^2$$

or

$$B_n(x^2;x) = x^2 + \frac{x(1-x)}{n}.$$

Since

$$\max_{0\le x\le 1} x(1-x) = \frac{1}{4}$$

it follows that

$$\|x^2 - B_n(x^2;x)\|_\infty \le \frac{1}{4n} \to 0 \text{ as } n \to \infty$$

The good approximation properties of Bernstein polynomials give us hope that $B_n(f;x)$ should be a good approximation to $f(x)$ over $[0,1]$. As it turned out however, Bernstein polynomials are good at mimicking the behavior of f, but slow in converging to f.

Bernstein basis polynomials are usually used to express Bézier curves. The polynomial is defined explicitly by

13

$$B_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i},$$

where t ϵ [0,1] and the binomial coefficients are given by

$$\binom{n}{i} = \begin{cases} \dfrac{n!}{i!(n-i)!} & if \quad 0 \le i \le n \\ \\ 0 & else \end{cases}$$

One important property of Bernstein polynomials is that they satisfy the recursion:

$$B_{n,i}(t) = (1-t)B_{n-1,i}(t) + tB_{n-1,i-1}(t)$$

with $B_{0,0}(t) \equiv 1$ and $B_{n,j}(t) \equiv 0$ for j ϵ {0,1,...,n}. The proof can be found in Farin [Ref. 6]. Also note that Bernstein polynomials are nonnegative over the interval [0,1]; this leads to the convex hull property. The <u>convex hull property</u> states that the curve is contained within the convex hull of the control polygon. The <u>convex hull</u> of a set of points is the smallest convex set containing the set of points. Bernstein polynomials are also symmetric with respect to t and (1-t), and thus $B_{n,j}(t)$ = $B_{n,n-j}(1-t)$. The Bernstein polynomial $B_{n,i}$ has only one maximum and attains it at t=i/n; given a function f(x), if the f(i/n) value is changed, then the Bernstein polynomial is mostly affected by this change in the region of the curve around the parameter value t = i/n, which relates to the local control property exhibited by B-spline curves.

D.   BÉZIER CURVES

Bézier curves and ⁓rfaces were independently developed by P. de Casteljau at Citroën and by P. Bézier at Renault Automobile Company. Bézier developed them in the early 1960's to fill a need for curves whose shape can be readily controlled by changing a few parameters and which exhibit a "local control" property. Bézier's application was to construct pleasing surfaces for car bodies. de Casteljau's development was never published, and so the whole theory of parametric polynomial curves and surfaces in Bernstein form now bears Bézier's name. Farin [Ref. 6].

Bézier curves are numerically the most stable of all polynomial bases currently used in Computer Aided Design (CAD) systems. Therefore, Bézier curves are the ideal geometric standard for representation of piecewise polynomial curves. Bézier curves also lend themselves easily to a geometric understanding of many CAGD phenomena and may be used to derive the theory of rational and non-rational B-spline curves.

Curve fitting has received a fair amount of attention, even before computer graphics came along. Piecewise polynomial curves have been used to mathematically interpolate discrete data sets. The algorithm developed fits a piecewise cubic Bézier curve to a set of data points. In particular it fits geometrically continuous (G^1) curves, rather than parametrically continuous (C^1) curves.

The control points or Bézier points are coordinate pairs (x_i, y_i) for $i=0,1,...N$

15

which are used to define a Bézier curve by setting

$$x(t) = \sum_{i=0}^{N} x_i B_{N,i}(t) \qquad y(t) = \sum_{i=0}^{N} y_i B_{N,i}(t) \quad .$$

The polygon formed by (x_0,y_0), $(x_1, y_1),\ldots(x_n,y_n)$ is called the <u>Bézier polygon</u> or <u>control polygon</u>.

Suppose we are given a set of data points, p_i, and the points do not necessarily progress from left to right. The coordinates of each point are treated as a two vector components.

$$p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad .$$

In parametric form, the curve is

$$P(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} , \quad 0 \le t \le 1 \quad .$$

The n-th degree Bézier polynomial determined by $n+1$ points is given by

$$P(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-1} t^{i} p_i \quad .$$

For $n=2$, this would give the quadratic curve defined by three points p_0, p_1, p_2:

$$P(t) = (1)(1-t)^2 p_0 + 2(1-t)(t)p_1 + (1)t^2 p_2 \quad .$$

The parametric form of the above curve is

$$x(t) = (1-t)^2 x_0 + 2(1-t)(t)x_1 + t^2 x_2 \quad ,$$

$$y(t) = (1-t)^2 y_0 + 2(1-t)(t)y_1 + t^2 y_2 \quad .$$

Note that if $t=0$, $x(0)=x_0$ and $y(0)=y_0$. If $t=1$, the point is (x_2,y_2). As t takes on values between 0 and 1, a curve is traced that goes from the first point to the

16

third point of the set. Usually, the curve will not pass through the central point of the three unless the points are collinear; if the points are collinear, the curve is a straight line through all three points. In effect, the points of the second-degree Bézier curve have coordinates that are weighted sums of the coordinates of the three points that are used to define it.

For n=3, we get the cubic Bézier polynomial, which we will use in our algorithm. The basic properties of higher degree Bézier polynomials are the same as for the cubic. The parametric form for the Bézier cubic is the following:

$$x(t) = (1-t)^3 x_0 + 3(1-t)^2(t)x_1 + 3(1-t)(t^2)x_2 + t^3 x_3 \ ,$$

$$y(t) = (1-t)^3 y_0 + 3(1-t)^2(t)y_1 + 3(1-t)(t^2)y_2 + t^3 y_3 \ .$$

Note again that $(x(0), y(0)) = p_0$ and $(x(1),y(1)) = p_3$, and that the curve will not ordinarily go through the intermediate points. As illustrated in Figure 3(a) and 3(b), changing the intermediate "control" points changes the shape of the curve. Figure 3(c) is a sixth degree Bézier curve, notice that the curve tends to follow along the path of the points (this will be further explained later).

Sometimes it is convenient to represent the Bézier curve in matrix form. For Bézier cubics, this is

$$P(t) = \frac{1}{6} [t^3, t^2, t, 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \ .$$
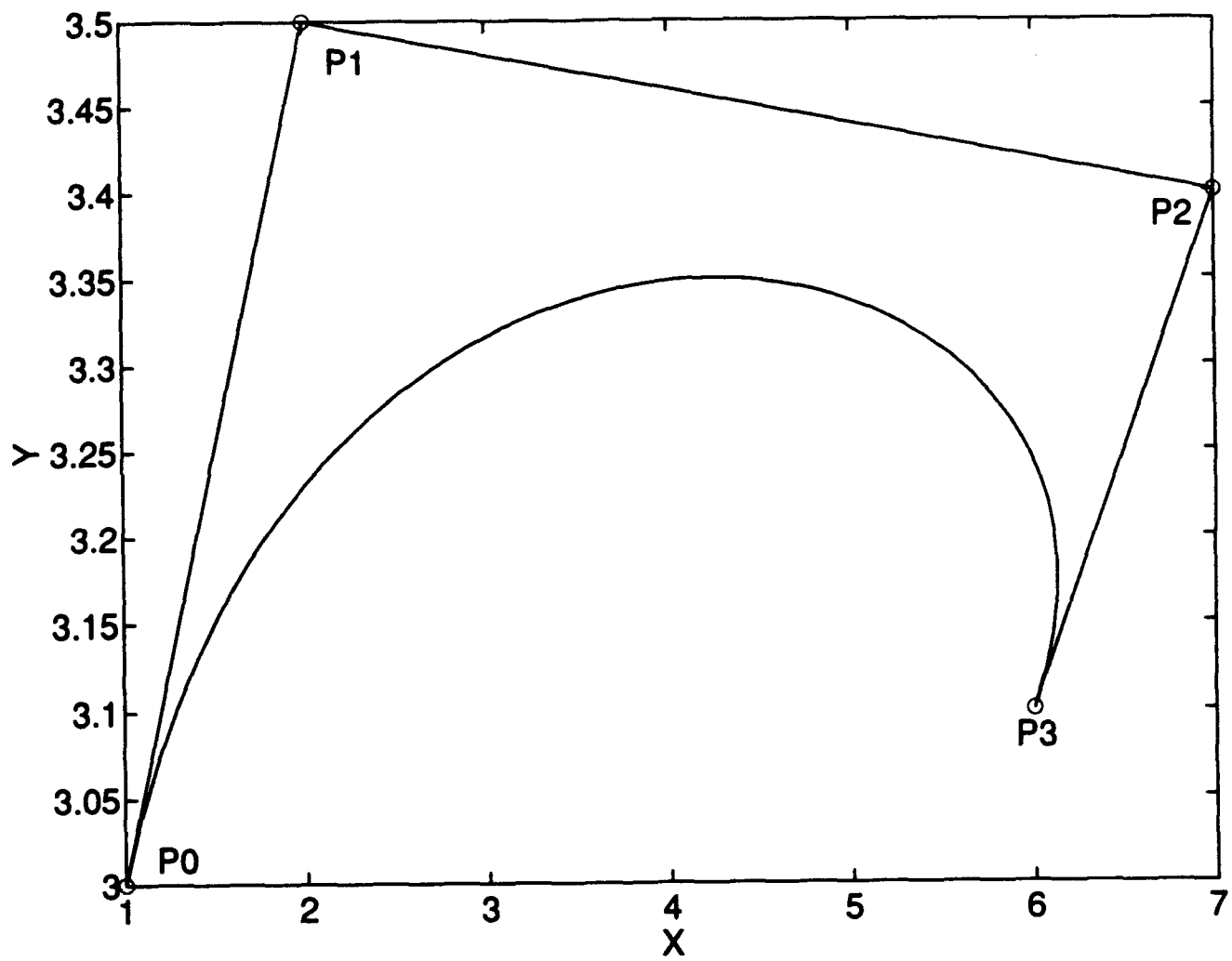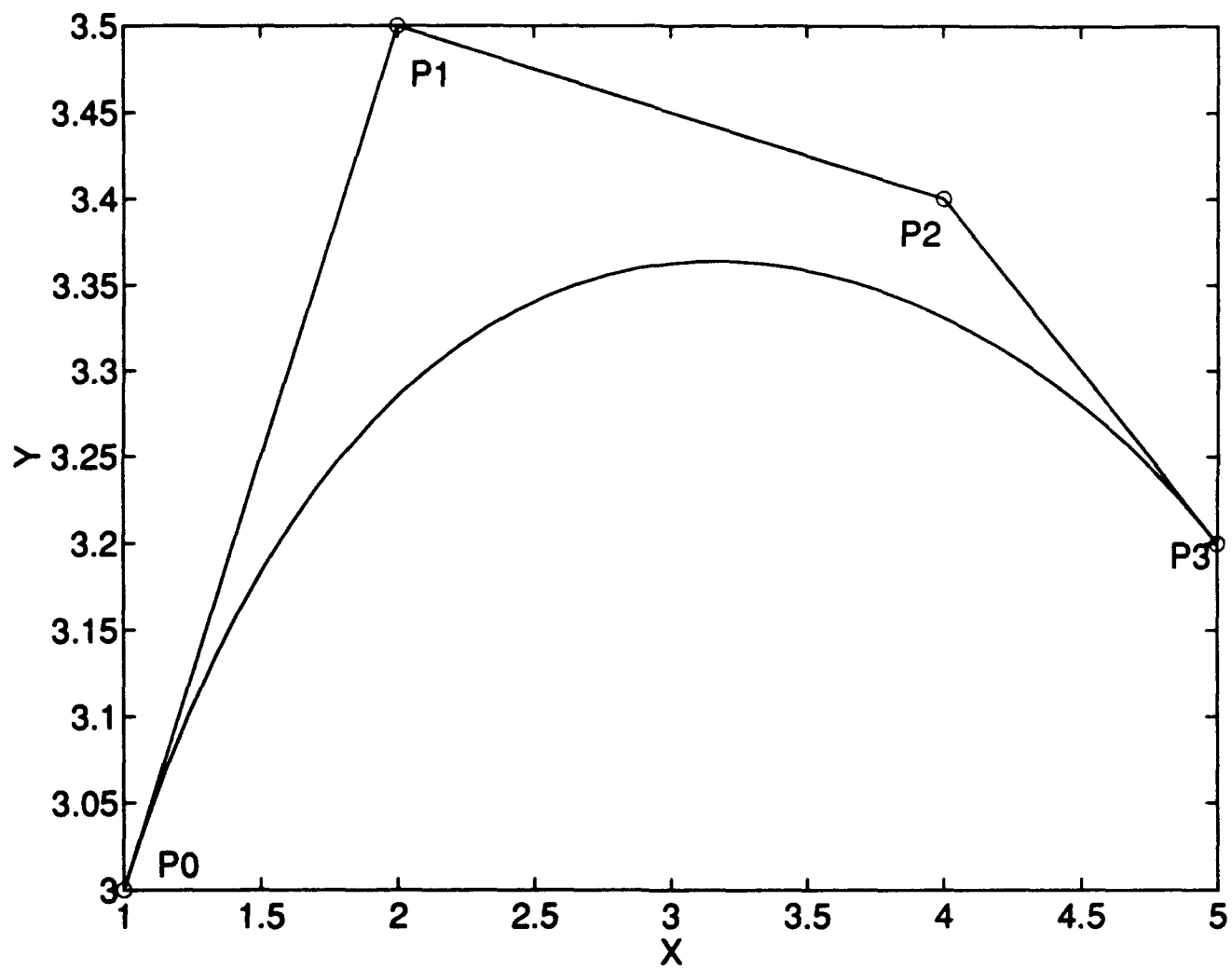
Some properties of Bézier cubics are:
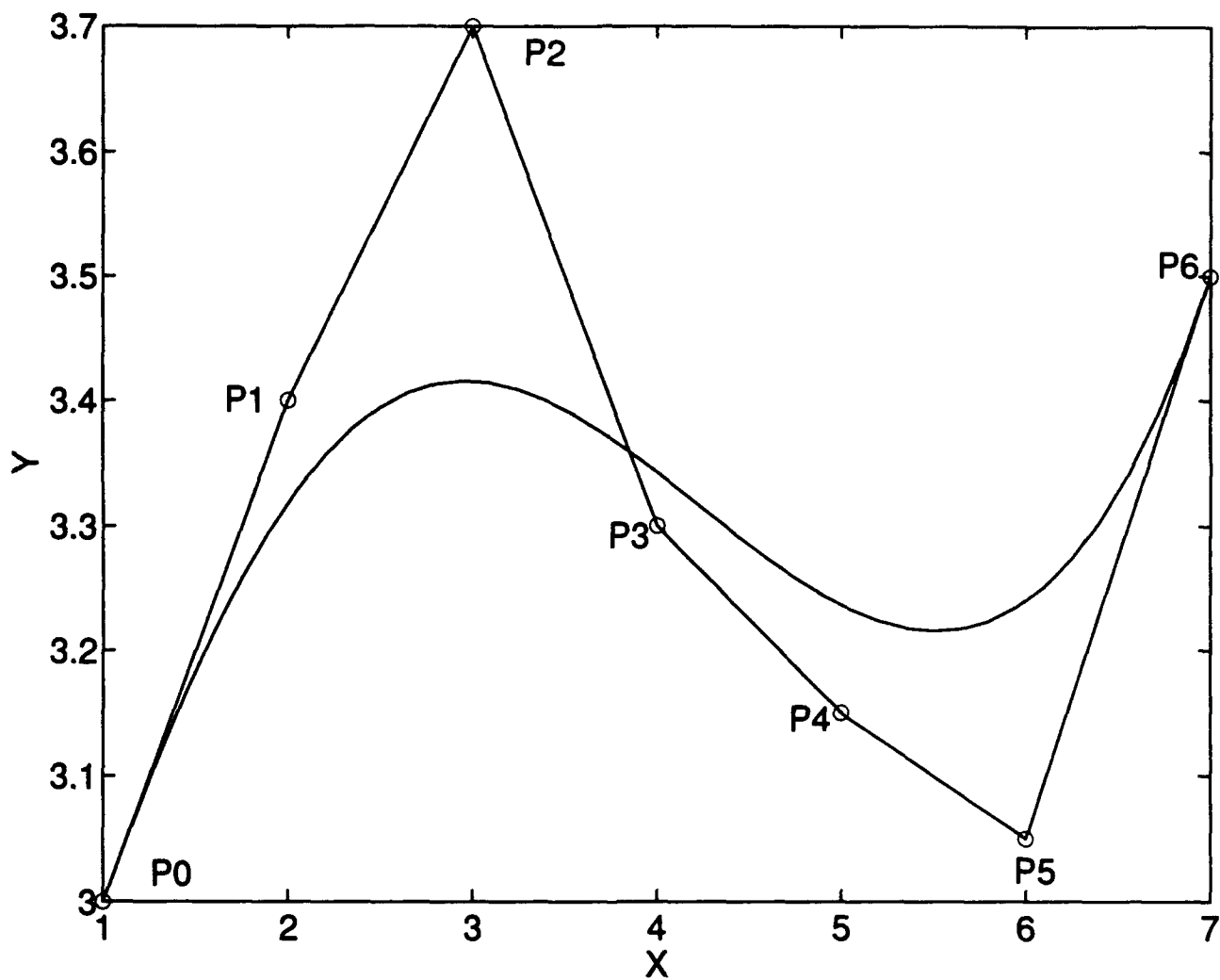
Figure 3(a).

