

NAVAL POSTGRADUATE SCHOOL
Monterey, California



19980810 033

THESIS

BÉZIER CURVE FITTING

by

Tim Andrew Pastva

September 1998

Advisor:
Second Reader:

Carlos F. Borges
Richard Franke

Approved for public release; Distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, Va 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1998		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE BÉZIER CURVE FITTING				5. FUNDING NUMBERS
6. AUTHORS Pastva, Tim A.				
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) We typically think of fitting data with an approximating curve in the linear least squares sense, where the sum of the residuals in the vertical, or y , direction is minimized. The problem addressed here is to fit a Bézier curve to an ordered set of data in the total least squares sense, where the sum of the residuals in both the horizontal and vertical directions is minimized. More exact: given an ordered set of m data points \mathbf{d}_i , $i = 1, 2, \dots, m$ find a set of control points \mathbf{b}_i , $i = 0, 1, \dots, n$ where n is the order of the Bézier curve, and a vector \mathbf{t} of nodes, $0 \leq t_1 \leq t_2 \leq \dots \leq t_m \leq 1$ that minimize $\ B(\mathbf{t})P - D\ _F$. The matrix D contains the data points, the matrix P contains the control points, and the matrix $B(\mathbf{t})$ is a Bernstein matrix. The algorithm to accomplish this is explained in detail and makes extensive use of the linear algebra representation of Bézier curves.				
14. SUBJECT TERMS Bézier Curves, Gauss-Newton Method, Affine Invariant Node Spacing				15. NUMBER OF PAGES 75
				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

Approved for public release; distribution is unlimited

BÉZIER CURVE FITTING

Tim A. Pastva
Major, United States Marine Corps
B.S., University of Michigan, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
September 1998

Author:

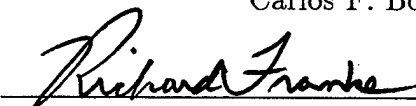


Tim A. Pastva

Approved by:



Carlos F. Borges, Advisor



Richard Franke, Second Reader



Max Woods, Chairman, Department of Mathematics

ABSTRACT

We typically think of fitting data with an approximating curve in the linear least squares sense, where the sum of the residuals in the vertical, or y , direction is minimized. The problem addressed here is to fit a Bézier curve to an ordered set of data in the total least squares sense, where the sum of the residuals in both the horizontal and vertical directions is minimized. More exact: given an ordered set of m data points \mathbf{d}_i , $i = 1, 2, \dots, m$ find a set of control points \mathbf{b}_i , $i = 0, 1, \dots, n$ where n is the order of the Bézier curve, and a vector \mathbf{t} of nodes, $0 \leq t_1 \leq t_2 \leq \dots \leq t_m \leq 1$ that minimize $\| B(\mathbf{t}) P - D \|_F$. The matrix D contains the data points, the matrix P contains the control points, and the matrix $B(\mathbf{t})$ is a Bernstein matrix. The algorithm to accomplish this is explained in detail and makes extensive use of the linear algebra representation of Bézier curves.

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	BERNSTEIN POLYNOMIALS	1
B.	BÉZIER CURVES	1
C.	PROPERTIES OF BÉZIER CURVES	3
1.	Endpoint Interpolation	3
2.	Affine Invariance	3
D.	MATRIX REPRESENTATION OF BÉZIER CURVES	4
1.	The Bernstein Matrix	4
2.	Slope of a Bézier Curve	6
II.	PROBLEM STATEMENT	11
A.	PROBLEM STATEMENT	11
B.	THE SEPARABLE LEAST SQUARES PROBLEM	12
C.	SHORTEST DISTANCE TO A CURVE	13
1.	A Necessary Condition	14
D.	ORDERED DATA	15
III.	BASIC ALGORITHM	21
A.	BASIC ALGORITHM	21
1.	Approaching $\{P^*, t^*\}$	21
B.	THE NEARESTPOINT METHOD	22
C.	THE GRADIENT METHOD	23
1.	Results of the Gradient Method	24
D.	THE GAUSS-NEWTON METHOD	24
1.	Review of Newton's Method	25
2.	Solving the Nonlinear Least Squares Problem	25
3.	Stopping Criteria	28
E.	PRIMARY ALGORITHM	29

IV. METHOD EXAMPLES	31
A. GRAPHICAL COMPARISONS	31
1. Gradient Method and Approaching $\{P^*, t^*\}$	31
2. Total Least Squares Versus Linear Least Squares	32
3. Effect of the Initial Set of Nodes	32
4. Failure to Maintain Ordering of Data	33
5. Gauss-Newton Method and Data with Noise	33
6. Gauss-Newton and Real World Data	34
B. GAUSS-NEWTON VERSUS NEARESTPOINT	35
1. Graphical Results	35
2. Computer Time and Iterations	35
C. GENERAL COMPARISON OF FUNCTIONS	35
APPENDIX. MATLAB FUNCTIONS	47
LIST OF REFERENCES	57
INITIAL DISTRIBUTION LIST	59

LIST OF FIGURES

1.	Cubic Bézier Curve	9
2.	Cubic Bézier Curve with Reordered Control Points	9
3.	Data Points and Fitted Curve	18
4.	Solution for the Nearest Points	18
5.	Data and Nodes for Linear Least Squares Problem	19
6.	Solution to Linear Least Squares Problem	19
7.	Gradient Method	30
8.	Gradient Method	37
9.	Gradient Method	37
10.	Linear Least Squares Fit	38
11.	Gauss-Newton Fit	38
12.	Affine Invariant Chord Method	39
13.	Affine Invariant Angle Method	39
14.	Failure to Maintain Order	40
15.	Affine Invariant Angle Maintains Order	40
16.	Data With Added Noise	41
17.	Fitting Data With Added Noise	41
18.	Curve With Too Much Freedom	42
19.	Real World Data and <i>grad7.m</i>	43
20.	Data Set One and <i>grad7.m</i>	44
21.	Data Set One and <i>grad5.m</i>	44
22.	Data Set Two and <i>grad7.m</i>	45
23.	Data Set Two and <i>grad5.m</i>	45
24.	Data Set Three and <i>grad7.m</i>	46
25.	Data Set Three and <i>grad5.m</i>	46

LIST OF TABLES

I.	Computer Time Used	36
II.	General Comparison of Functions	36

ACKNOWLEDGMENTS

This paper is the result of the ideas and guidance of Carlos Borges. I greatly appreciate the detailed review by Richard Franke along with his help and advice, and I appreciate the help from Sam Buttrey in getting over several hurdles. A big thanks goes to David Canright for his thesis style for L^AT_EX.

Most importantly, I thank God for the learning ability I have and for leading me to this school.

I. INTRODUCTION

Notation used in this paper is as follows: *matrices* are slanted upper case letters, *vectors* are lower case bold letters, and *scalars* are slanted lower case letters. Also, vectors are column arrays unless otherwise indicated. For ease of discussion, a *point* and a vector are equivalent.

A. BERNSTEIN POLYNOMIALS

Bernstein polynomials have the following general form:

$$B_i^n(t) = \binom{n}{i} \frac{(t-a)^i(b-t)^{n-i}}{(b-a)^n}, \quad i = 0, 1, \dots, n \quad (\text{I.1})$$

where $t \in [a, b]$.

We will only consider the case where $a = 0$ and $b = 1$. Equation (I.1) now takes on the form;

$$B_i^n(t) = \binom{n}{i} (t)^i (1-t)^{n-i}, \quad i = 0, 1, \dots, n \quad (\text{I.2})$$

which is a scalar for a particular $t \in [0, 1]$. From Equation (I.2) note that $B_0^0(t) = 1$. Also, we define $B_j^n(t) = 0$ if $j < 0$ or $j > n$.

The set of Bernstein polynomials of degree n form a basis for \mathcal{P}_n , the space of polynomials of degree n or less. The linear transformation from power basis coefficients to Bernstein basis coefficients is explained in detail in [Ref. 1]. For a complete discussion on the properties of Bernstein polynomials see [Ref. 2].

B. BÉZIER CURVES

Bézier curves are named after P. Bézier and are used extensively in computer aided geometric design. Before presenting the general form for a degree n Bézier curve, let us look at an example.

Consider two points on the x -axis given by $\mathbf{b}_0^T = (2, 0)$ and $\mathbf{b}_1^T = (4, 0)$, and suppose that we want to describe a degree 1 curve between these two points. A

parametric representation of this curve is;

$$x(t) = 2 + 2t, \quad y(t) = 0, \quad t \in [0, 1]$$

Note that since $B_i^1(t)$ for $i = 0, 1$ is a basis for \mathcal{P}_1 , we can write $x(t)$ and $y(t)$ in terms of Bernstein polynomials. Using matrices, we have;

$$\begin{aligned} [x(t), y(t)] &= [2 + 2t, 0] \\ &= [2(1 - t) + 4t, 0(1 - t) + 0t] \\ &= \begin{bmatrix} B_0^1(t) & B_1^1(t) \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 4 & 0 \end{bmatrix} \\ &= \begin{bmatrix} B_0^1(t) & B_1^1(t) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0^T \\ \mathbf{b}_1^T \end{bmatrix} \end{aligned} \quad (I.3)$$

The parametric representation of a curve in the form of Equation (I.3) is of central importance in this paper. Also, all examples in this paper are in 2-dimensions, but it is understood that this representation holds for higher dimensional space as well.

In general, a Bézier curve of degree n , denoted $\mathbf{b}_0^n(t)$, can be written as

$$\begin{aligned} (\mathbf{b}_0^n(t))^T &= \sum_{i=0}^n B_i^n(t) \mathbf{b}_i^T \\ &= B_0^n(t) \mathbf{b}_0^T + B_1^n(t) \mathbf{b}_1^T + \dots + B_n^n(t) \mathbf{b}_n^T \end{aligned} \quad (I.4)$$

where $\mathbf{b}_0^n(t)$ is a point on the curve for a particular $t \in [0, 1]$. The points $\mathbf{b}_i, i = 0, \dots, n$ are called *control points*. We see that a point on a Bézier curve is a weighted sum of control points, where the weights are Bernstein polynomials evaluated at a particular value of t .

Let us look at an example of a cubic Bézier curve before discussing properties of these curves. Figure (1) on page 9 shows four control points and a curve starting at control point \mathbf{b}_0 and ending at control point \mathbf{b}_3 . Note that we need $n + 1$ control points for a degree n curve. Also note that, in this example, the curve does not pass through \mathbf{b}_1 or \mathbf{b}_2 .

Figure (2) on page 9 shows how the curve changes when the positions of control points \mathbf{b}_1 and \mathbf{b}_2 are interchanged. This is why Bézier curves are so widely used in computed aided geometric design; by changing the position of one or more control points the user can easily change the shape of the curve.

C. PROPERTIES OF BÉZIER CURVES

The following properties are basic to understanding Bézier curves and provide the necessary background for discussion later in this paper.

1. Endpoint Interpolation

Bézier curves interpolate between the end control points \mathbf{b}_0 and \mathbf{b}_n . We saw in Figure (1) that the Bézier curve had \mathbf{b}_0 as one endpoint and \mathbf{b}_3 as the other. This is always the case and is shown using Equation (I.4) with $t = 0$ and $t = 1$. Because $B_i^n(0) = 0$ except for $i = 0$, and $B_i^n(1) = 0$ except for $i = n$, we have;

$$(\mathbf{b}_0^n(0))^T = \sum_{i=0}^n B_i^n(0) \mathbf{b}_i^T = \mathbf{b}_0^T$$

and

$$(\mathbf{b}_0^n(1))^T = \sum_{i=0}^n B_i^n(1) \mathbf{b}_i^T = \mathbf{b}_n^T$$

Though Bézier curves are guaranteed to begin at \mathbf{b}_0 and end at \mathbf{b}_n , it is not guaranteed that they pass through any of the intermediate control points $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{n-1}$.

2. Affine Invariance

An affine transformation is of the form $\Phi(\mathbf{p}(t)) = A\mathbf{p}(t) + \mathbf{c}$. Examples of affine transformations are scaling, rotation, and reflection. The equation describing Bézier curves is affine invariant in that given

$$(\mathbf{b}_0^n(t))^T = \sum_{i=0}^n B_i^n(t) \mathbf{b}_i^T$$

under the transformation Φ we will have

$$\Phi((\mathbf{b}_0^n(t))^T) = \sum_{i=0}^n B_i^n(t) \Phi(\mathbf{b}_i^T)$$

In other words, given an affine transformation Φ , performing the following two steps produces equivalent results.

1. Evaluate Equation (I.4) for a given set of control points and a value t to obtain $(\mathbf{b}_0^n(t))^T$, and then apply Φ to $(\mathbf{b}_0^n(t))^T$.
2. Apply Φ to each of the control points and then evaluate Equation (I.4) using the transformed control points and the same value t as in 1.

For example, suppose we want to rotate a given Bézier curve. We can either transform the relatively few control points or transform all the points of the curve. In general, transforming the control points and re-plotting the curve costs much less. Other properties of the Bézier curve are discussed in detail in [Ref. 2].

D. MATRIX REPRESENTATION OF BÉZIER CURVES

In this section we represent Bézier curves in linear algebra form. This paper makes extensive use of this representation for Bézier curves to simplify their manipulation.

1. The Bernstein Matrix

Equation (I.4) showed that a Bézier curve is described by

$$(\mathbf{b}_0^n(t))^T = \sum_{i=0}^n B_i^n(t) \mathbf{b}_i^T, \quad t \in [0, 1]$$

This is equivalent to the linear algebra form:

$$(\mathbf{b}_0^n(t))^T = [B_0^n(t) \dots B_n^n(t)] \begin{bmatrix} \mathbf{b}_0^T \\ \vdots \\ \mathbf{b}_n^T \end{bmatrix} = B(t) P \quad (\text{I.5})$$

where we will refer to $B(t)$ as a *Bernstein matrix*. A Bernstein matrix is a generalized Vandermonde matrix which, for a vector $\mathbf{t} = [t_1, t_2, \dots, t_m]^T$, $t_i \in [0, 1]$ and a given

degree n , has the form;

$$B(\mathbf{t}) = \begin{bmatrix} B_0^n(t_1) & B_1^n(t_1) & \cdots & B_n^n(t_1) \\ B_0^n(t_2) & B_1^n(t_2) & \cdots & B_n^n(t_2) \\ \vdots & \vdots & & \vdots \\ B_0^n(t_m) & B_1^n(t_m) & \cdots & B_n^n(t_m) \end{bmatrix}$$

Therefore, we see that for $t \in \mathcal{R}^m$ and a given degree n we have $B(\mathbf{t}) \in \mathcal{R}^{m \times (n+1)}$.

In 2-dimension, the matrix P contains the x and y coordinates of the control points and has the following form:

$$P = \begin{bmatrix} x_0 & y_0 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

For example, if instead of evaluating Equation (1.5) for a particular t we are interested in getting $m = 3$ points on a Bézier curve of degree $n = 3$ we would evaluate;

$$(\mathbf{b}_0^n(\mathbf{t}))^T = B(\mathbf{t}) P$$

where $B(\mathbf{t}) \in \mathcal{R}^{3 \times 4}$ and $P \in \mathcal{R}^{4 \times 2}$. Therefore, $(\mathbf{b}_0^n(\mathbf{t}))^T \in \mathcal{R}^{3 \times 2}$ and is a matrix of points on the curve.

As another example, let $n = 3$ and only consider a particular t . We will expand the elements of $B(t)$ to further demonstrate the inherent linear algebra form of Bézier curve representation. Note that we can write

$$B(t)^T = \begin{bmatrix} B_0^3(t) \\ B_1^3(t) \\ B_2^3(t) \\ B_3^3(t) \end{bmatrix} = \begin{bmatrix} -t^3 + 3t^2 - 3t + 1 \\ 3t^3 - 6t^2 + 3t \\ -3t^3 + 3t^2 \\ t^3 \end{bmatrix}$$

and, therefore, we see that

$$B(t)^T = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} = M^T T^T$$

where $T \in \mathcal{R}^{1 \times 4}$ for the array. Equivalently, we have;

$$B(t) = T M \quad (\text{I.6})$$

If we now consider a vector $\mathbf{t} \in \mathcal{R}^m$, then instead of $T \in \mathcal{R}^{1 \times 4}$ we would have $T \in \mathcal{R}^{m \times 4}$. For a given matrix of control points we get the following equation which describes m points on a degree 3 Bézier curve.

$$\begin{aligned} (\mathbf{b}_0^3(\mathbf{t}))^T &= \begin{bmatrix} t_1^3 & t_1^2 & t_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ t_m^3 & t_m^2 & t_m & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} \\ &= T M P \\ &= B(\mathbf{t}) P \end{aligned}$$

2. Slope of a Bézier Curve

Now that we have an equation for a degree n Bézier curve, we will derive the equation for the first derivative at a particular t , which in turn gives the slope at a point on the curve. Since Bézier curves are parametric in t , we want to solve

$$\begin{aligned} \frac{d}{dt}(\mathbf{b}_0^n(t))^T &= \frac{d}{dt} \sum_{i=0}^n B_i^n(t) \mathbf{b}_i^T \\ &= \sum_{i=0}^n \frac{d}{dt} B_i^n(t) \mathbf{b}_i^T \end{aligned}$$

From [Ref. 2: page 46] we have

$$\frac{d}{dt} B_i^n(t) = n B_{i-1}^{n-1}(t) - n B_i^{n-1}(t)$$

and, therefore,

$$\begin{aligned}
\frac{d}{dt}(\mathbf{b}_0^n(t))^T &= \sum_{i=0}^n (n B_{i-1}^{n-1} - n B_i^{n-1}) \mathbf{b}_i^T \\
&= (n[0 - B_0^{n-1}(t)]) \mathbf{b}_0^T + (n[B_0^{n-1}(t) - B_1^{n-1}(t)]) \mathbf{b}_1^T + \dots \\
&\quad + (n[B_{n-1}^{n-1}(t) - 0]) \mathbf{b}_n^T
\end{aligned}$$

After combining like terms of $n B_i^{n-1}(t)$ we have, finally

$$\begin{aligned}
\frac{d}{dt}(\mathbf{b}_0^n(t))^T &= n B_0^{n-1}(t)(\mathbf{b}_1^T - \mathbf{b}_0^T) + n B_1^{n-1}(t)(\mathbf{b}_2^T - \mathbf{b}_1^T) + \dots \\
&\quad + n B_{n-1}^{n-1}(t)(\mathbf{b}_n^T - \mathbf{b}_{n-1}^T) \\
&= n \sum_{i=0}^{n-1} B_i^{n-1}(t)(\mathbf{b}_{i+1}^T - \mathbf{b}_i^T) \\
&= n \sum_{i=0}^{n-1} B_i^{n-1}(t) \Delta \mathbf{b}_i^T \tag{I.7}
\end{aligned}$$

where Δ is the forward difference operator.

Equation (I.7) lends itself to representation in linear algebra form as follows;

$$\begin{aligned}
\frac{d}{dt}(\mathbf{b}_0^n(t))^T &= n [B_0^{n-1}(t) \ B_1^{n-1}(t) \ \dots \ B_{n-1}^{n-1}(t)] \Delta \begin{bmatrix} \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \vdots \\ \mathbf{b}_n^T \end{bmatrix} \\
&= n \hat{B}(t) \Delta P
\end{aligned}$$

where $\Delta \in \mathcal{R}^{n \times (n+1)}$ and has the form

$$\begin{bmatrix} -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 \\ \vdots & . & \ddots & \ddots & & \vdots \\ 0 & \dots & \dots & 0 & -1 & 1 \end{bmatrix}$$

For example, consider the case where $n = 3$. Since we have that

$$(\hat{B}(t))^T = \begin{bmatrix} B_0^2(t) \\ B_1^2(t) \\ B_2^2(t) \end{bmatrix} = \begin{bmatrix} t^2 - 2t + 1 \\ -2t^2 + 2t \\ t^2 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^2 \\ t \\ 1 \end{bmatrix}$$

we can write

$$\frac{d}{dt}(\mathbf{b}_0^3(t))^T = 3 \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \\ \mathbf{b}_3^T \end{bmatrix}$$

We can obtain the first derivative at m points along a Bézier curve by evaluating Equation (I.7) for a given vector $\mathbf{t} \in \mathcal{R}^m$ and $t_i \in [0, 1]$. For example, when $n = 3$ we have;

$$\frac{d}{dt}(\mathbf{b}_0^3(\mathbf{t}))^T = 3 \begin{bmatrix} t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ \vdots & \vdots & \vdots \\ t_m^2 & t_m & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \\ \mathbf{b}_3^T \end{bmatrix}$$

There are similar equations for higher derivatives, for these see [Ref. 2].

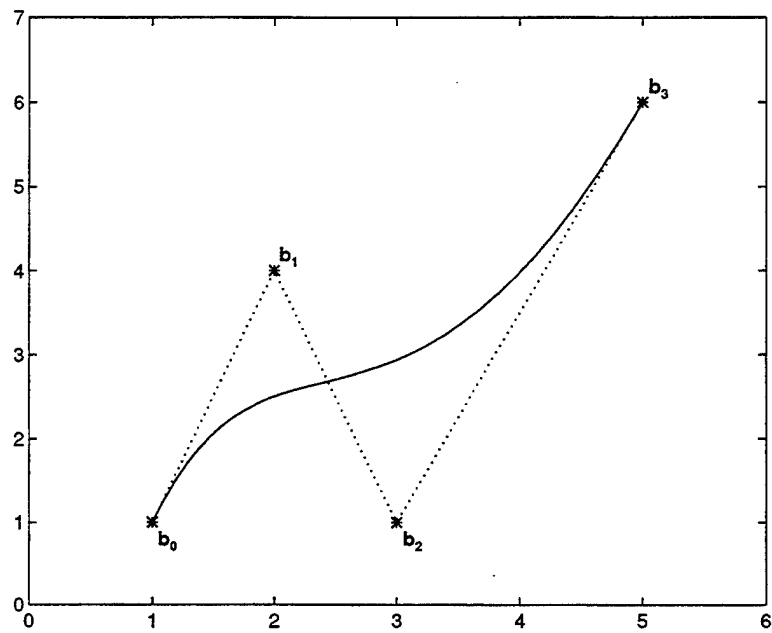


Figure 1. Cubic Bézier Curve with Control Points

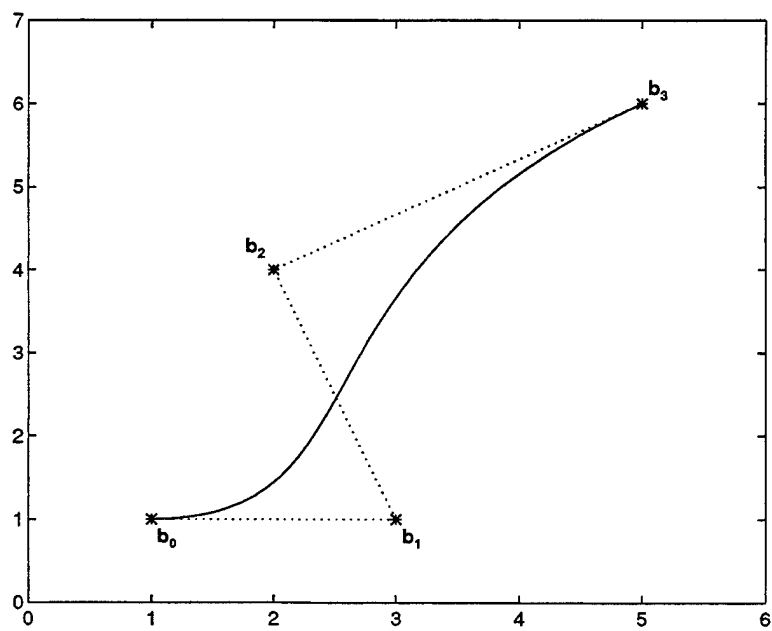


Figure 2. Cubic Bézier Curve with Reordered Control Points

II. PROBLEM STATEMENT

This chapter presents the problem of fitting a given ordered set of data with a Bézier curve in the total least squares sense. We will see how the linear algebra representation of Bézier curves lends itself to solving this problem. We refer to the point along a Bézier curve determined by a particular node t_i as the *point* τ_i . So, from Equation (I.4) we have

$$\tau_i = \mathbf{b}_0^n(t_i)$$

A. PROBLEM STATEMENT

The problem to solve is stated as follows: given an ordered set of m data points \mathbf{d}_i , $i = 1, 2, \dots, m$ find a set of control points \mathbf{b}_i , $i = 0, 1, \dots, n$, where n is the order of the Bézier curve, and a vector \mathbf{t} of nodes, $0 \leq t_1 \leq t_2 \leq \dots \leq t_m \leq 1$ that minimize

$$\| B(\mathbf{t}) P - D \|_F \quad (\text{II.1})$$

where the matrix D contains the data points. Though neither the matrix P^* nor the vector \mathbf{t}^* that minimize Equation (II.1) are unique, and though in practice the resulting value of Equation (II.1) is determined in part by the stopping criteria of the algorithm, for ease of reference we will equate minimizing Equation (II.1) with determining $\{P^*, \mathbf{t}^*\}$. Also, we will only consider the case where $n < m$ since for $n \geq m$ we could produce a curve which passes through all the data points and this is uninteresting in the context of this paper.