Figure 3(b).

19

Figure 3(c). Sixth degree Bézier curve.

(i) Since

$$\frac{dx}{dt} = 3(x_1 - x_o) \quad and \quad \frac{dy}{dt} = 3(y_1 - y_0) \quad at \quad t=0 \quad ,$$

the slope of the curve at t=0 is

$$\frac{dy}{dx} = \frac{(y_1 - y_0)}{(x_1 - x_0)}$$

which is the slope of the secant line between p_0 and p_1. Similarly, the slope at t=1 is the same as the secant line between the last two points.

(ii) The Bézier curve is contained in the convex hull determined by the four points.

If a curve has a complex shape, then its Bézier representation will have a very high degree (for practical purposes, degrees exceeding 10 are prohibitive). However, such complex curves can be modeled using piecewise or composite Bézier curves.

Bézier cubic curves can be continued beyond the first set of four points by subdividing seven points (p_0 to p_6) into two groups of four points, with the central point p_3 belonging to both sets. Because the curve is tangent to the control polygon at the end control points, the piecewise curve has continuous slope provided the points p_2, p_3, and p_4 are collinear. See Figure 4 for an example of this.

A piecewise Bézier curve s is the continuous map of a collection of intervals $u_0 < \ldots < u_L$ into $\mathbb{R}^n$, where each interval $[u_i, u_{i+1}]$ is mapped onto a polynomial curve segment. Each real number u_i is called a breakpoint or knot. The collection of all u_i is called the knot sequence. For every parameter value u, there is a corresponding
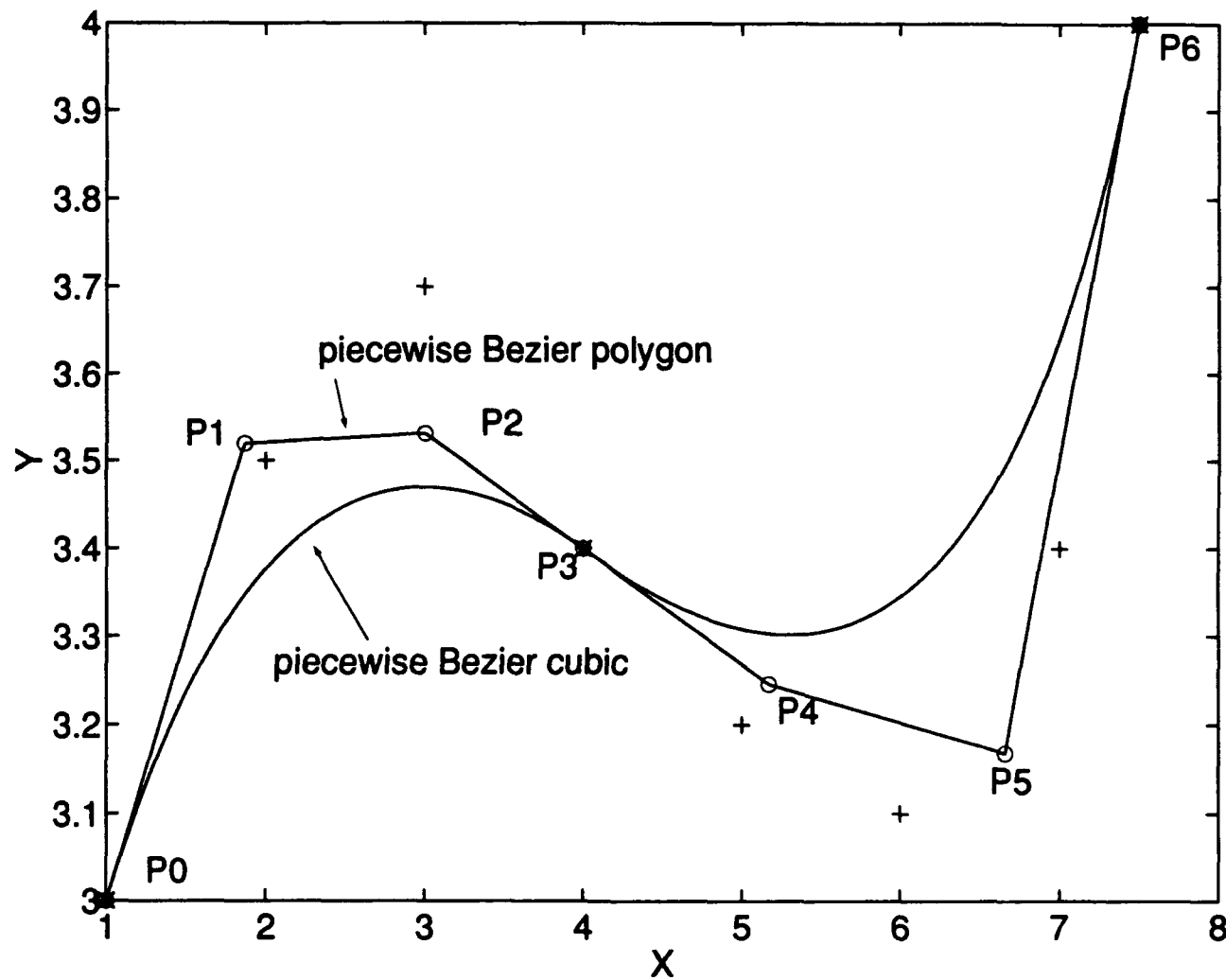
21

Figure 4. O → control points, ⊗ → knot points, + → data points. The curve is G^1 continuous at P3 because of tangency at the end control points.

point s(u) on the curve s. Let value v be from an interval $[u_i, u_{i+1}]$. We can introduce a local coordinate (or local parameter) t for the interval $[u_i, u_{i+1}]$ by setting

$$t = \frac{v - u_i}{u_{i+1} - u_i} \quad .$$

One can check that t varies from 0 to 1 as v varies from u_i to u_{i+1}. The collection of the Bézier polygons for all curve segments itself forms a polygon; it is called the piecewise Bézier polygon of s. Farin [Ref. 6].

It is the geometry of Bézier curves that makes them a good choice for our purposes. Sometimes, the shape of the curve may require a cusp or corner; this can be achieved by having knot points coincide, which produced zero tangent vectors. Piecewise Bézier curves have the geometric property that by changing one of the points we only change one portion of the curve, a "local" effect. Generally, cubic spline curves have a "global" effect, in that the curve from the first to the last point would be affected. Bézier curves are not really interpolating splines, since the curves do not normally pass through all of the points. In this respect they show some similarity to least-squares curves and thus are effective in approximating data. Other properties of Bézier curves are discussed by Farin [Ref. 6] and Gerald [Ref. 14: pp. 217-222].