Notice that we are minimizing the *Frobenius norm* of the residual in Equation (II.1). For $A \in \mathcal{R}^{m \times n}$, this norm is given by

$$\| A \|_F = \left(\sum_{i=1}^m \sum_{j=1}^n a_{i,j}^2 \right)^{\frac{1}{2}}$$

Consider the case $m = 3$ and $n = 1$. The residual in Equation (II.1) is then

$$\begin{bmatrix} B_0^1(t_1) & B_1^1(t_1) \\ B_0^1(t_2) & B_1^1(t_2) \\ B_0^1(t_3) & B_1^1(t_3) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0^T \\ \mathbf{b}_1^T \end{bmatrix} - \begin{bmatrix} \mathbf{d}_1^T \\ \mathbf{d}_2^T \\ \mathbf{d}_3^T \end{bmatrix}$$

However, minimizing the Frobenius norm is the same as minimizing the 2 norm of

$$\begin{bmatrix} B_0^1(t_1) & B_1^1(t_1) & 0 & 0 \\ B_0^1(t_2) & B_1^1(t_2) & 0 & 0 \\ B_0^1(t_3) & B_1^1(t_3) & 0 & 0 \\ 0 & 0 & B_0^1(t_1) & B_1^1(t_1) \\ 0 & 0 & B_0^1(t_2) & B_1^1(t_2) \\ 0 & 0 & B_0^1(t_3) & B_1^1(t_3) \end{bmatrix} \begin{bmatrix} b_{0,x} \\ b_{1,x} \\ b_{0,y} \\ b_{1,y} \end{bmatrix} - \begin{bmatrix} d_{1,x} \\ d_{2,x} \\ d_{3,x} \\ d_{1,y} \\ d_{2,y} \\ d_{3,y} \end{bmatrix} \quad (\text{II.2})$$

where, for example, $b_{0,y}$ denotes the y component of the vector \mathbf{b}_0 . We will call Equation (II.2) the *uncoupled* form of the residual in Equation (II.1).

B. THE SEPARABLE LEAST SQUARES PROBLEM

Minimizing Equation (II.1) is a nonlinear least squares problem because it is nonlinear in t . Minimizing Equation (II.1) is also *separable* because we can separate out from the original nonlinear problem a linear least squares parametric functional problem for the matrix P . In particular, for a given vector \mathbf{t} , the minimizing matrix P of the residual in Equation (II.1) satisfies

$$P = B^+(\mathbf{t})D$$

where $B^+(\mathbf{t})$ is the Moore-Penrose generalized inverse, or pseudo-inverse, of $B(\mathbf{t})$.

Note, since we have that $P = B^+(\mathbf{t})D$ for a given \mathbf{t} that we are able to separate the unknown matrix P from Equation (II.1). Therefore, the optimal vector \mathbf{t} is found by minimizing the *variable projection functional*

$$\| B(\mathbf{t})B^+(\mathbf{t})D - D \|_F$$

Further, consider the nonlinear least squares problem

$$B(\mathbf{t})P = \begin{bmatrix} B_0^2(t_1) & B_1^2(t_1) & B_2^2(t_1) \\ \vdots & \vdots & \vdots \\ B_0^2(t_4) & B_1^2(t_4) & B_2^2(t_4) \end{bmatrix} \begin{bmatrix} \mathbf{b}_0^T \\ \mathbf{b}_1^T \\ \mathbf{b}_2^T \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1^T \\ \vdots \\ \mathbf{d}_4^T \end{bmatrix} \quad (\text{II.3})$$

where the matrix $P \in \mathcal{R}^{4 \times 2}$ and the vector $\mathbf{t} \in \mathcal{R}^4$ are unknown and $n = 2$. Note that if the vector \mathbf{t} is given, then our least squares problem becomes linear. If, instead, we are given a matrix P , then Equation (II.3) is a nonlinear least squares problem for the vector \mathbf{t} only.

C. SHORTEST DISTANCE TO A CURVE

Given an ordered set of data, minimizing Equation (II.1) determines the points τ which are nearest to their associated data points for a particular control point matrix P . We will refer to these particular points as the *nearest points*. For example, let $\tau_{i,x}$ and $\tau_{i,y}$ represent the x and y values of the point τ_i for a particular matrix P . We saw that the nodes and control points that minimize Equation (II.1) also minimize the 2 norm of Equation (II.2). This is equivalent to minimizing the *objective function*;

$$(\tau_{1,x} - d_{1,x})^2 + \cdots + (\tau_{m,x} - d_{m,x})^2 + (\tau_{1,y} - d_{1,y})^2 + \cdots + (\tau_{m,y} - d_{m,y})^2 \quad (\text{II.4})$$

For a particular matrix P , a change in the node t_i only affects the point τ_i . Therefore, Equation (II.4) can be broken down into m independent objective functions. For example, the node t_1 only affects the following terms of Equation (II.4)

$$(\tau_{1,x} - d_{1,x})^2 + (\tau_{1,y} - d_{1,y})^2 \quad (\text{II.5})$$

Minimizing Equation (II.5) is equivalent to determining the nearest point for data point $(d_{1,x}, d_{1,y})$. Proceeding like this for each node, we can determine each nearest point and minimize Equation (II.1) for a particular matrix P . So we see that with a given matrix P the nonlinear least squares problem for the vector \mathbf{t} is equivalent to solving the nearest point problem.

1. A Necessary Condition

Treating t_1 as a variable, Equation (II.5) is minimized by finding the stationary point of

$$g(t_1) = (\tau_{1,x} - d_{1,x})^2 + (\tau_{1,y} - d_{1,y})^2$$

The stationary point is such that

$$\frac{d}{dt_1}g(t_1) = 2(\tau_{1,x} - d_{1,x})\frac{d}{dt_1}\tau_{1,x} + 2(\tau_{1,y} - d_{1,y})\frac{d}{dt_1}\tau_{1,y} = 0$$

In matrix notation this becomes

$$\begin{bmatrix} \frac{d}{dt_1}\tau_{1,x} & \frac{d}{dt_1}\tau_{1,y} \end{bmatrix} \begin{bmatrix} \tau_{1,x} - d_{1,x} \\ \tau_{1,y} - d_{1,y} \end{bmatrix} = 0 \quad (\text{II.6})$$

So, we see that the point τ_i which minimizes Equation (II.5) is perpendicular to the tangent vector at that same point. In general, this is a necessary condition for minimizing Equation (II.5).

Figure (3) on page 18 shows a given ordered set of data and the degree 2 Bézier curve produced from a given set of control points. Figure (4) on page 18 shows the points τ gotten by solving the nonlinear least squares problem for the vector \mathbf{t} . Notice that each τ_i satisfies the necessary condition except for the point τ_4 . Because our parameter values are constrained within $[0, 1]$, $t_4 = 1$ is the best we can do.

Now, consider again the problem of minimizing Equation (II.4) except we are given an initial set of nodes. Figure (5) on page 19 shows the same data set as used in Figure (3) and the node associated with each data point. The nodes were chosen uniformly spaced on the interval $[0, 1]$ and are given by

$$\mathbf{t} = \begin{bmatrix} 0 \\ .33 \\ .67 \\ 1 \end{bmatrix}$$

The degree of the Bézier curve we want to fit to the data is again two. After solving the linear least squares parameter functional problem for the control points, we can

plot the Bézier curve. Figure (6) on page 19 shows the resulting Bézier curve along with the points τ . Note that point τ_3 does not minimize Equation (II.5) or satisfy the necessary condition. What the least squares solution for the control points produced was the best answer possible given the initial set of nodes and desired degree of the Bézier curve.

D. ORDERED DATA

In the problem statement at the beginning of the chapter, we are given an ordered set of m data points. We will order the data with respect to t because Bézier curves are parametric in t . So, $t_j > t_i$ implies that the data point associated with t_j is ordered *after* the data point associated with t_i . Therefore, the problem of ordering a set of data with respect to t becomes one of determining an initial set of nodes.

Consider a set of m data points and that we want to determine a set of control points which will produce an approximating Bézier curve. We need an initial set of nodes in order to solve the least squares problem for the matrix P , and we want the initial set of points τ in the neighborhood of the nearest points. The reason for this last condition on the initial set of points τ is explained in Chapter III. Since the elements of \mathbf{t} must be contained in $[0, 1]$, we might use either the *uniform* method

$$t_i = \frac{i-1}{m-1}, \quad i = 1, \dots, m$$

or the *chord length* method

$$t_i = t_{i-1} + \frac{\|\mathbf{d}_i - \mathbf{d}_{i-1}\|_2}{\sum_{i=2}^m \|\mathbf{d}_i - \mathbf{d}_{i-1}\|_2}, \quad i = 2, 3, \dots, m$$

where we define $t_1 = 0$.

The main problem with the uniform method is that it does not take into account nonuniform distribution of the data. The main problem with the chord length method is that it is not invariant under all affine transformations [Ref. 3]. For example, consider a set of data and that the user wants to scale one of the components by a constant while using the same initial set of nodes in the new scaling. Because

the chord length method is not invariant under scaling, the resulting Bézier curves would not have the same relationship to the data.

The *affine invariant chord* method as described in [Ref. 3] takes into account nonuniform spacing of data and is based on the following *metric*

$$\begin{aligned} |D_{i+1} - D_i|_D^2 &= [x_{i+1} - x_i, y_{i+1} - y_i] A [x_{i+1} - x_i, y_{i+1} - y_i]^T \\ &= [x_{i+1} - x_i, y_{i+1} - y_i] \begin{bmatrix} \frac{\sigma_Y^2}{g} & -\frac{\sigma_{XY}}{g} \\ -\frac{\sigma_{XY}}{g} & \frac{\sigma_X^2}{g} \end{bmatrix} \begin{bmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \end{bmatrix} \quad (\text{II.7}) \end{aligned}$$

where D_i and D_{i+1} are successive data points, σ_Y^2 is the sample variance in the y components of the entire data set, σ_X^2 is the sample variance in the x components of the entire data set, σ_{XY} is the sample covariance in the x and y components of the entire data set, and

$$\begin{aligned} g &= |A^{-1}| \\ &= \sigma_X^2 \sigma_Y^2 - (\sigma_{XY})^2 \end{aligned}$$

The matrix A is the inverse of the *covariance matrix* used in the *bivariate normal distribution function*.

The term *metric* means a way to measure distance. Normally, we think of the Euclidean metric

$$\begin{aligned} \|D_{i+1} - D_i\|^2 &= [x_{i+1} - x_i, y_{i+1} - y_i] I [x_{i+1} - x_i, y_{i+1} - y_i]^T \\ &= (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 \end{aligned}$$

Note that in the case of the Euclidean metric $A = I$. The reason is that the Euclidean metric assumes no scaling or correlation between the data points.

An improvement on the affine invariant chord method is the *affine invariant angle* method. This method combines the metric in Equation (II.7) and the *bending* of the data. Let three noncollinear data points be the vertices of a triangle. The bending of the data is how much of a turn is made when going from one side of the

triangle to another at a vertex. For details of the construction of this method see [Ref. 4]. The primary algorithm of this paper to solve the problem stated at the beginning of the chapter uses the affine invariant angle method to obtain the initial set of nodes.