II. THE PROBLEM: FITTING ORDERED DATA IN A PLANE

A.   USING BÉZIER CURVES AS A BASIS

It is interesting to note how piecewise cubic polynomials relate to a more common specification, Bézier curves. In this representation, the variables are vector values called control points which are multiplied by real-valued functions of t to fix the shape of the curve. The control points are therefore coordinate pairs (x_i, y_i) for $i=0,1,...,N$ which are used to define a parametric curve - a Bézier curve - by setting

$$x(t) = \sum_{i=0}^{N} x_i B_{N,i}(t)$$
$$y(t) = \sum_{i=0}^{N} y_i B_{N,i}(t) \quad .$$

Such a specification can be converted to a scheme of vector-valued basis functions and real-valued coefficients as follows: let the control points be specified as

$$\vec{a}_j = a_{1j} \vec{b}_{1j} + a_{2j} \vec{b}_{2j}$$

each control point having its own coordinate system with basis

$$\{\vec{b}_{1j}, \vec{b}_{2j}\} \quad .$$

Furthermore, let the original basis functions be $\{\phi_j(t)\}$. Then the parametric curve is

$$\sum_j \vec{a}_j \phi_j(t) = \sum_j (a_{1j} \vec{b}_{1j} + a_{2j} \vec{b}_{2j}) \phi_j(t)$$
$$= \sum_j (a_{1j}(\vec{b}_{1j} \phi_j(t)) + a_{2j}(\vec{b}_{2j} \phi_j(t))$$

24

so the appropriate set of vector valued basis functions is $\{\vec{b}_{ij}\phi_j(t)\,|\,i=1,2\}$. Plass and Stone [Ref. 15].

This means we can preset linear constraints on the control points for the resulting curve. Since each control point has been given its own coordinate system, by appropriate choice of the coordinate systems and the free coefficients, each control point can be constrained to be on a particular point or line, or to be free to move anywhere in the plane.

It is the control points as seen in Figure 1(a) and 1(b), that control the behavior of the Bézier curve, and in some sense make the curve "mimic" the Bézier polygon. This is the reason why Bézier curves provide such a handy tool for the design of curves: In order to reproduce the shape of a hand drawn curve, it is sufficient to specify a control polygon that "exaggerates" the shape of the curve. Letting the computer draw the Bézier curve defined by the polygon, and if necessary, adjusting the location and possibly the number of the polygon vertices, one will reproduce a given curve after two to three iterations, Farin [Ref. 6].

B. USING PIECEWISE PARAMETRIC CUBICS WITH G¹ CONTINUITY

Our design methodology is to represent all shapes analytically using piecewise parametric cubic polynomials, from an ordered set of discrete data points. A parametric cubic polynomial is quite flexible, but has some limits. For instance, it can contain at most one loop or, if it has no loop, at most two inflection points. Therefore, the least-squares fit is very vulnerable to a poorly chosen continuity constraint such as a bad tangent vector.

Following Plass and Stone [Ref. 15], the common specification of a parametric cubic polynomial is $F(X(t), Y(t))$, where X and Y are cubic polynomials and t lies in the range 0 to 1. Using continuous, piecewise functions will enable us to constrain the endpoints and tangents for each cubic piece. To obtain G¹ continuity the knot point must lie between two adjacent control points on the same line, as in Figure 5; p_1 is a knot point. A cusp will occur if the two control points both precede or follow the knot point along the same line. A cusp is also defined as a point on a curve where the first derivative vector vanishes. Thus zero tangent vectors may give rise to corners or cusps in curves, occurring when two or all three points coincide. A corner is a point on a curve where the tangent, not necessarily the tangent vector, changes in a discontinuous way.
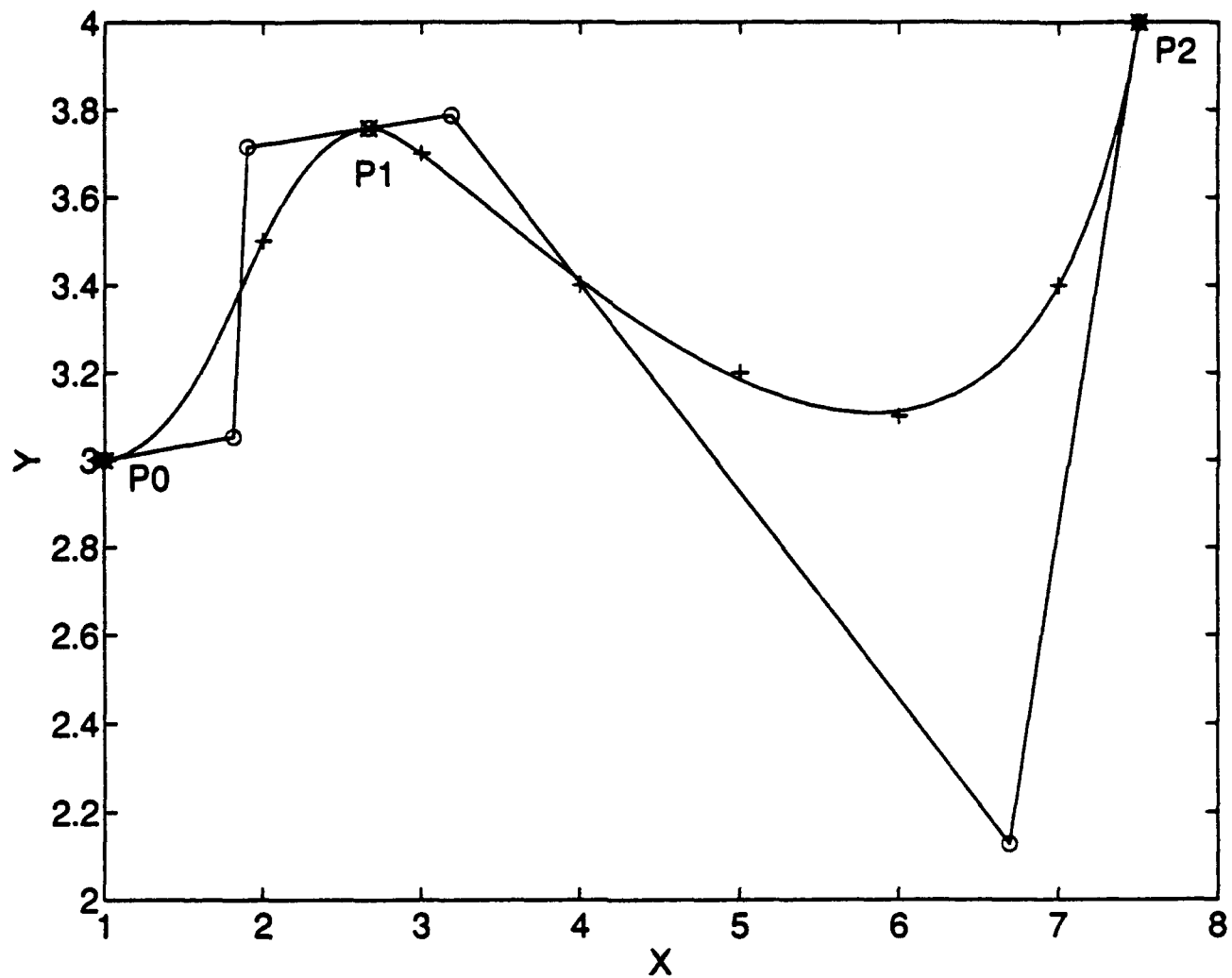
Figure 5. $\bigcirc \rightarrow$ control points, $\otimes \rightarrow$ knot points, $+ \rightarrow$ data points. Shows curve with G^1 continuity at the second knot point.

C.   ALLOWING FREE PARAMETERS

We have seen the parametric form of a Bézier curve. Bézier curves were developed to fill a need for curves whose shape can be readily controlled by changing a few parameters. By Figure 6, one can easily determine the parameters of the curve. The parameters are the locations of the knot points, the direction or angle of the unit tangent vectors at the knot points, and the distance from the knot points to adjacent control points. Note that the unit tangent vector is parallel to the three collinear control points. These three sets of parameters determine the shape of the approximating curve.

The number and placement of the knot points are critical in getting a good representation of the data, since the approximating curve must pass through each knot point. Choosing the knot locations can be and has been done in several ways. The technique described by Reeves [Ref. 16] in fitting cubic pieces to a set of data points is stated thusly; "one simple method for defining the knots is to "grow" the pieces out until the fit for that piece exceeds some threshold. In other words, starting with the previous knot, keep adding data points until the piece is as long as possible. Each new piece must be constrained to maintain continuity." This will work in the sense that the resulting curve will fit everywhere within some specified tolerance, but is vulnerable to local phenomena such as a badly defined tangent. Another choice, also described by Reeves, is subdivision.

Plass and Stone [Ref. 15] used "dynamic programming" to search a subset of

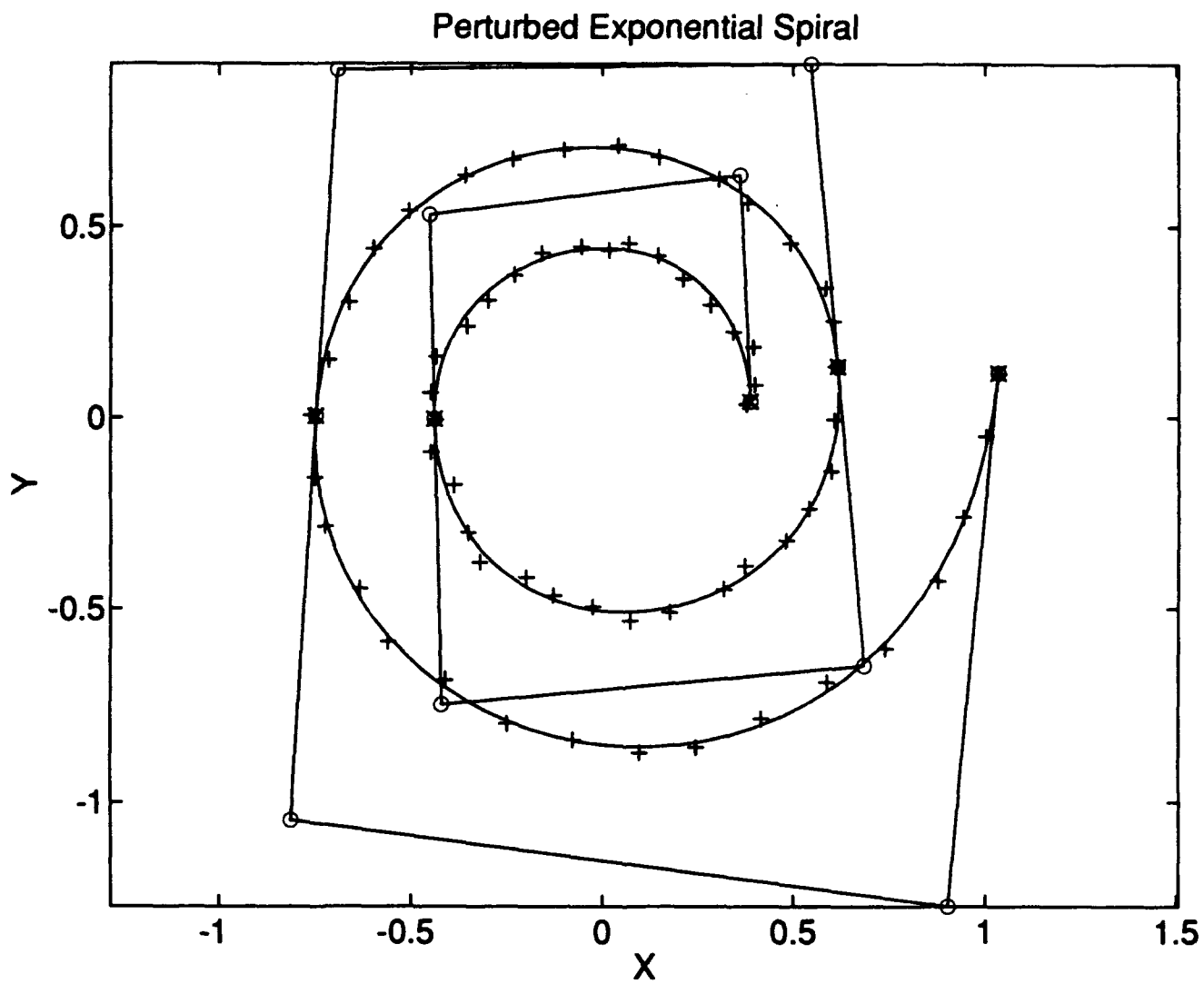**Figure 6.** $\bigcirc \rightarrow$ control points, $\otimes \rightarrow$ knot points, $+ \rightarrow$ data points. Parameters: Location of the knots, the direction or angle of the unit tangent vectors, and the distance from the knot points to adjacent control points.

the sample points as potential knot positions. The criterion of Powell [Ref. 17: pp. 65-83] called the "trend" for data fitting, is that knots are inserted successively until a satisfactory fit is obtained. An advantage of this criterion is that it is applicable even if the magnitude of error is unknown. This method was also used by Ichida and Kiyono [Ref. 18: pp. 164-174] in their one-pass method for curve fitting.