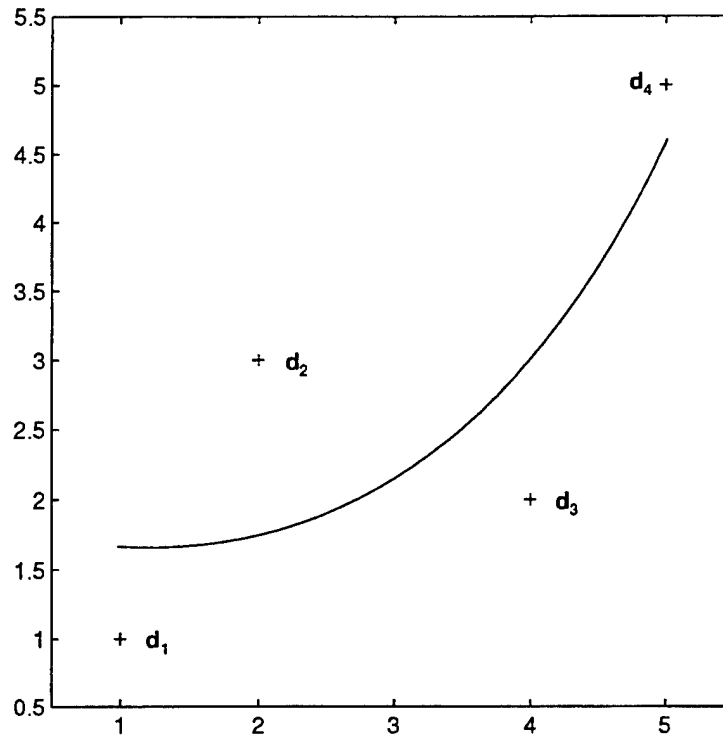


Figure 3. Data Points and Fitted Curve

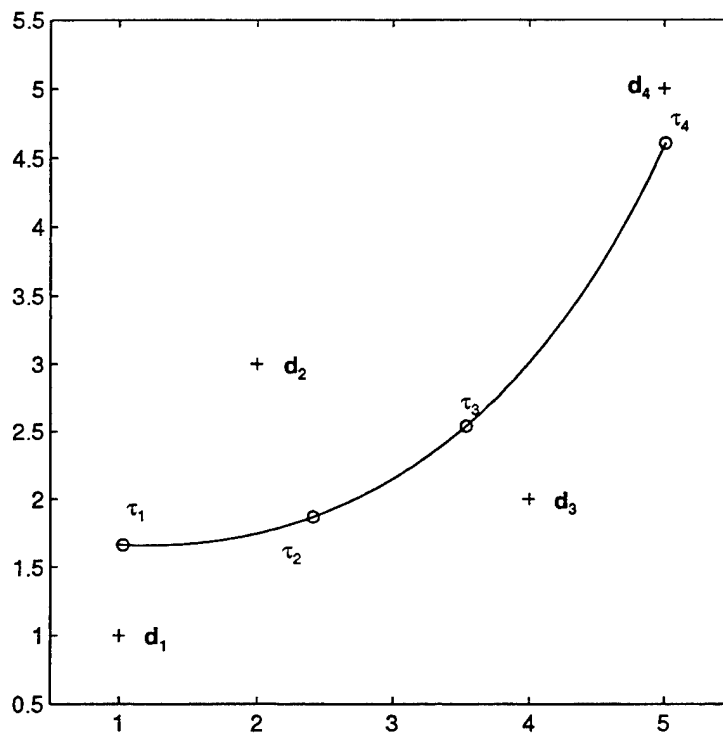


Figure 4. Solution for the Nearest Points

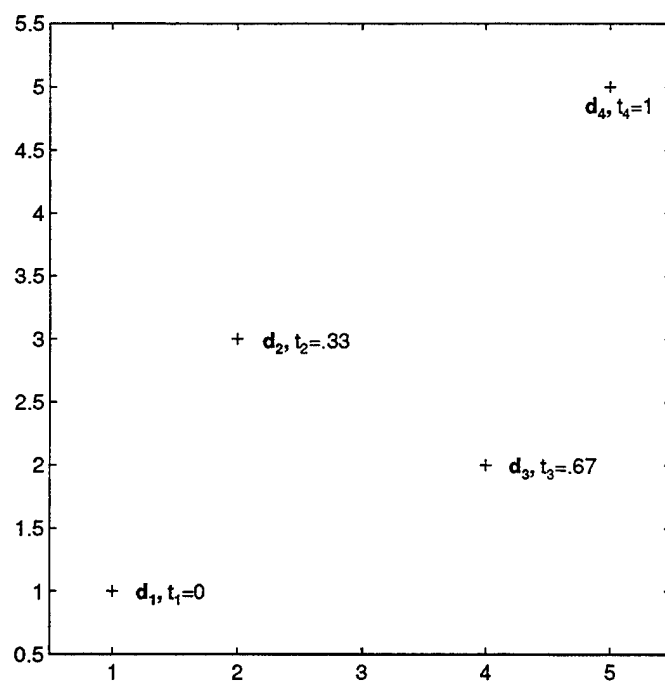


Figure 5. Data and knots for Linear Least Squares Problem

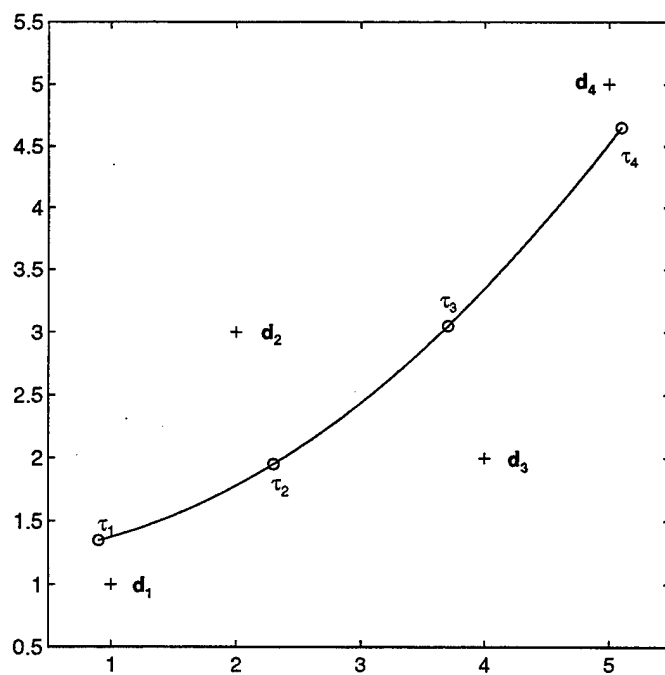


Figure 6. Solution to Linear Least Squares Problem

III. BASIC ALGORITHM

This chapter presents the basic algorithm used in this paper to determine $\{P^*, t^*\}$ and also presents three methods used to solve the nearest point problem. Finally, the primary algorithm to determine $\{P^*, t^*\}$ is presented.

A. BASIC ALGORITHM

The basic algorithm used in this paper to determine $\{P^*, t^*\}$ makes use of the separability of minimizing Equation (II.1) and follows an *alternating projection* approach. The basic steps are as follows:

1. Determine the initial set of nodes.
2. Solve the linear least squares parametric functional problem for the control points.
3. Solve the nonlinear least squares problem for the nearest points.
4. Repeat steps two and three until the algorithm reaches the stopping criteria.

The difference between the three MATLAB functions used in researching this paper is the method each uses to accomplish step three of the basic algorithm.

1. Approaching $\{P^*, t^*\}$

Let us look at the idea behind the basic algorithm as a solution technique for determining $\{P^*, t^*\}$. Given an ordered set of data and an initial vector of nodes t_1 , after solving the least squares problem for the matrix P_1 we would have

$$\| B(t_1)P_1 - D \|_F = \sigma_1 \quad (\text{III.1})$$

Note that, in general, the Bézier curve will not pass exactly through all the data points.

As we saw in Figure (6) on page 19 the initial set of points τ will most likely not be the nearest points, so some improvement to t_1 is possible. Solving the nearest

point problem for \mathbf{t}_2 gives

$$\| B(\mathbf{t}_2)P_1 - D \|_F = \sigma_2 < \sigma_1$$

Now we have an improved set of nodes but we have not improved the initial Bézier curve as described by P_1 . If there is a better fitting Bézier curve using the vector of nodes \mathbf{t}_2 then solving the least squares problem for P_2 will result in

$$\| B(\mathbf{t}_2)P_2 - D \|_F = \sigma_3 < \sigma_2$$

In this manner, we approach $\{P^*, \mathbf{t}^*\}$. The above argument is not a proof of convergence.

B. THE NEARESTPOINT METHOD

Given a matrix P , the *NearestPoint* method is a code found in [Ref. 5] which determines the nearest point associated with each data point \mathbf{d}_i by numerically solving for the roots of

$$\left[B(t_i)P - \mathbf{d}_i^T \right] \cdot \frac{d}{dt_i} B(t_i)P = 0$$

For example, for a degree 3 Bézier curve, the left-hand side of the above equation is a degree 5 polynomial. A subroutine called *FindRoots* returns the real roots of the resulting polynomial which, after ensuring they are within the interval $[0, 1]$, are used to determine points on the curve. The distance from the data point \mathbf{d}_i to each determined point on the curve along with the distance from the data point to each endpoint of the curve is compared, and the shortest distance indicates the parameter value of the nearest point. Because this is a numerical method, there is some degree of error in the solution.

There are two notable differences between this method and the two other methods used to accomplish step three of the basic algorithm. First, whereas the *NearestPoint* code considers each nearest point separately, the linear algebra representation of Bézier curves is used by the two other methods to consider all the nearest

points at once. Second, note again that the basic algorithm is an alternating projection approach where a new curve is produced with each new matrix P . In other words, the problem changes each time step two of the basic algorithm is performed. So, when accomplishing step three of the basic algorithm the two other methods do not continue to iterate to reach a numerical solution to the nearest point problem. Instead, the two other methods only move one step in the direction of the nearest points.

C. THE GRADIENT METHOD

The *gradient* method resulted from seeing the nonlinear least squares problem for the nearest points as an optimization problem requiring a gradient search technique [Ref. 6: 220]. From an initial set of nodes, we want to change each value so that the points τ approach the nearest points. However, instead of holding to the formal gradient search method we proceed as follows.

Recall that in Chapter II we showed that the nearest point on a Bézier curve from a given data point d_i will minimize

$$\| \tau_i - d_i \|_F \quad (\text{III.2})$$

and satisfy the necessary condition

$$\begin{bmatrix} \frac{d}{dt_1} \tau_{1,x} & \frac{d}{dt_1} \tau_{1,y} \end{bmatrix} \begin{bmatrix} \tau_{1,x} - d_{1,x} \\ \tau_{1,y} - d_{1,y} \end{bmatrix} = 0 \quad (\text{III.3})$$

Therefore, if we have an initial point τ_i in the neighborhood of the nearest point, then by finding the point τ_i which satisfies Equation (III.3) we find the nearest point.

Recall that the cosine of the angle between two vectors \mathbf{a} and \mathbf{b} is

$$\cos \theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\| \mathbf{a} \| \| \mathbf{b} \|}$$

Let $\mathbf{a} \equiv \frac{d}{dt} \tau_i$ and $\mathbf{b} \equiv \tau_i - d_i$. Given a point τ_i , which is in the neighborhood of the nearest point, we want to change the node t_i so that $\mathbf{a} \cdot \mathbf{b}$ approaches zero. Note that

the closer we get to satisfying Equation (III.3) the smaller the magnitude of $\mathbf{a} \cdot \mathbf{b}$. Therefore, we use $\mathbf{a} \cdot \mathbf{b}$ to not only tell us what direction to move the node t_i but also, it was initially thought, to give sufficient indication of how far to move.

Consider the following example. Figure (7) on page 30 shows four ordered data points and a Bézier curve fitted to the data points. Since $\theta < \frac{\pi}{2}$ we have that $\mathbf{a} \cdot \mathbf{b} > 0$. This tells us to move point τ_3 to the left, which corresponds to decreasing the value t_3 . We change t_3 with

$$\hat{t}_3 = t_3 - \Delta t_3 \quad (\text{III.4})$$

where Δt_3 is a scalar multiple of $\mathbf{a} \cdot \mathbf{b}$. This iterative process is likened to bracketing the nearest point.

1. Results of the Gradient Method

The first problem with the method as presented is that Equation (III.3) is not a sufficient condition for minimizing Equation (III.2). Starting out with an initial set of points τ in the neighborhood of the nearest points may overcome this problem in most cases, but it would still require an added check in any algorithm.

Secondly, when we fit higher degree Bézier curves to data the nearest point problem becomes nonlinear and in these cases this method as presented is insufficient to consistently move the points τ closer to the nearest points. The problem of determining the correct step size to take in the direction of the gradient requires more effort than relying on a user defined value of some scalar multiplying $\mathbf{a} \cdot \mathbf{b}$. Instead of attempting a possibly involved search technique to overcome this obstacle, a better solution technique for the nearest points was sought.

D. THE GAUSS-NEWTON METHOD

For a given matrix P , the *Gauss-Newton* method is a better way to solve the nonlinear least squares problem for the nearest points.

1. Review of Newton's Method

Under appropriate conditions on the function $f : t \rightarrow f(t)$, Newton's method is known to converge quadratically for initial estimates in the neighborhood of the roots. The idea behind Newton's method is that near a root we can model the behavior of the function with the following:

$$M_c(t) = f(t_c) + f'(t_c)(t - t_c) \quad (\text{III.5})$$

where $M_c(t)$ is a *linear approximation* of the function f and t_c is an estimate to a root of f . The value t such that $M_c(t) = 0$ is an improved approximation. This improved approximation is then used as the next estimate. So, we see that Newton's method is iterative with each iteration, or step, bringing the estimate closer to a root of f . This is also the idea behind the Gauss-Newton method for solving the nonlinear least squares problem.