The unit tangent vectors are calculated at each knot point. The direction or angle of the unit tangent vectors determined the direction in which the approximating curve will travel. They are also used to calculate the positions of the interior control points for each cubic segment. These control points lie along the same line as the knot point and their location determines the geometric continuity of the curve at that knot point; this line is parallel to the unit tangent vector.

So, the collinearity of three successive control points does guarantee a continuously varying tangent. However, it is important to note that collinearity of three distinct control points is not sufficient to guarantee C^1 continuity as explained by Farin; this is because the notion of C^1 is based on the interplay between domain and range configurations. Collinearity of three points is purely a range phenomenon. Without additional information on the domain of the curve under consideration, we cannot make any statements concerning differentiability. However, C^1 continuity can exist under certain conditions, see Farin [Ref. 6: p. 93].

The distances from the knot point to adjacent control points determines the continuity and shape of the curve at that knot point. If these distances were equal, the curve would have G^2 continuity. As seen in Figure 6, these distances are usually not

equal, thus the curve is G^1. If these distances become zero, or the control points coincide with the knot point, a corner will be formed. This will be shown in a later example. If both control points precede or follow the knot point a cusp will occur. These parameters are also available to the optimization function. Henceforth, the control points, knot points, and data points will be represented as in Figure 6.

D. COMPUTING NORMAL DISTANCE FROM DATA POINTS TO CURVE

The basic problem can be stated thusly: given a parametric curve F(t) and a point Q, both in the plane, find the point on the curve closest to Q. In other words, find the parameter value t such that the distance from Q to F(t) is a minimum. Note that the line segment (whose length we wish to minimize) from Q to F(t) is perpendicular to the tangent vector of the curve at F(t), unless the closest point is an endpoint of the curve segment. The equation we wish to solve for t is

$$[F(t)-Q] \cdot F'(t) = 0 \qquad (1)$$

Curve F(t) is a cubic Bézier polynomial in our case;

$$F(t) = \sum_{i=0}^{n} V_i B_{n,i}(t), \quad t \in [0,1], \quad where \quad V_i$$

are the control points. Curve F'(t) can also be expressed in Bézier form;

$$F'(t) = n \sum_{i=0}^{n-1} (V_{i+1} - V_i) B_{n-1,i}(t) \ .$$

We are dealing with cubics, so polynomial F(t) is of degree three, and F'(t) is of degree two. Polynomial F(t) - Q is also of degree three, so the polynomial

31

described by equation (1) is generally of degree five. Thus the problem can be restated as one of finding the roots of this fifth-degree polynomial. There is no closed-form solution to this problem. A technique developed by Schneider [Ref. 19: pp. 607-611], converts the equation to Bézier form, and then uses a recursive algorithm to find the roots. The roots are evaluated to find the points on the curve. By comparing the distances from those points on the curve to the arbitrary point, and also considering the endpoints of the curve, the desired result is found - the point on the curve closest to the arbitrary point, and also its parameter value.

Plass and Stone use Newton-Raphson iteration to find the root. The Newton-Raphson iteration for solving f(t) = 0 is

$$t - t - \frac{f(t)}{f'(t)} \quad .$$

Referring back to equation (1), each iteration will reduce t by

$$\frac{[F(t) - Q] \cdot F'(t) \cdot t^2}{[(F(t) - Q) \cdot F'(t)]'} \quad .$$

Because Newton-Raphson iteration converges quickly and is fairly inexpensive, only a few steps are needed to find a close approximation to the root. This method has proven to be very stable, however their algorithm is not suitable for fitting smooth, continuous, piecewise functions because the endpoints and tangents for each cubic piece cannot easily be constrained.

With the data parameterized by normalized accumulated chord length, Marin and Smith [Ref. 20] used ODR (Orthogonal Distance Regression) splines to

32

approximate the data. Their fit is better as compared with using ordinary least squares, however under certain circumstances, there will not exist a solution, or the solution will not be unique.

III. IMPLEMENTATION

A. INITIAL GUESSES

The algorithm consists of several MATLAB programs including a main driver, "lsg2". We first start with a set of ordered data points, input by the user. The user can either select the knot spacing or let the program do it. Regardless, the initial knot locations are a subset of the data points. The program's knot sequence is based upon the size of the data and the number of knot points. The ultimate goal of knot placement is, of course, to use as few knots as possible to get a "good" representation of the shape of the data.

Next, the unit tangent vector was estimated for each control point. This was accomplished by using a chord length parametrization to fit a parametric quadratic curve to five data points. The five data points consisted of a center knot point and its two adjacent data points; at the end points the first five, or last five, points were used. The unit tangent vectors were approximated by the unit tangent vectors for those parametric quadratic curves, and indicated the direction of the adjacent control points. The angles of the unit tangent vectors are also computed as they are the parameter used in the optimization function.

The distance from the knot points to adjacent control points were calculated by first taking one-third of the distance between successive knots. The unit tangent vector multiplied by that distance was added or subtracted from each knot point to obtain the locations of adjacent control points.

34

The above process can be found in the function "iguess". The data points, knot points, and control points are then plotted; the control points are used to calculate the points of the approximating cubic Bézier curve. Thus, an initial guess of the shape of the curve is obtained and plotted.

B.   OPTIMIZATION ROUTINE

The optimization routine used was "fmins". The purpose of "fmins" is to minimize a function of several variables. For example, x=fmins('fun',x0) returns a vector x which is a local minimizer of fun(x) near the starting vector x0; 'fun' is a string containing the name of the objective function to be minimized. In our algorithm, the x0 array consists of the knot points, the angles of the unit tangent vectors and the distance from the knot points to adjacent control points. These are the control parameters.

The objective function, named "objf2" calculates the function that will be minimized by the optimization routine "fmins". Along with the x-array which was the old x0 array, "objf2" also calls the function "ctpts" that computes the control points and "newk" that partitions the data points among the cubic segments.

Because the data is ordered, it is necessary for the closest points on the approximating curve to be ordered in the same way. Consequently, it is necessary to associate the data points with particular cubic segments to avoid computing distances to a closer, but incorrect, cubic segment, as might occur if the data makes a loop. The function "newk" determines which data points partition the data set into subsets

35

associated with the various cubic segments. These data points will be called dividing points. So, given an initial knot sequence, the search for the dividing data points for the interior knots is achieved by finding the smallest distance from that knot point to the surrounding data points. From that knot point, the data points tested are those up to but not including the previous and subsequent dividing points. For the first and last knots, the data points tested are those up to the next and previous dividing points respectively.

During the optimization routine, the knot points initially coincide with the data points, and the subscripts of those data points indicate the dividing points (which are represented by the global variable "dpkpc") for each segment. However, as the knot points move, the dividing points may change. With each new iteration, the (possibly new) dividing points must be obtained.

When the function "sod" is called, the sum of the squares of the distances from the data points to the nearest point on the cubic segment is computed. In addition, the square of the distance from the first and last data points to the first and last knot points, respectively, was included in this sum. This was done to ensure that the curve begins near the first knot point and ends near the last knot point.

IV. CONCLUSIONS

A. EXAMPLES

The following examples illustrate how well the optimization routine approximates the curve given the initial guess. The initial guess is presented first, followed by the final optimized curve.

A major limitation of the initial guess curve is the number and placement of the knot points. If the spacing of the knot points is too close together the initial guess curve will make a straight line where a small curve should appear. Given that the curve must pass through the knot points, if the data points or knot points are spaced far apart, certain sections of the curve would not come close to those data point.s However, that distance is minimized by the optimization routine, resulting in a good approximating final curve, as will be seen in the examples.

Example (1): This data is from a reacting chemical system, taken from Marin and Smith [Ref. 20]. The abscissa, C_{co}, of each data point is an input concentration of carbon monoxide in a catalytic system. The ordinate, R, is the steady state oxidation rate achieved by the system. The initial guess curve consists of one interior knot, and is not a "bad" approximating curve. The initial knot positions are represented by "k" for the initial curve, and the dividing points for the cubic segments are represented by "dpkpc", which in some sense is similar to new knot positions, for the final curve. Figures 7 and 8 shows the improvement of the final curve.

Example (2): This data is also taken from Marin and Smith, and is a sample data for parametric curve in R^2, with seven knots. The initial guess curve is not very
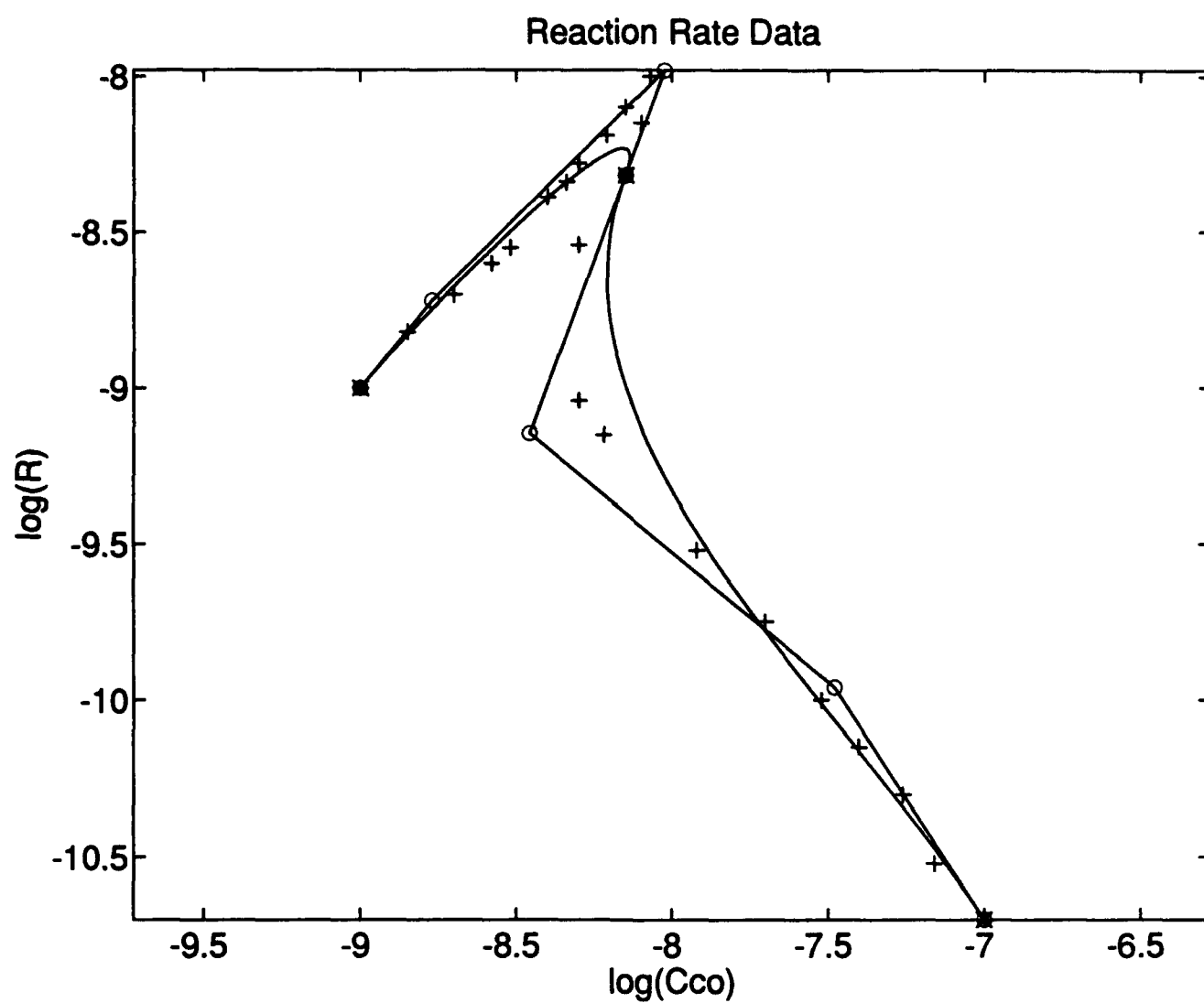
37