2. Solving the Nonlinear Least Squares Problem

For a detailed discussion on the Gauss-Newton method, see [Ref. 7]. Let control point matrix $P \in \mathcal{R}^{m \times 2}$ be known and that we want to solve the following:

$$B(t)P - D = 0 \quad (\text{III.6})$$

where 0 is an $m \times 2$ matrix of zeros. To change Equation (III.6) to a form solvable using the Gauss-Newton method, we need to uncouple the left hand side as

$$\begin{bmatrix} B_0^n(t_1) & B_1^n(t_1) & \cdots & B_n^n(t_1) & 0 & 0 & \cdots & 0 \\ B_0^n(t_2) & B_1^n(t_2) & \cdots & B_n^n(t_2) & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ B_0^n(t_m) & B_1^n(t_m) & \cdots & B_n^n(t_m) & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & B_0^n(t_1) & B_1^n(t_1) & \cdots & B_n^n(t_1) \\ 0 & 0 & \cdots & 0 & B_0^n(t_2) & B_1^n(t_2) & \cdots & B_n^n(t_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & B_0^n(t_m) & B_1^n(t_m) & \cdots & B_n^n(t_m) \end{bmatrix} \begin{bmatrix} b_{0,x} \\ b_{1,x} \\ \vdots \\ b_{n,x} \\ b_{0,y} \\ b_{1,y} \\ \vdots \\ b_{n,y} \end{bmatrix} - \begin{bmatrix} d_{1,x} \\ d_{2,x} \\ \vdots \\ d_{m,x} \\ d_{1,y} \\ d_{2,y} \\ \vdots \\ d_{m,y} \end{bmatrix}$$

Let the above expression be the residual $R(\mathbf{t})$. We see that $R(\mathbf{t})$ is a system of $2m$ polynomials and, therefore, minimizing $R(\mathbf{t})$ is in some sense similar to root solving.

The Gauss-Newton method proceeds as follows. As in Newton's method, we first model the behavior of $R(\mathbf{t})$ with

$$M_c(\mathbf{t}) = R(\mathbf{t}_c) + \left[\frac{d}{d\mathbf{t}} R(\mathbf{t}_c) \right] (\mathbf{t} - \mathbf{t}_c)$$

which is analogous to Equation (III.5). We will use the term *Jacobian* to describe the derivative of a vector function. Therefore, our Jacobian matrix is

$$J(\mathbf{t}_c) = \begin{bmatrix} \frac{dR(t_{1,c})_x}{dt_1} & \dots & \frac{dR(t_{1,c})_x}{dt_m} \\ \vdots & & \vdots \\ \frac{dR(t_{m,c})_x}{dt_1} & \dots & \frac{dR(t_{m,c})_x}{dt_m} \\ \frac{dR(t_{1,c})_y}{dt_1} & \dots & \frac{dR(t_{1,c})_y}{dt_m} \\ \vdots & & \vdots \\ \frac{dR(t_{m,c})_y}{dt_1} & \dots & \frac{dR(t_{m,c})_y}{dt_m} \end{bmatrix}$$

Note that

$$\frac{dR(t_{i,c})_x}{dt_j} = 0 \text{ for } i \neq j$$

which also holds for the y components. Therefore, our Jacobian matrix has the following special form

$$J(\mathbf{t}_c) = \begin{bmatrix} \frac{dR(t_{1,c})_x}{dt_1} & 0 & \dots & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \dots & 0 & \frac{dR(t_{m,c})_x}{dt_m} \\ \frac{dR(t_{1,c})_y}{dt_1} & 0 & \dots & 0 \\ 0 & \ddots & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \dots & 0 & \frac{dR(t_{m,c})_y}{dt_m} \end{bmatrix} \quad (\text{III.7})$$

To get an improved estimate we want to solve for \mathbf{t} so that

$$\begin{aligned} M_c(\mathbf{t}) &= R(\mathbf{t}_c) + J(\mathbf{t}_c)(\mathbf{t} - \mathbf{t}_c) \\ &= \mathbf{0} \end{aligned}$$

or, equivalently

$$J(\mathbf{t}_c)\mathbf{t} = J(\mathbf{t}_c)\mathbf{t}_c - R(\mathbf{t}_c) \quad (\text{III.8})$$

where the right hand side is a known vector in \mathcal{R}^{2m} . Note that Equation (III.8) is a linear least squares problem for the vector \mathbf{t} . Formally, we proceed by forming the normal equations

$$J^T(\mathbf{t}_c)J(\mathbf{t}_c)\mathbf{t} = J^T(\mathbf{t}_c)J(\mathbf{t}_c)\mathbf{t}_c - J^T(\mathbf{t}_c)R(\mathbf{t}_c)$$

where $J^T(\mathbf{t}_c)J(\mathbf{t}_c)$ is a square matrix and invertible. Though one could argue for a more numerically stable method to solve this least squares problem, in our particular case the normal equations lead to a simple expression for the change to make in our nodes and to results which are favorable when using the alternating projection approach. So, after multiplying both sides by $(J^T(\mathbf{t}_c)J(\mathbf{t}_c))^{-1}$ we have

$$\mathbf{t} = \mathbf{t}_c - (J^T(\mathbf{t}_c)J(\mathbf{t}_c))^{-1} J^T(\mathbf{t}_c)R(\mathbf{t}_c)$$

Therefore, our improved estimate $\hat{\mathbf{t}}_c$ is given by

$$\begin{aligned} \hat{\mathbf{t}}_c &= \mathbf{t}_c - (J^T(\mathbf{t}_c)J(\mathbf{t}_c))^{-1} J^T(\mathbf{t}_c)R(\mathbf{t}_c) \\ &= \mathbf{t}_c - \Delta(\mathbf{t}_c) \end{aligned} \quad (\text{III.9})$$

Our matrix $J(\mathbf{t}_c)$ from Equation (III.7) has a special form so that the right hand side of Equation (III.9) simplifies. First, note that $J^T(\mathbf{t}_c)J(\mathbf{t}_c)$ is a diagonal matrix with elements on the main diagonal of the form

$$\left(\frac{dR(t_{i,c})_x}{dt_i} \right)^2 + \left(\frac{dR(t_{i,c})_y}{dt_i} \right)^2 \text{ for } i = 1, \dots, m \quad (\text{III.10})$$

Therefore, $(J^T(\mathbf{t}_c)J(\mathbf{t}_c))^{-1}$ is another diagonal matrix with elements which are the reciprocals of those described by Equation (III.10). The term $J^T(\mathbf{t}_c)R(\mathbf{t}_c)$ is a vector in \mathcal{R}^m with elements of the form

$$\left[R(t_{i,c})_x \frac{dR(t_{i,c})_x}{dt_i} + R(t_{i,c})_y \frac{dR(t_{i,c})_y}{dt_i} \right] \quad (\text{III.11})$$

Combining the results from Equation (III.10) and Equation (III.11), we have the following expression for each element of $\Delta(\mathbf{t}_c)$,

$$\left(\left(\frac{dR(t_i)_x}{dt_i} \right)^2 + \left(\frac{dR(t_i)_y}{dt_i} \right)^2 \right)^{-1} \left[R(t_{i,c})_x \frac{dR(t_{i,c})_x}{dt_i} + R(t_{i,c})_y \frac{dR(t_{i,c})_y}{dt_i} \right] \quad (\text{III.12})$$

Equation (III.12) is easy to program into MATLAB and very inexpensive. Note that $\hat{\mathbf{t}}_c$ is one step in the direction of the nodes which will minimize $R(\mathbf{t})$. That is, $\hat{\mathbf{t}}_c$ is one step in the direction of the nearest points.

The effectiveness of the Gauss-Newton method depends in part on our initial points $\boldsymbol{\tau}$ being in the neighborhood of the nearest points. Just like the Newton method, a poor initial estimate could cause the method to fail. Therefore, we use the affine invariant angle method in our primary solution algorithm to get the initial set of nodes.

3. Stopping Criteria

Because relative error is a more meaningful indicator of change for larger values, the stopping criteria used in this paper to determine $\{P^*, \mathbf{t}^*\}$ is determined by the value of $\| R(\mathbf{t}_{j+1}, P_{j+1}) \|_2$. When the 2 norm of this residual is greater than one, the stopping criteria is the relative error. When the 2 norm of the residual is less than or equal to one, the stopping criteria is the absolute error expressed as

$$\| R(\mathbf{t}_{j+1}, P_{j+1}) - R(\mathbf{t}_j, P_j) \|_2$$

where

$$R(\mathbf{t}_j, P_j) = B(\mathbf{t}_j)P_j - D$$

is the residual of iteration j . Note that the residual depends on the nodes and the control points. When either measure of error gets below a user defined value, the function stops improving the fit of the curve.

We could also use the absolute error

$$\| \mathbf{t}_{j+1} - \mathbf{t}_j \|_2$$

However, a small change in our estimates does not necessarily mean we are close to the nearest points. It is the *curve* we are fitting to the data and not the vector \mathbf{t} , so it is best to use the curve itself to indicate when we are close enough to determining $\{P^*, \mathbf{t}^*\}$.

E. PRIMARY ALGORITHM

The following is the primary algorithm this paper uses to determine $\{P^*, \mathbf{t}^*\}$. It is presented to aid in the understanding of the MATLAB function *grad7.m*. The MATLAB functions included below are provided in the Appendix.

Step one. The user sends *grad7.m* the data matrix $D \in \mathcal{R}^{m \times 2}$ in the sequence he wants the data ordered, the degree Bézier curve to fit to the data, and an exponent value which determines the stopping criteria. The function *aff_angle.m* is called as a subroutine and returns the initial set of nodes \mathbf{t}_1 using the affine invariant angle method. The function *mxbern2.m* is called as a subroutine and returns the Bernstein matrix $B(\mathbf{t}_1)$. Then, the linear least squares parametric functional problem $B(\mathbf{t}_1)P_1 = D$ is solved for the matrix P_1 of control points. Finally, the residual $R(\mathbf{t}_1, P_1)$ is calculated.

Step two. Begin the *while loop*. The stopping criteria is checked, and, if met, we exit the *while loop*. Otherwise, the derivative of $R(\mathbf{t}_i, P_i)$, which is the derivative of $B(\mathbf{t}_i)P_i$, is determined as presented in Chapter I again using *mxbern2.m*. We now use Equation (III.12) to obtain the improved estimate, where $\mathbf{t}_{i+1} = \hat{\mathbf{t}}_i$. Thus, we take only one step in the direction of the nearest points. Once we have the improved estimate we ensure its elements are within the interval $[0, 1]$.

Step three. Solve the least squares problem for the matrix P_{i+1} . The residual $R(\mathbf{t}_{i+1}, P_{i+1})$ is calculated. Return to step two. End of the *while loop*.

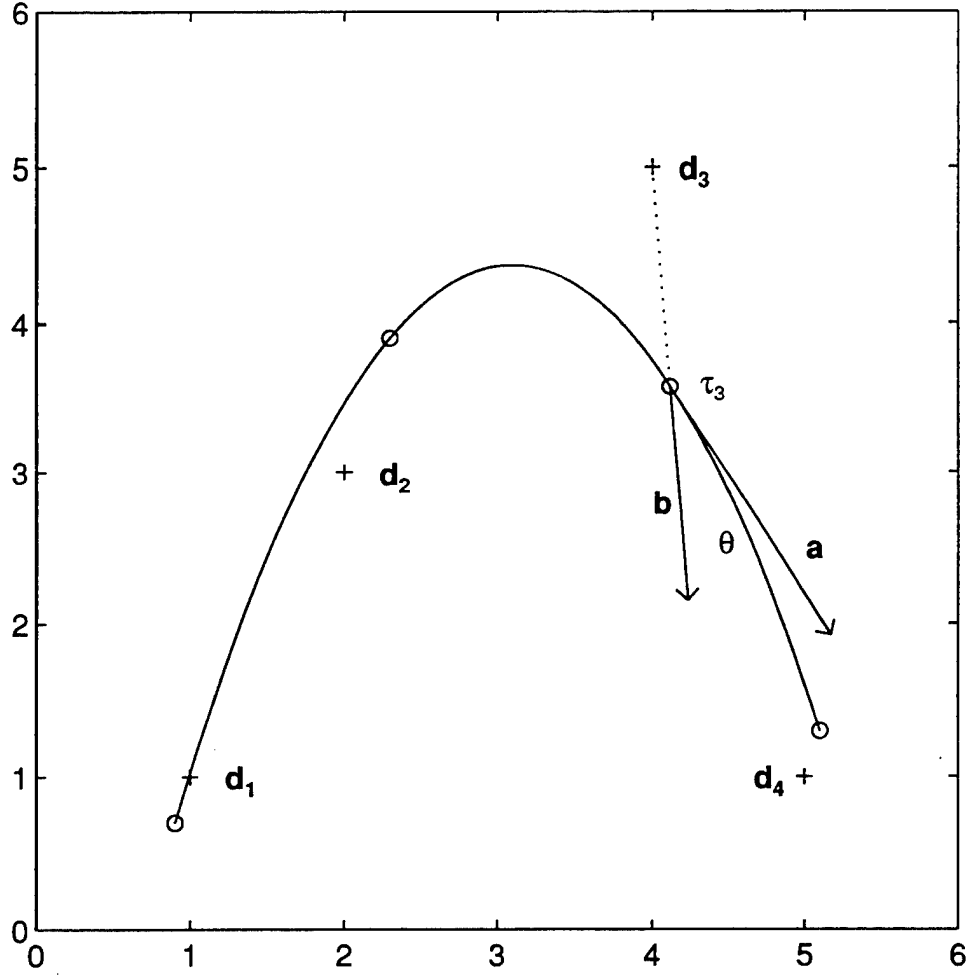


Figure 7. Moving Nodes with Gradient Method

IV. METHOD EXAMPLES

In this chapter we will consider graphic examples of the different MATLAB functions used while researching this paper to determine $\{P^*, t^*\}$. The difference between the functions is how each handles the nonlinear least squares problem for the nearest points. The MATLAB functions examined in this chapter are:

1. *grad3.m*. Uses the gradient method.
2. *grad5.m*. Uses the *NearestPoint* method.
3. *grad7.m*. Uses the Gauss-Newton method.

A. GRAPHICAL COMPARISONS

This section examines graphical examples of the three MATLAB functions above and also the graphical difference between the two affine invariant node spacing methods presented in Chapter II.

1. Gradient Method and Approaching $\{P^*, t^*\}$

As stated in Chapter III, the gradient method as developed for this paper was insufficient to consistently move the points τ closer to the nearest points. Figure (8) and Figure (9) on page 37 show that the algorithm reaches a stage where the points τ cycle back and forth along the curve. The two curves are similar, but we are no longer approaching a better fit. By the third iteration the control points cycle back and forth between

$$P_1 = \begin{bmatrix} 0.8297 & 0.8911 \\ 6.0947 & 9.5031 \\ 5.0183 & 1.0117 \end{bmatrix}$$

and

$$P_2 = \begin{bmatrix} 0.7802 & 0.7811 \\ 5.9106 & 9.6305 \\ 4.9913 & 0.9784 \end{bmatrix}$$

The nodes cycle back and forth between

$$\mathbf{t}_1 = \begin{bmatrix} 0.0000 \\ 0.1626 \\ 0.3566 \\ 1.0000 \end{bmatrix}$$

and

$$\mathbf{t}_2 = \begin{bmatrix} 0.0413 \\ 0.1011 \\ 0.4272 \\ 1.0000 \end{bmatrix}$$

Though either of the curves seem like a pretty good fit, an algorithm that approaches $\{P^*, \mathbf{t}^*\}$ is what any user expects and, therefore, this method is unacceptable.

2. Total Least Squares Versus Linear Least Squares

We might ask what the difference is between a fitted curve produced by traditional linear least squares and a fitted curve produced by the total least squares function *grad7.m*. Figure (10) and Figure (11) on page 38 are both cubic curves fitted to the same data set but using these two different approaches.

We notice that each curve makes a different assumption about the behavior of the data about the point \mathbf{d}_1 . Most important to the user is that the linear least squares fit is only assuming error in the y values, while *grad7.m* is minimizing error in the x and the y values. This is more practical since the input, or the x values, will also often contain some degree of error.

3. Effect of the Initial Set of Nodes

We will now see how the initial set of nodes may determine the fit of the approximating curve. Figure (12) and Figure (13) on page 39 show the Bézier curves returned by *grad7.m* using two different affine invariant node spacing methods. Figure (12) reflects the affine invariant chord method while Figure (13) reflects the affine invariant angle method. Both figures show the nodes associated with the data points.

Because the affine invariant angle method takes into account the bending of the data, the approximating curve fits more naturally. This is opposed to what we have in Figure (12) where it looks like the curve is alternately stretched and slack.

4. Failure to Maintain Ordering of Data

Figure (14) and Figure (15) on page 40 are two approximating curves for the same set of data. Both curves were generated by *grad7.m* with stopping criteria of 10^{-1} . Figure (14) is the result of using the affine invariant chord method to obtain the initial set of nodes and Figure (15) is the result of using the affine invariant angle method.

In the same figures, notice the difference in interpolating about the point d_4 . When the order of the data is not maintained the user gets an entirely different picture of the trend in the data. In general, whether the methods presented in this paper maintain the ordering of data is dependent on the initial set of nodes and the order of the curve the user fits to the data. For example, using the same data set as in Figure (15) and a degree 2 curve, the function *grad7.m*, using the affine invariant angle method, will fail to maintain the order of the data about d_4 .

Consider the function *grad5.m*. Because the *NearestPoint* method is free to look anywhere along the curve to solve the nearest point problem, *grad5.m* more often fails to maintain the ordering of data. This is opposed to the Gauss-Newton method which assumes that each initial point τ_i is in the neighborhood of the nearest point.

5. Gauss-Newton Method and Data with Noise

Consider a set of data that lies on a cubic curve and that the data is then transmitted to a receiver. In the transmission of the data a small amount of noise gets added. How well does *grad7.m* fit the data with the noise added, and how much like the original curve is the resulting fitted curve? Note, in the following examples all random sets of numbers were generated using the MATLAB *rand* command which returns a uniformly distributed set of numbers on the interval $[0, 1]$. Figure (16) on

page 41 shows a random sampling of 10 data points from a cubic Bézier curve which was generated from a random matrix P . Noise was added to each data point by generating a random set of points around the unit circle and scaling each point by .012. Figure (17) on the same page is the curve fitted by *grad7.m*.

This example shows one of the limitations of *grad7.m* even when using an effective initial node spacing method: *grad7.m* views the cluster of data points in the upper right hand side of the plot as just another set of points to fit with a curve. This is why we get the sharp bend within the cluster. The user who wants to avoid this type of behavior could, for example, substitute one data point for the entire cluster or use a weighted least squares approach to fitting the data.

Figure (18) on page 42 is the same set of data but now fitted with a degree 5 curve. This plot reflects what occurs when the fitted curve has too much freedom. A degree 5 curve has more freedom than needed for the data, and *grad7.m* makes use of all the freedom available in order to improve the fit.

6. Gauss-Newton and Real World Data

This section briefly presents a real world problem where the function *grad7.m* could be used. The data for this section comes from recording positions along a road leading to Fort Ord, California with a Global Positioning System (GPS) receiver. Many similar data sets are gathered using several different receivers along the same route and curves are then fitted to the data. The fitted curves are used to determine the bias present based on the particular satellites being used by the receivers. Imagine the road in the x - y plane. There will be error in both the x and y coordinates of each location in the data set, so we will want to fit a curve to the data in the total least squares sense. Figure (19) on page 43 is the data and the degree 3 curve fitted to the data using a stopping criteria of 10^{-4} .

B. GAUSS-NEWTON VERSUS NEARESTPOINT

We now look specifically at the performance of *grad7.m* and *grad5.m*. We will consider both graphical results and computer time as indicators of performance. Both functions get their initial set of nodes using the affine invariant angle method.

1. Graphical Results

We will consider cubic Bézier curves on which we select 11 evenly spaced points with respect to the parameter t and then add a small amount of random noise, as above, to each point. Each of the two functions fits a cubic Bézier curve to the resulting data set and the original and newly generated curve are displayed on one plot to compare how well each function performs. Figure (20) on page 44 to Figure (25) on page 46 are pairs of plots for three different data sets. The solid line is the curve returned by the functions.

Note that in each case the pair of plots is slightly different. Regardless, the overall graphical comparison of the two functions, after conducting many other examples not shown, is that they perform equally well except when *grad5.m* fails to maintain the ordering of data.

2. Computer Time and Iterations

Table I on page 36 reflects how much computer time was used for several sizes of data sets and stopping criteria of 10^{-4} . Each data set is a matrix of uniformly distributed random entries and ordered with respect to the x and y values. Because the *NearestPoint* method finds each new node separately, it is part of a *for loop* within *grad5.m*. As the data sets get larger, the function slows down in comparison to *grad7.m* where the form of the normal equations in the Gauss-Newton method remains relatively fast.

C. GENERAL COMPARISON OF FUNCTIONS

Table II on page 36 reflects the general results of researching this paper by comparing the three solution functions examined in this chapter.

Size	Time (sec) / Iter	
	<i>grad5.m</i>	<i>grad7.m</i>
10	3.0135 / 121	1.7405 / 126
100	10.8809 / 53	3.7208 / 91
1000	62.2660 / 36	.9655 / 3

Table I. Computer Time Used

Function	General Results
<i>grad3.m</i>	<ul style="list-style-type: none"> • Inconsistently indicates how large a step to take in the direction of the nearest point.
<i>grad5.m</i>	<ul style="list-style-type: none"> • Faster than <i>grad7.m</i> in rare cases. • Graphically, performs equally well as <i>grad7.m</i>. • Code is already provided but more difficult than <i>grad7.m</i> to examine and follow, see [Ref. 5]. • More readily fails to maintain the ordering of data. • More expensive to run on larger data sets and many smaller data sets as seen during research.
<i>grad7.m</i>	<ul style="list-style-type: none"> • Faster than <i>grad5.m</i> in most cases. • In the many examples conducted during research, has not failed to reach a reasonable stopping criteria. • Simple to code and algorithm is easy to follow.