Figure 7. Initial guess, K=[1 13 23].

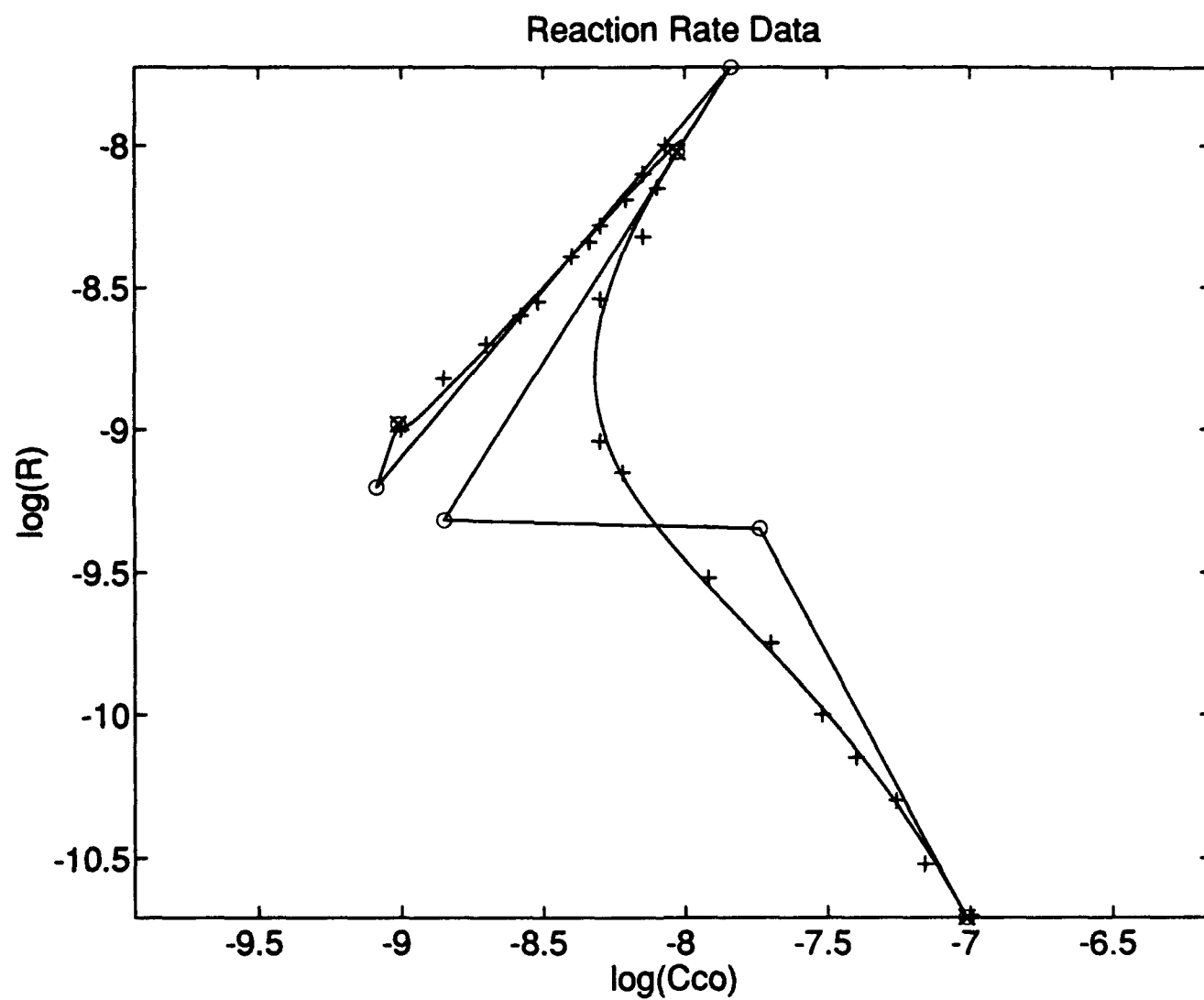**Figure 8.** Final curve, dpkpc = [1 11 23].

good. The curve loops over itself and thus misses data points by great distances, see Figure 9. But the power of the optimization function reduced this distance and removed the loop, producing an excellent approximating final curve, Figure 10.

Example (3): This data contained eighty-two data points with eight knots. The data represents the figure of the letter "H", Figures 11 and 12, again a much improved final curve.

Example (4): This data contained thirty-six data points for the curve $y = |x|$. The curve contains one interior knot point located at the vertex. The data forms a straight line, but the initial guess curve does not, because of the distances of the knot points, see Figure 13. However, in Figure 14, the control points that were located near $x = \pm 0.5$ converged to the knot point at the vertex, forming a needed corner. Thus, the final curve produced was an exact approximation, to within the convergence tolerance. However, note that the "curve" degenerates to two straight-line segments and that the location of one interior control point is arbitrary on the line segment. This non-uniqueness however, did not affect the curve's convergence. In some examples, such as (2) and (3), the maximum iterations allowed by "fmins" was reached, but if required, the continuation of iterations is incorporated in the main driver program "lsg2".

Figure 6, which contained sixty-four data points and five knots, was also a very good final curve for that data; the initial guess curve contained no loops but nevertheless, it was not good.
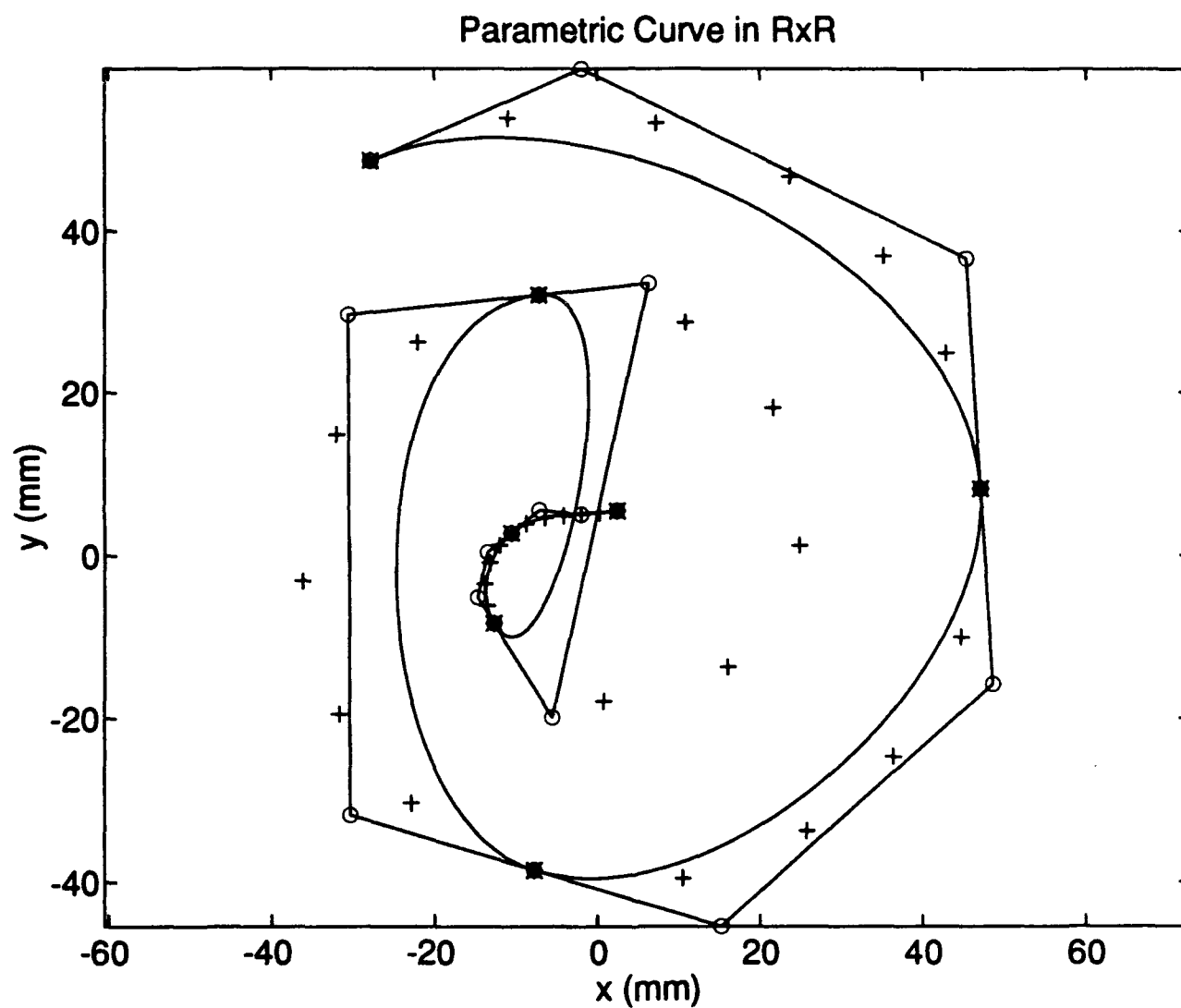
Figure 9. Initial guess, k=[1 7 12 18 24 29 35].

41

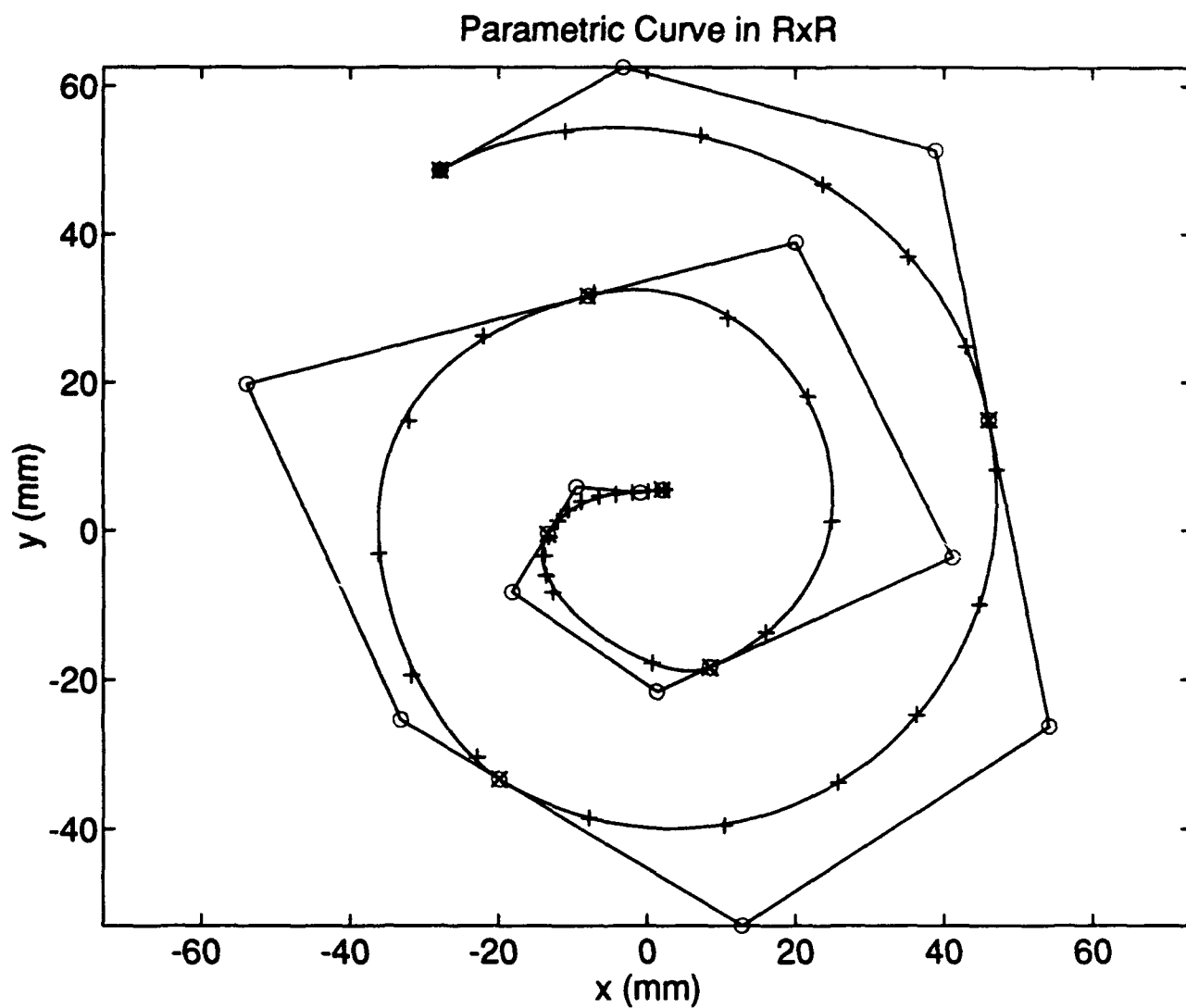**Figure 10.** Final curve, dpkpc=[1 7 13 18 23 29 35].
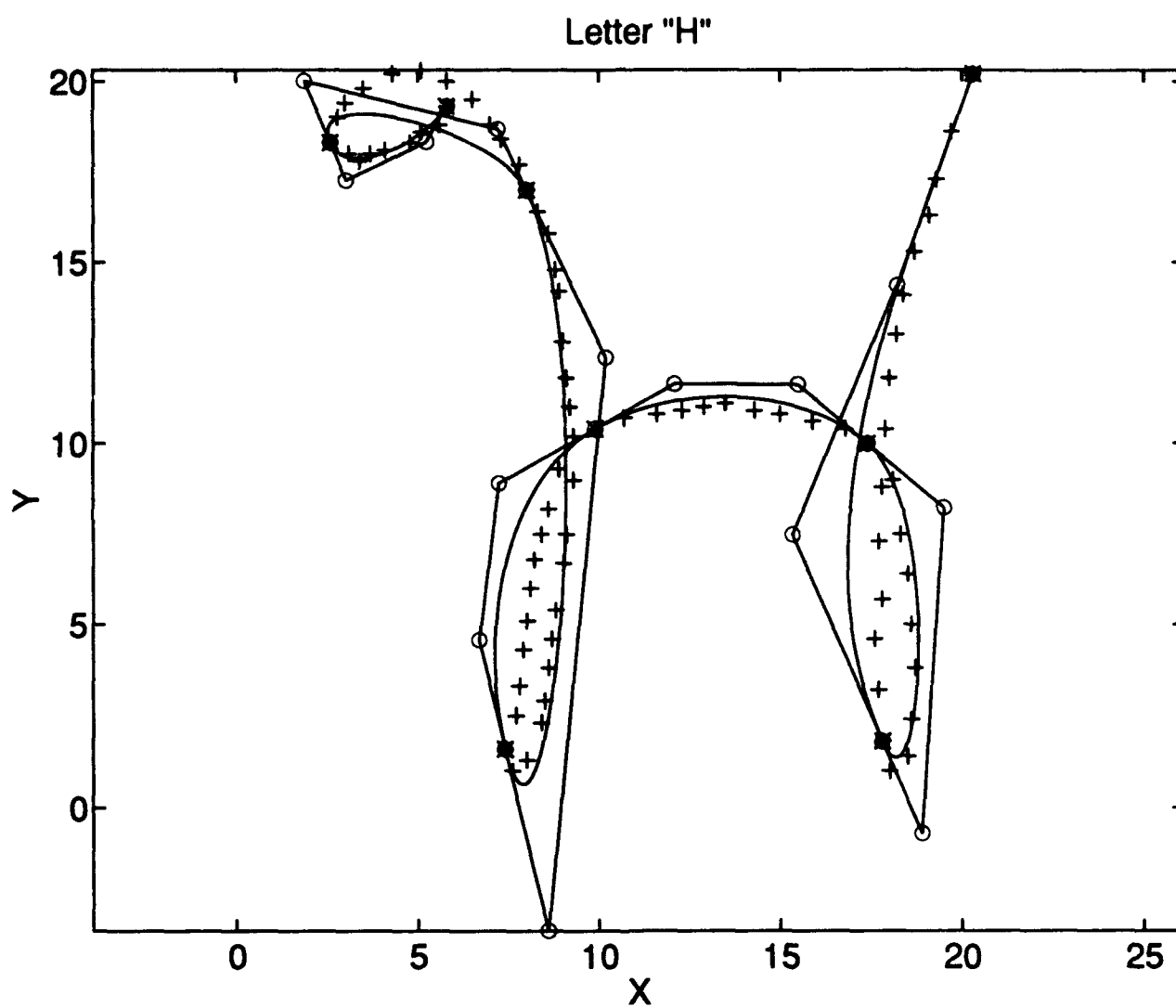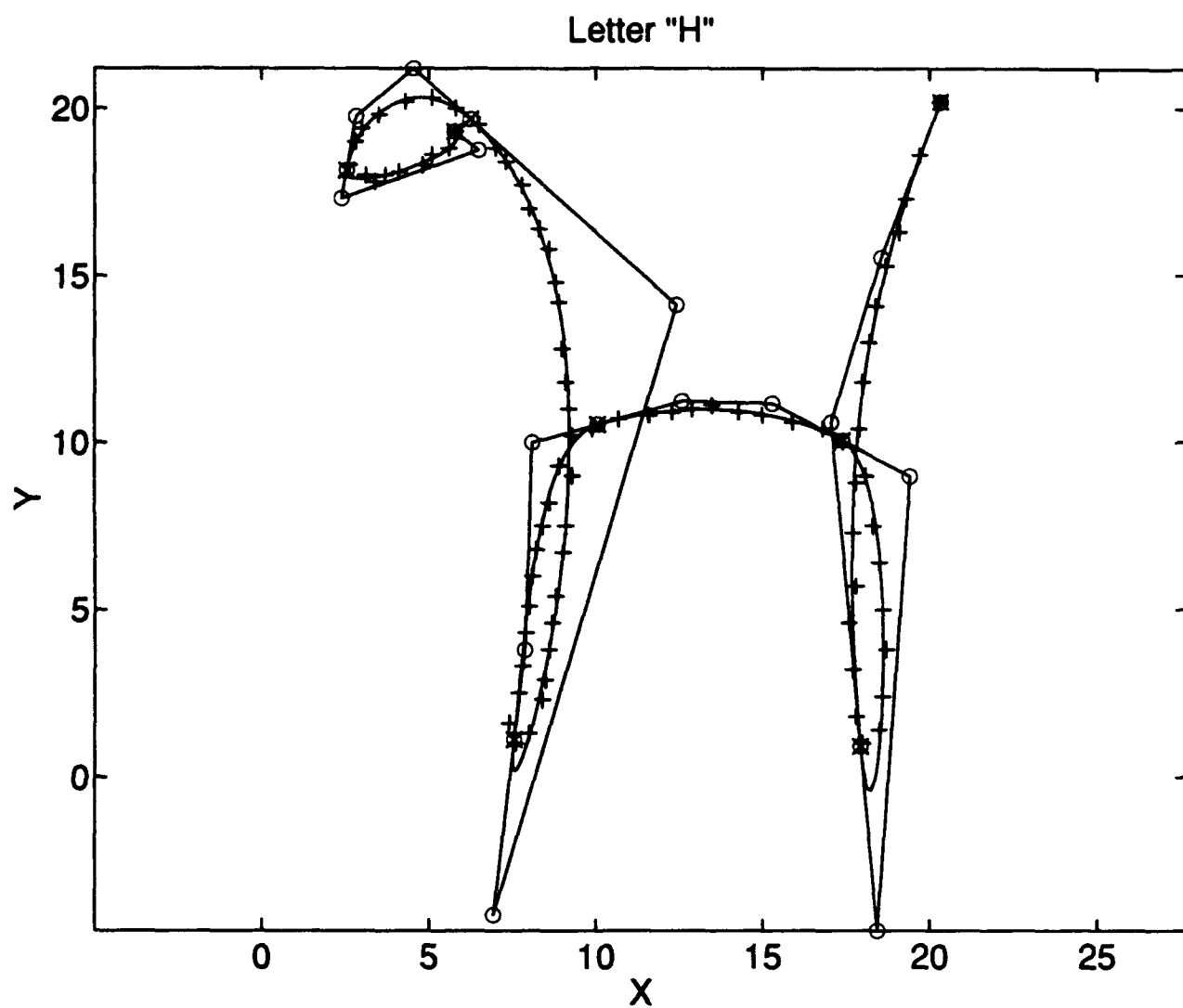
Figure 11. Initial guess, k=[1 9 20 38 49 59 68 82].
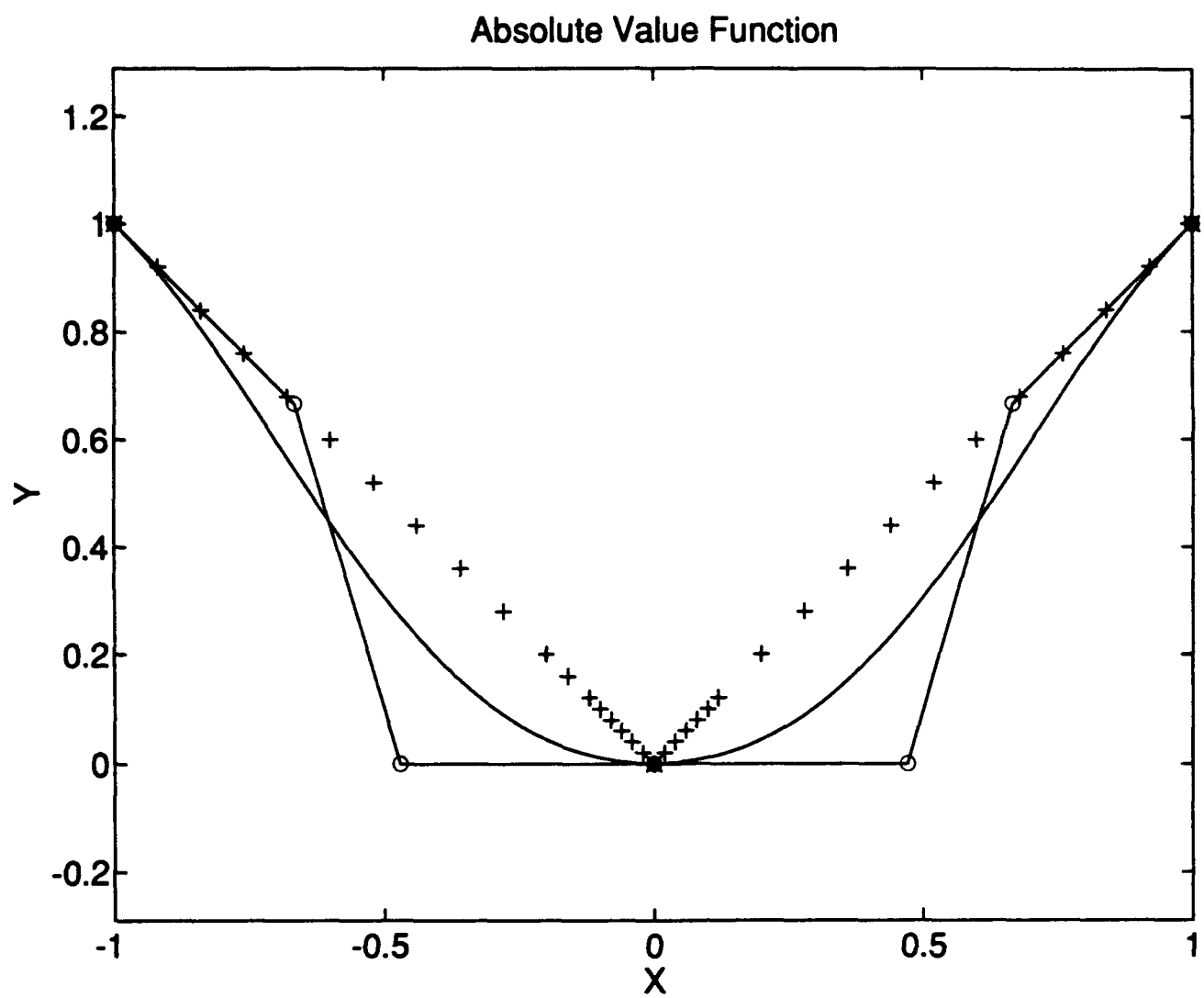
Figure 12. Final curve, dpkpc=[1 9 16 37 49 67 82].