Table II. General Comparison of Functions

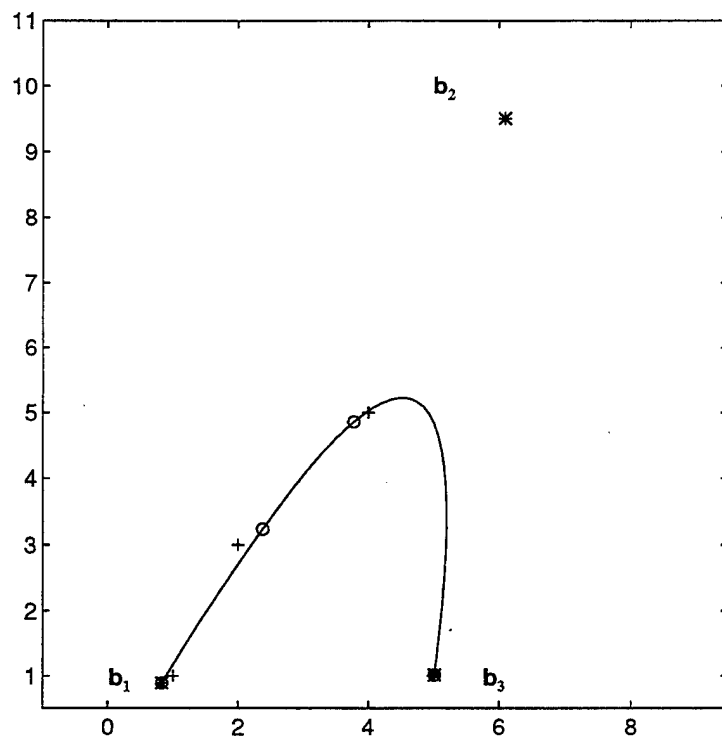


Figure 8. Gradient Method Fails to Converge

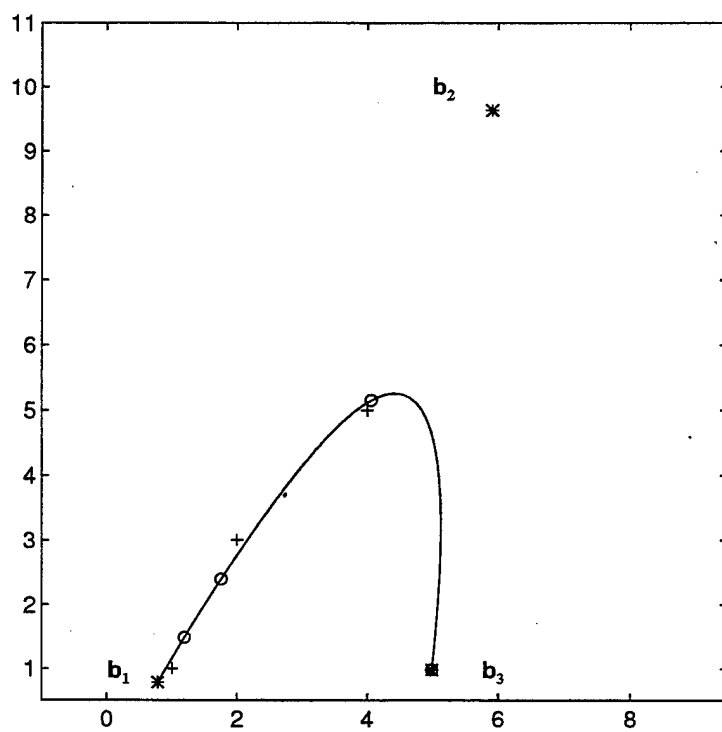


Figure 9. Gradient Method Fails to Converge

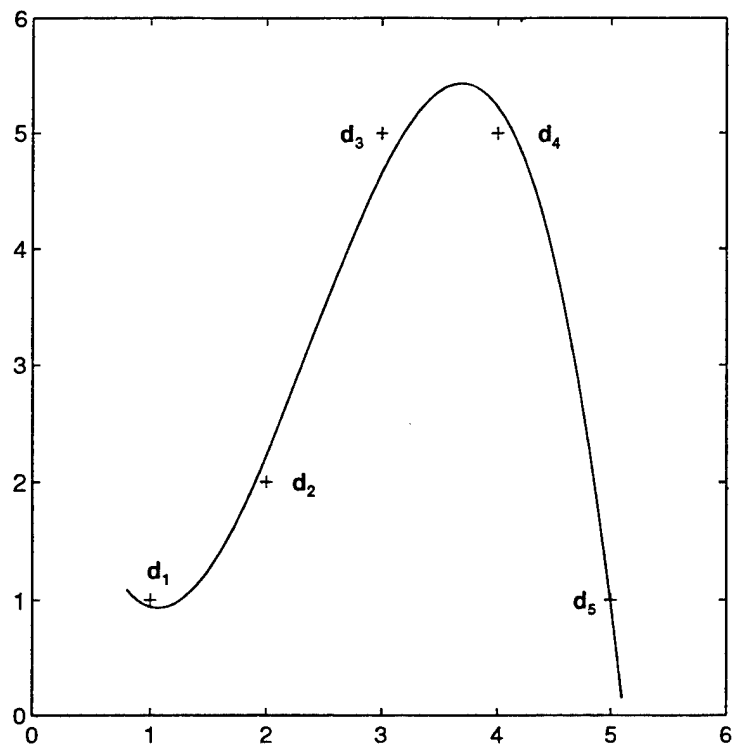


Figure 10. Linear Least Squares Fit

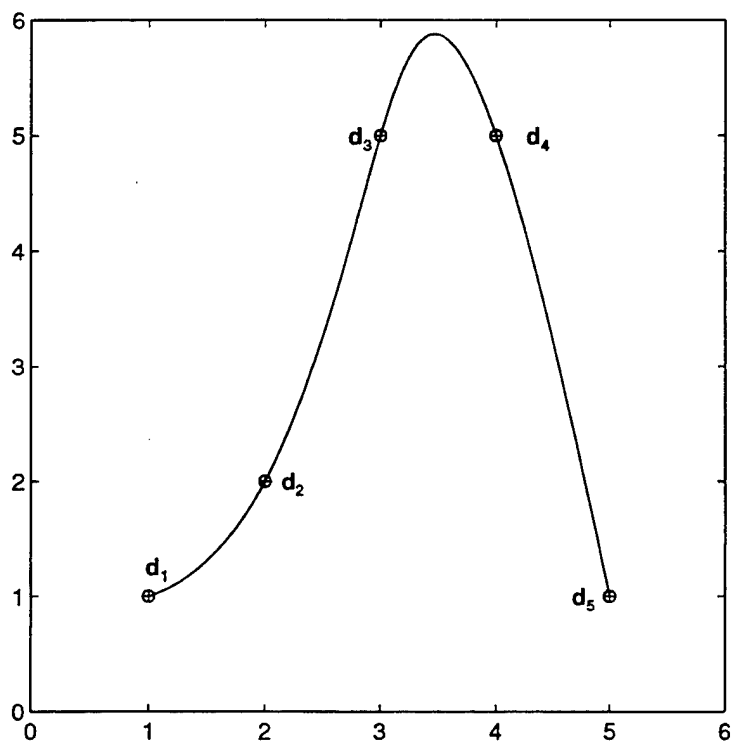


Figure 11. Gauss-Newton Fit

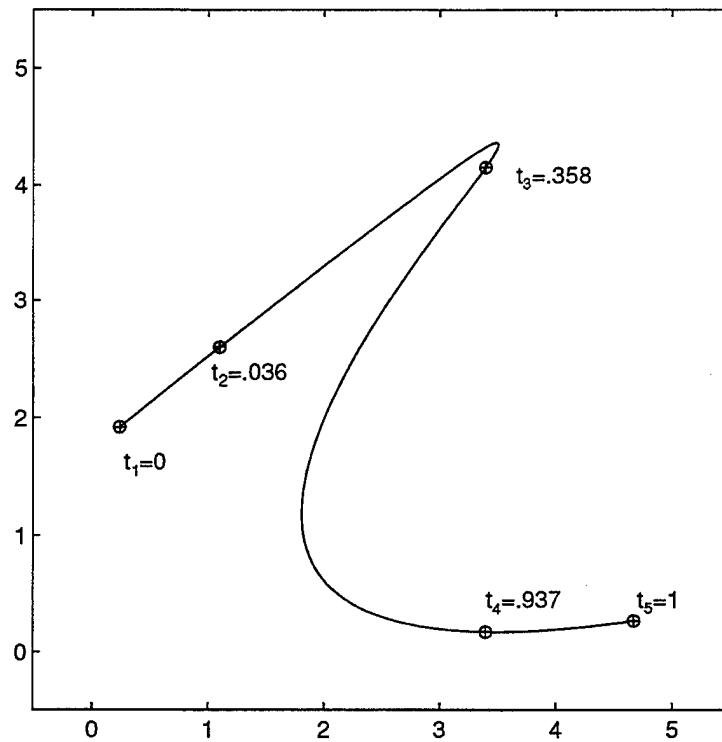


Figure 12. Affine Invariant Chord Method

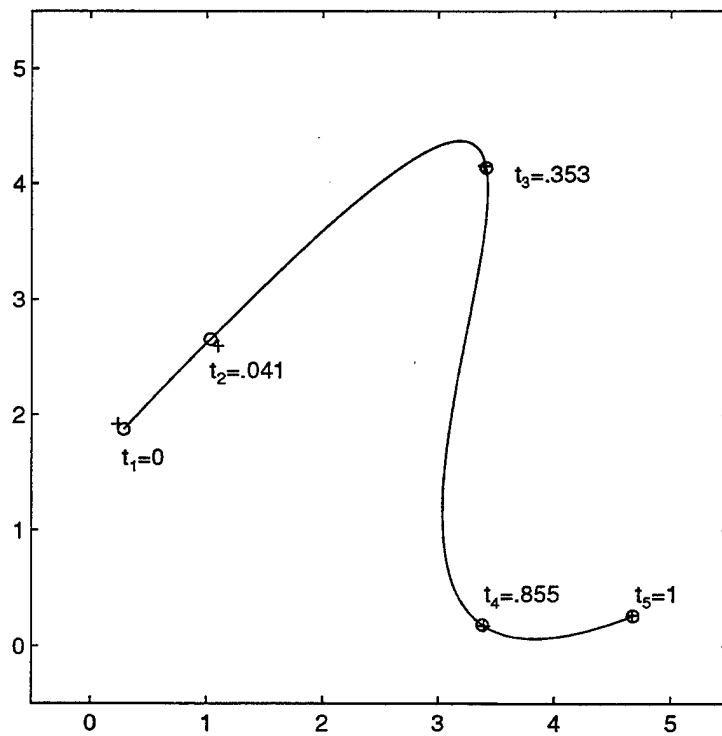


Figure 13. Affine Invariant Angle Method

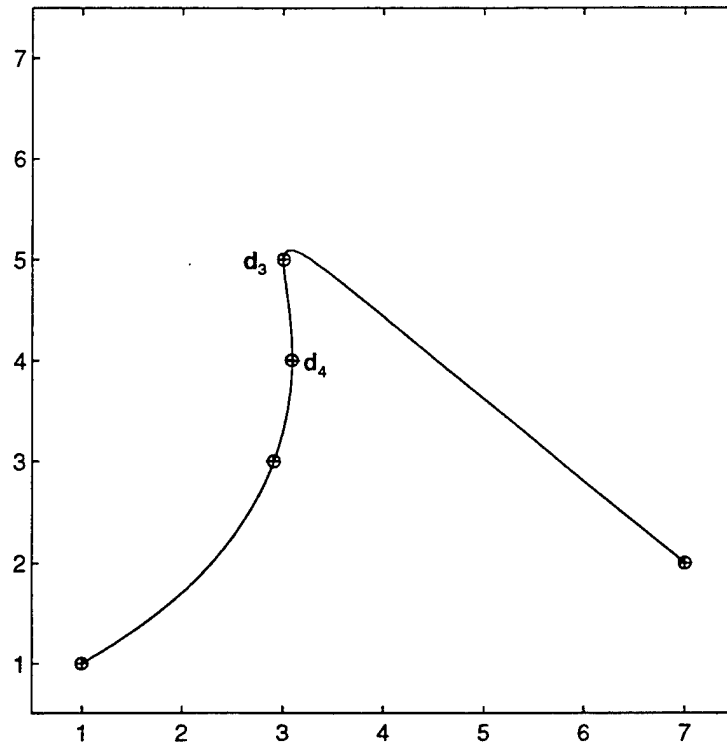


Figure 14. Failure to Maintain Order

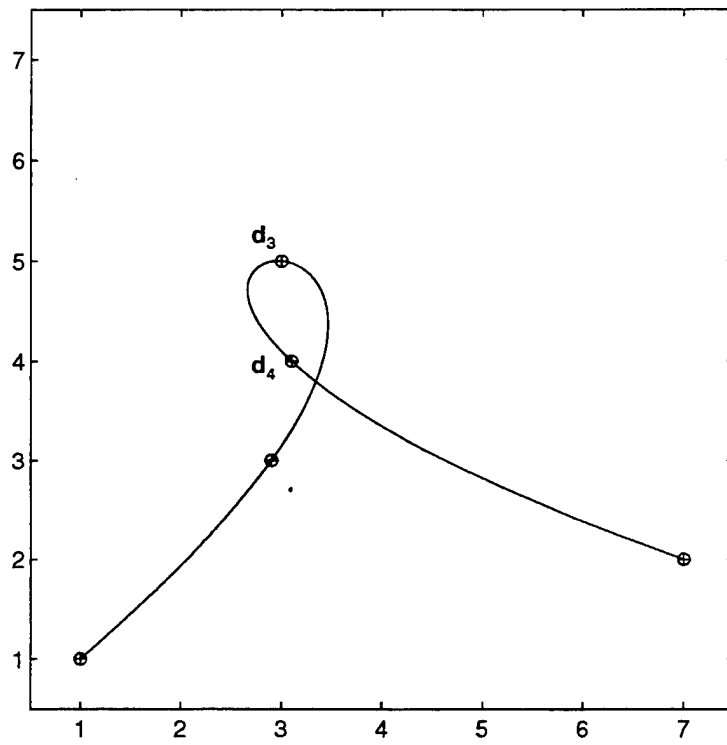