44

Figure 13. Initial guess, k=[1 19 36].

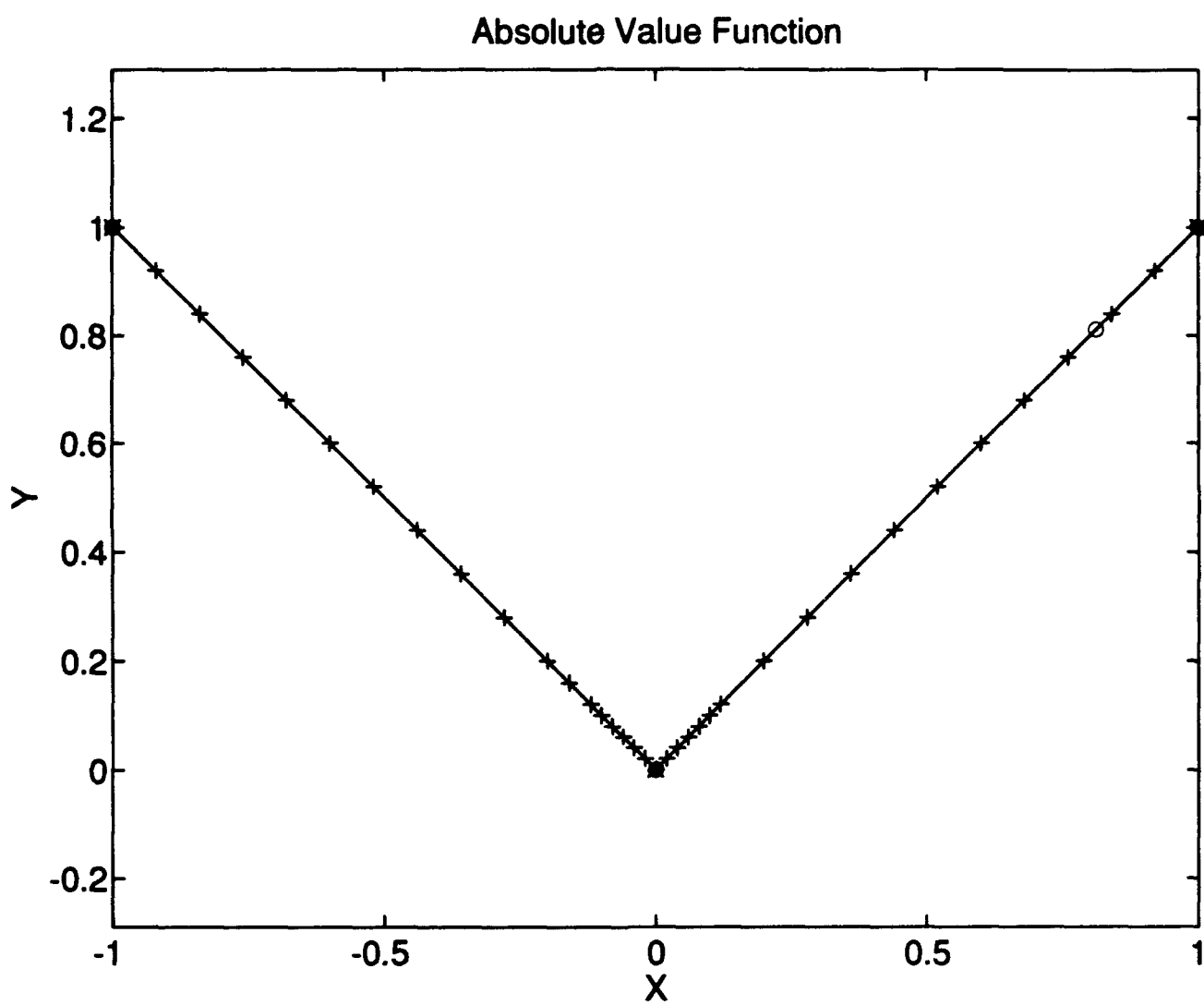**Figure 14.** Final curve, dpkpc=[1 19 36].

B. CONCLUSIONS

We have presented a method for approximating ordered data by using geometrically continuous piecewise parametric cubics. After choosing the knot points which were a subset of the data points, we used local quadratic approximations with chord length parametrization to estimate the unit tangent vectors. In turn, the unit tangent vectors are used to calculate the position of the interior control points for each cubic segment. Thus, an initial guess of the curve was approximated using the above parameters. The optimization routine "fmins" minimized the sum of the squares of the distances from the data points to the points on the approximating curve. By the examples, this optimization routine works well, however, in some examples such as (3), it was slow to converge which can become computationally expensive.

A better approach to this problem would be to first optimize on each cubic segment, which can be achieved by fixing the knots and the unit tangent vectors for that segment, and allowing the location of the control points on the necessary line to be free parameters. The "sub" optimization routine would minimize the sum of the squares of the distances between the data points associated with that cubic segment and the curve. Since the above process would be performed for each cubic segment, an improved initial approximating curve would result. Choosing a better initial guess should speed up convergence to the final approximation curve.

Another approach would be to gear the optimization process to this problem instead of using a general routine. In other words, an algorithm could be developed that exploits the good approximating and other properties of piecewise cubic

47

polynomials. Exactly how this can be done remains open, but the process is conceivable.

A large volume of work has already been done and is presently continuing in approximating curves to fit ordered data in two or three dimensions. The algorithms presented here, can be used as a basis for further work by geometric modelling researchers and graphics programmers in the field of computer aided geometric design.

APPENDIX: MAIN PROGRAMS

Program lsg2

```
% Program    lsg2
% This program is the main driver for Least Squares Approximation By G1
% Piecewise Parametric Cubics. It takes data points,Q; and uses the
% the function 'iguess' to get a plot of the initial guess curve. It also
% uses the array returned by the function 'iguess' for input to the
% optimization function 'FMINS'. Furthermore, it enables the user to continue
% with the iterations, if the maximum iterations are reached by the
% optimization routine.

 global dpkpc

 disp('Choose one of the following: dpts, dpts1, ..., dptsN')
   Q = input('  ');

 [r,m] = size(Q);

% Error checking.

 if r ~= 2
    disp('Set of points must be 2 x n matrix, try again.'), pause(2)
    lsg2
 end

% Obtains the x0 array for input to the optimization function 'FMINS'.

 x0 = iguess(Q);

 pause(5)

 options = [0,1.e-3,1.e-3];

 tb = clock;

% Calls the function 'FMINS'.

 x = fmins('objf2', x0, options, [],Q);

 et = etime(clock,tb);

% Enables the user to continue with the iterations, start anew or end.

 disp('Type "1" to continue iterations; "2" to start anew, "3" to end')
   b = input('  ');
```

```
if b == 1
   disp(' How many more iterations would you like? ')
   o = input('   ');
   options(14) = o;
   x0 = x;
   x = fmins('objf2', x0, options, [], Q) ;

elseif b == 2
   lsg2;

elseif b == 3 | b ~= 1 | b ~= 2 ;
    disp(' This program will now end! ')
end
```

function G = iguess(Q)

```
% function G = iguess(Q)
% This function takes data points, Q; and with a subset of the data points
% computes knot points,P; the position of the knot points,k; and the distance
% between the successive knot points,dt. It calls the functions 'unitv' to
% compute the unit tangent vectors for each knot point, 'ctpts' to compute
% the control points, and 'plotC' to plot the initial guess curve. The function
% returns the array that contains the initial guess curve's parameters. This
% array will be used by the main driver program 'lsg2' for input to the
% optimization function 'FMINS'.


global dpkpc

[r,m] = size(Q);

disp(' Give the number of knotpoints')
n = input('   ');

disp('Type "1" for default knot position or "2" to input your own.')
h = input(' ');

if h == 1

    k = round(((m-1)/(n-1))*[0:n-1] + ones(1,n));

    k;
elseif h == 2

    disp('Input the initial knot point position, example; [1 4 8 ...n]')
        k = input('   ');

elseif h ~= 1 | h ~= 2

    disp(' You can only choose "1" or "2". You must start over.'),pause(2)
        iguess
end

dpkpc = k;

% Gets the knotpoints

P = [P Q(:, k)];

% Computes the distance between successive points

[s,t] = size(P);
```

51

```
d =  P(:,1:t-1) - P(:,2:t);

d = (1/3) * sqrt(sum(d.*d));

dt = [d;d];

% Calls the function "unitv" to compute the unit tangent vectors.

u = unitv(Q,k);
ang = atan2( u(2,:) ,u(1,:) );

% Calls the function "ctpts" to compute the control points.

C = ctpts(P,ang,dt);

% Calls the  function "plotC" to plot the initial guess curve.

 plot = plotC(C,Q,P);

% Sets up the array that contains the initial guess curve's parameters
% that are available to the optimization routine.

G = [P(1,:)  P(2,:)  ang  dt(1,:)  dt(2,:)];
```

function os = objf2(x,Q)

```
% function os = objf2(x,Q)
% This is the objective function that will be minimized by the optimization
% function 'fmins'. The input arguments are the vector x that minimizes
% function objf2(x), and the data points Q. The output is the sum from
% the function 'sod' plus the distance squared from the first and last data
% points to the first and last knot points.

% Begin function
  global dpkpc
  m = length(x);
  n = round(m/5);
  [r,s] = size(Q);


% Picks out the knot points.
  P(1,:) = x(1:n);
  P(2,:) = x(n+1:2*n);

% Picks out the angles of the unit tangent vectors.
  ang = x(2*n+1:3*n);

% Picks out the distances.
  dt(1,:) = x(3*n+1:4*n-1);
  dt(2,:) = x(4*n:m);

% Calls function that compute the control points.
  C = ctpts(P,ang,dt);

% Calls function that computes the new dividing point positions.
  dpkpc = newk(Q,P);

% Calls the function that computes the sums of the square of the distances
% from the data points the nearest point on the cubic segment. The distance
% from the first and last data points to the first and last optimization
% points are also added to this sum.

  fp = P(:,1) - Q(:,1);
  lp = P(:,n) - Q(:,s);


  os = sod(C,Q,dpkpc) + fp'*fp + lp'*lp;


 disp( [ os ])
```

function uv = unitv(Q,k)

```
% function uv = unitv(Q,k)
% This function takes data points 'Q' from a separate file, and the position of
% knotpoints 'k'as input variables. It uses chord lengths parameterization to
% fit a parametric quadratic curve to five data points. The unit tangent
% vectors are approximated by the unit tangent vectors for these quadratic
% functions.The output is the set of unit tangent vectors in the direction of
% the knot points.

% Begin function.

  [r,m] = size(Q);

  n = length(k);

  for j = 1:n
      if j == 1, k(j) = 1; kt = 1;
      elseif j == n, k(j) = m-4; kt = 5;
      else k(j) = k(j)-2; kt = 3;
      end

% Extracting the knot point and four adjacent points
      x = Q(1,k(j):k(j)+4)';
      y = Q(2,k(j):k(j)+4)';

% Calculating the chord lengths
      xd = diff(x);
      yd = diff(y);
       d = sqrt( xd.*xd + yd.*yd);
       t(1) = 0; t(2) = d(1);
       t(3) = t(2) + d(2);
       t(4) = t(3) + d(3);
       t(5) = t(4) + d(4);

% Approximating coefficients of quadratic
      c = [ones(5,1)  t'  (t.*t)'] \ [x  y];

      u = c(2,:) + 2*c(3,:)*t(kt);

      u = u / norm(u);

      uv(:,j) = u';

  end
```

function nk = newk(Q,P)

```
% function nk = newk(Q,P)
% This function takes data points, Q; and knot points, P; as input. The
% function finds the closest data point of that cubic segment that is
% associated with that knot point, and returns a new k-array, nk.
% dpkpc is a global variable that is initially equal to the old k-array, k.

% Begin function.

  global dpkpc

  [r,m] = size(Q);

  [s,n] = size(P);

  nk(1) = 1;   nk(n) = m;


  for i = 2:n-1
      js = dpkpc(i-1); je = dpkpc(i+1); jm = dpkpc(i);
      z =  je-js+1;   mm = jm - js + 1;
      R = Q(:,js:je) - P(:,i) * ones(1,z);

      for jj = 1:z
          D(jj) = R(:,jj)' * R(:,jj);
      end

      sd = sign( D(mm) - D(mm+1) );
      while  D(mm) - D(mm+sd) > 0
         if mm == 2 & sd < 0
             break, end

         if mm == m-1 & sd > 0
             break, end

         mm = mm + sd;
      end

      nk(i) = mm + js - 1;
  end

  dpkpc = nk;
```

function sumofdist = sod(C,Q,dpkpc)

```
% function sumofdist = sod(C,Q,dpkpc)
% This function takes control points,C; data points,Q; and the position
% of the data point closest to that knot point within that segment. The
% function returns the sum of the distance squared from the data points
% to the nearest point on the cubic segment.

% Begin function

   n = length(C);
   [r,s] = size(Q);

   y = dpkpc;
   t = length(dpkpc);

% Initialize counter and sum.

   cntr = 0;
   sum = 0;

   for i = 1:3:n-3
       cntr = cntr + 1;
       for j = y(cntr):y(cntr+1)
           xy = NearestPoint( C(:,i:i+3)', Q(:,j)');    % See note.
           d = ( Q(:,j)' - xy );
           sum = sum + d * d';
           if j == y(cntr) &  i>1
               d2 = d*d';
               ds2 = ds*ds';
               dm = max(d2,ds2);
               sum = sum - dm;
           end
       end
       ds = d;
   end


   sumofdist = sum;

% Note: NearestPoint; obtained from 'Solving the Nearest Point-on-Curve
% Problem' and 'A Bezier Curve Root-Finder' developed by Philip J.
% Schneider in "Graphics Gems", Academic Press, 1990.
```

function C = ctpts(P,ang,dt)

```
% function C = ctpts(P,ang,dt)
% This function takes knot points,P;angle of the tangent vector, ang ; and
% the distance between successive knot points,dt; as input.It then
% computes the  unit tangent vector and the control points.

% Begin function.

  n = length(P);


% Converts the angle to its x and y components,and computes the control points.


  for k = 2:n-1

      u = [cos(ang(k)) ; sin(ang(k))];

      T = [T  P(:,k)-u*dt(2,k-1)  P(:,k)  P(:,k)+u*dt(1,k)];
  end

  u1 = [cos(ang(1)); sin(ang(1))]; un = [cos(ang(n)) ; sin(ang(n))];

  C = [P(:,1)  P(:,1)+u1*dt(1,1)  T  P(:,n)-un*dt(2,n-1)  P(:,n)];
```

function plotC = pltC(C,Q,P)

```
% function plotC = pltC(C,Q,P)
% This function takes as input: control points,C; data points,Q; and knot
% points. The control points are used to calculate the points of the
% approximating cubic Bezier curves. The control, data, and knot points
% are then plotted along with the curve.

% Begin function.

  [s,t] = size(C);

  x = [0:.025:1];
  [a,b] = size(x);

  W = [ ];
  for j = 1:3:t-3
      Y = zeros(2,b);
      M = [berny(3,0,x)' berny(3,1,x)' berny(3,2,x)' berny(3,3,x)'];
      Y = Y + C(:,j:j+3) * M' ;
      W = [W  Y];
  end

  plot( W(1,:) , W(2,:) ), hold

  plot( C(1,:) , C(2,:) )

  plot( Q(1,:) , Q(2,:) , '+')

  plot( P(1,:) , P(2,:) , 'x')

  plot( C(1,:) , C(2,:) , 'o')
```

function pop = poplt(x,Q)

```
% function pop = poplt(x,Q)
% This function picks out the knot points, angles, and distances from
% the optimization function x = 'fmins'. It then calls the function that
% computes the control points. Using the given data points; Q, a plot of
% the piecewise cubic curve is returned.

% Begin function.

m = length(x);
n = round(m/5);

% Picks out knot points,P; angles,ang; and distances, dt.

  P(1,:) = x(1:n);    P(2,:) = x(n+1:2*n);

  ang = x(2*n+1:3*n);

  dt(1,:) = x(3*n+1:4*n-1);    dt(2,:) = x(4*n:m);

% Calls the function that computes the control points.

  C = ctpts(P,ang,dt);

% Calls the function that plots the piecewise cubic curves.

  plotC(C,Q,P)
```

function val = berny(n, i, x)

```
% function val = berny(n, i, x)

% This function  is a non-recursive formula for Bernstein Polynomials which
% form a basis for Bezier curves. The inputs are the degree of the poly-
% nomial, n; the particular curve that is assigned a value of zero up to
% and including the degree, i; and the points between [0,1] to be evaluated, x.
% The output is the coordinates of points on the curve.

% Begin function.

   ni = [1 3 3 1];        % See note.

   m = size(x);

   if n < i
      val = zeros(m);
   elseif i < 0
      val = zeros(m);
   elseif ((n == 0) & (i == 0))
      val = 1;
   else
      val = ni(i+1) * (x.^i) .* ((ones(m) - x) .^(n-i));
   end

% Note: This function will only work for cubics (which are used throughout
% the supporting programs). For that reason, it is more efficient to use
% the coefficients for a third degree polynomial rather than one for a
% general n-degree polynomial.
```

LIST OF REFERENCES

1.   Weeg, G. P., and Reed, G. B., Introduction to Numerical Analysis, Blaisdell Publishing Co., 1966.

2.   Carnahan, B., and others, Applied Numerical Analysis, John Wiley & Sons, Inc., 1969.

3.   Ralston, A., and Rabinowitz, P., A First Course in Numerical Analysis, McGraw-Hill Book Co., 1978.

4.   Phillips, G.M., and Taylor, P. J., Theory and Applications of Numerical Analysis, Academic Press, 1973.

5.   Burden, R. L., and Faires, J. D., Numerical Analysis, fifth ed., PWS-Kent Publishing Co., 1993.

6.   Farin, G., Curves and Surfaces for Computer Aided Geometric Design, second ed., Academic Press, 1990.

7.   Foley, T. A., "Interpolation with Interval and Point Tension Controls Using Cubic Weighted ν-splines", ACM Transactions on Mathematical Software, V. 13, No. 1, 1987.

8.   Ahlberg, J. H., and others, The Theory of Splines and Their Applications, Academic Press, 1967.

9.   Epstein, M. P., "On the Influence of Parametrization in Parametric Interpolation", SIAM Journal on Numerical Analysis, V. 13, 1976.

10.   Foley, T. A., and Nielson, G. M., "Knot Selection for Parametric Spline Interpolation, Mathematical Methods in Computer Aided Geometric Design, T. Lynch and L. Schumaker (eds.), Academic Press, 1989.

11.   deBoor, C., A Practical Guide to Splines, Springer-Verlag, 1978.

12.   Gregory, J. A., "Geometric Continuity", Mathematical Methods in Computer Aided Geometric Design, T. Lynch and L. Schumaker (eds.), Academic Press, 1989.

13.   Buchanan, J. D., and Turner, P. R., Numerical Methods and Analysis, McGraw-Hill, 1992.

14. Gerald, C. F., and Wheatley, P. O., <u>Applied Numerical Analysis</u>, fourth edition, Addison-Wesley, 1989.

15. Plass, M., and Stone, M., "Curve Fitting with Piecewise Parametric Cubics", <u>Computer Graphics</u>, V. 17, No. 3, 1983.

16. Reeves, W. T., "Quantitative Representation of Complex Dynamic Shapes for Motion Analysis", Ph.D. Thesis, Department of Computer Science, University of Toronto, 1980.

17. Powell, M. J. D., "Curve Fitting by Splines in One Variable", <u>Numerical Approximation to Functions and Data</u>, J. G. Hayes (ed.), Athlone Press, 1970.

18. Ichida, K., and Kiyono, T., "Curve Fitting by a One-Pass Method with a Piecewise Cubic Polynomial", <u>ACM Transactions on Mathematical Software</u>, V. 3, No. 2, 1977.

19. Scheider, P. J., "Solving the Nearest-Point-on-Curve Problem", <u>Graphics Gems</u>, A. S. Glassner (ed.), Academic Press, 1990.

20. Marin, S. P., and Smith, P. W., "Parametric Approximation of Data Using ODR Splines", to appear in <u>Computer Aided Geometric Design</u>.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center   2
    Cameron Station
    Alexandria, VA 22304-6145

2. Library, Code 52   2
    Naval Postgraduate School
    Monterey, CA 93943-5002

3. Professor Richard Franke, Code MA/Fe   4
    Department of Mathematics
    Naval Postgraduate School
    Monterey, CA 93943-5216

4. Professor Carlos F. Borges, Code MA/Bc   1
    Department of Mathematics
    Naval Postgraduate School
    Monterey, CA 93943-5216

5. LT Marion R. Holmes   2
    9231 Fairhaven PL
    Jonesboro, GA 30236

6. Dr. Sharon E. Wormly   1
    4822 Verano PL
    Irvine, CA 92715

7. Professor Henry Gore   1
    Department of Mathematics
    Morehouse College
    830 Westview Drive S.W.
    Atlanta, GA 30314

8. Dr. Samuel P. Marin   1
    Mathematics Department
    General Motors Research Laboratories
    Warren, MI 48090-9055

9. Professor G.M. Nielson   1
    Department of Computer Science
    Arizona State University
    Tempe, AZ 85287