Figure 15. Affine Invariant Angle Maintains Order

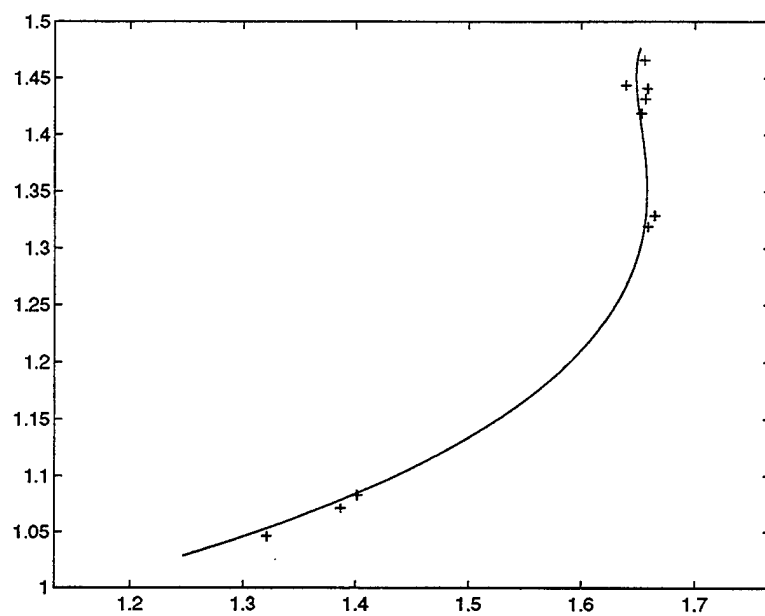


Figure 16. Data With Added Noise

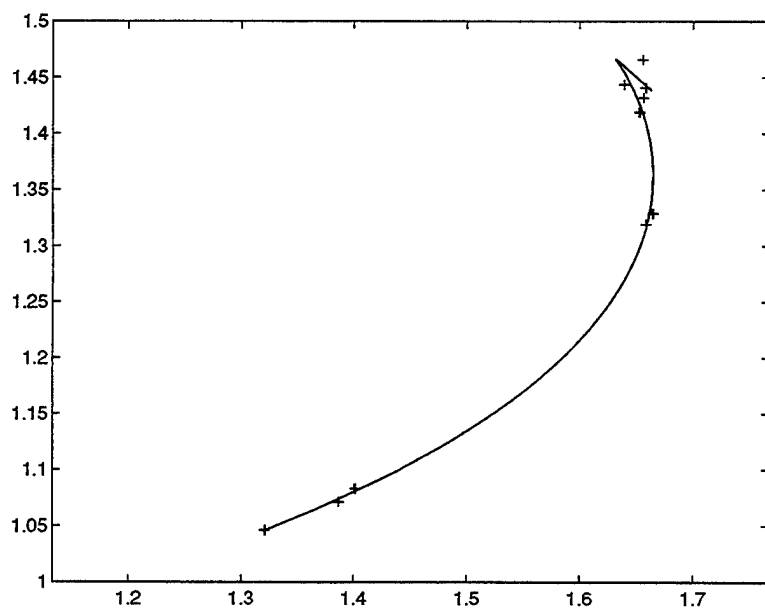


Figure 17. Fitting Data With Added Noise

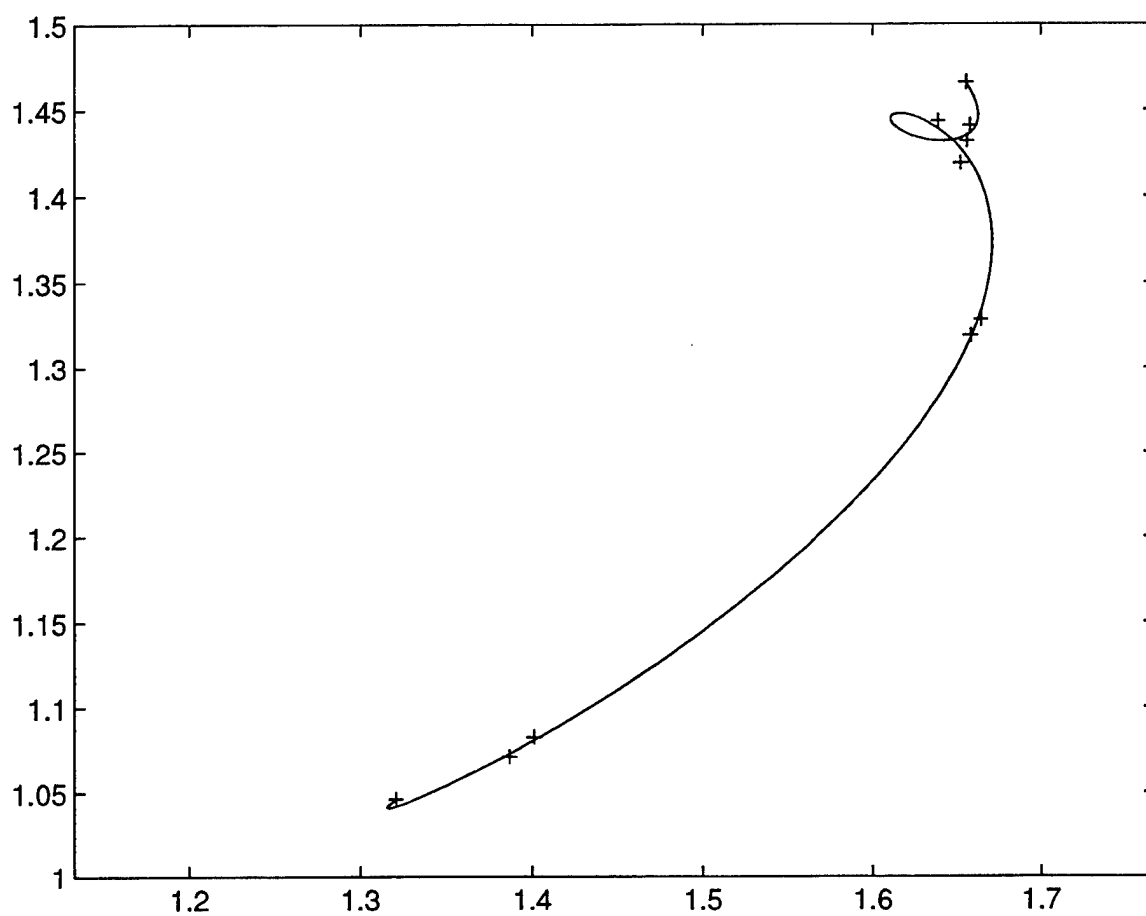


Figure 18. Curve With Too Much Freedom

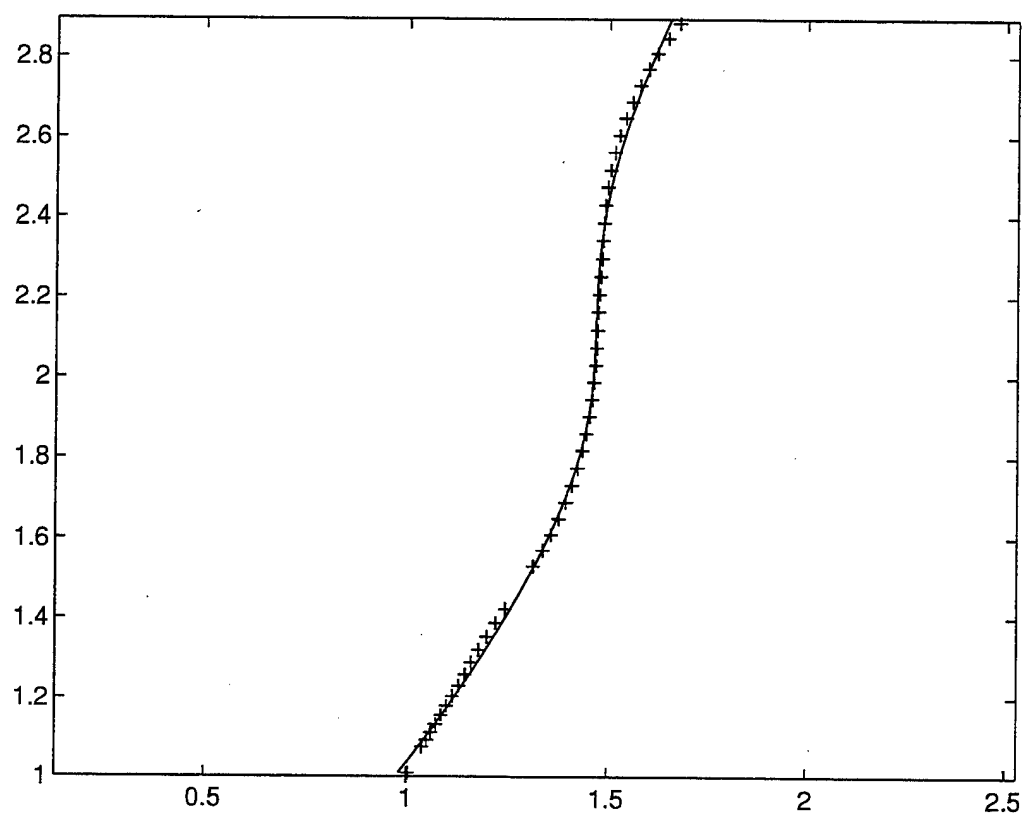


Figure 19. Real World Data and *grad7.m*

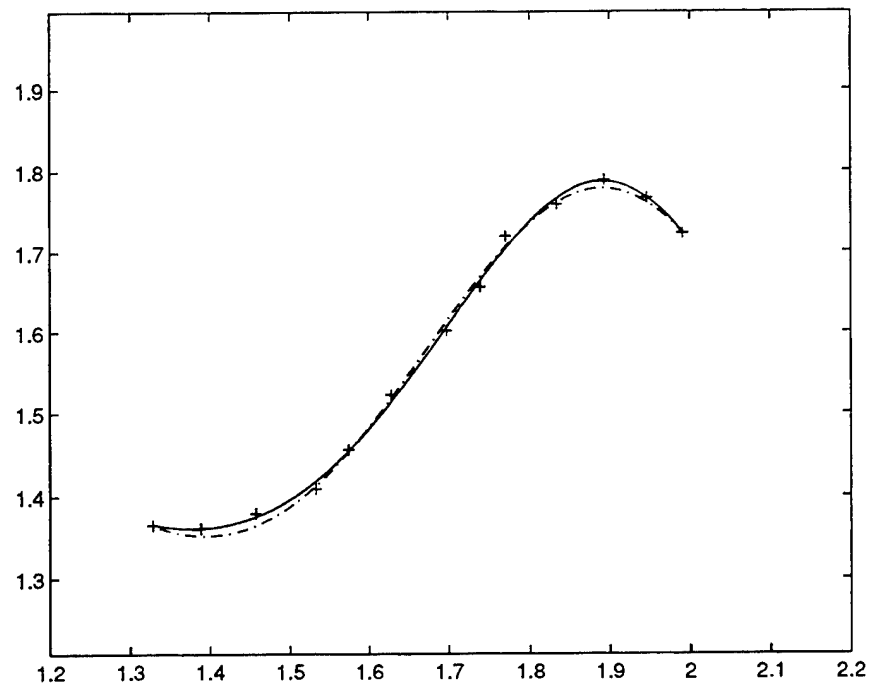


Figure 20. Data Set One and *grad7.m*

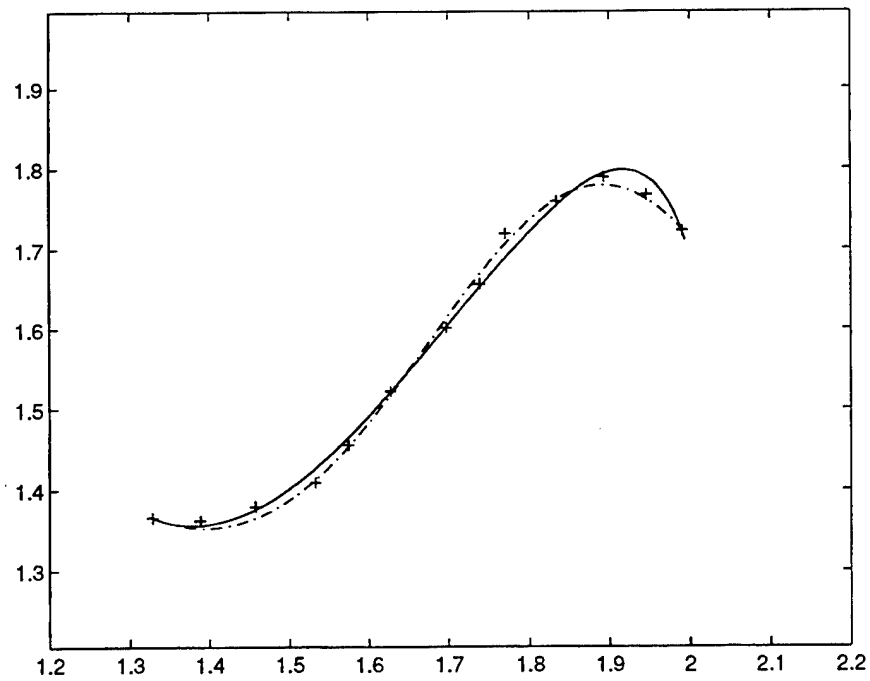


Figure 21. Data Set One and *grad5.m*

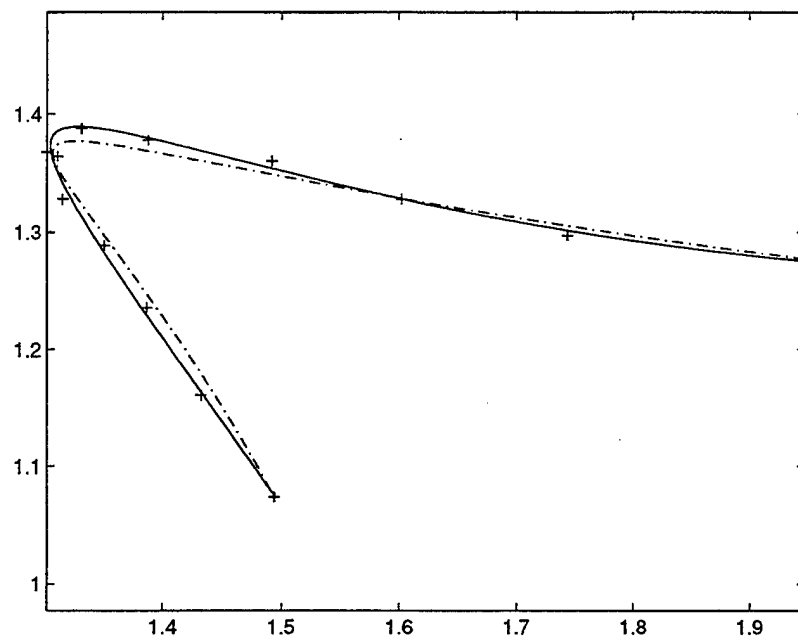


Figure 22. Data Set Two and *grad7.m*

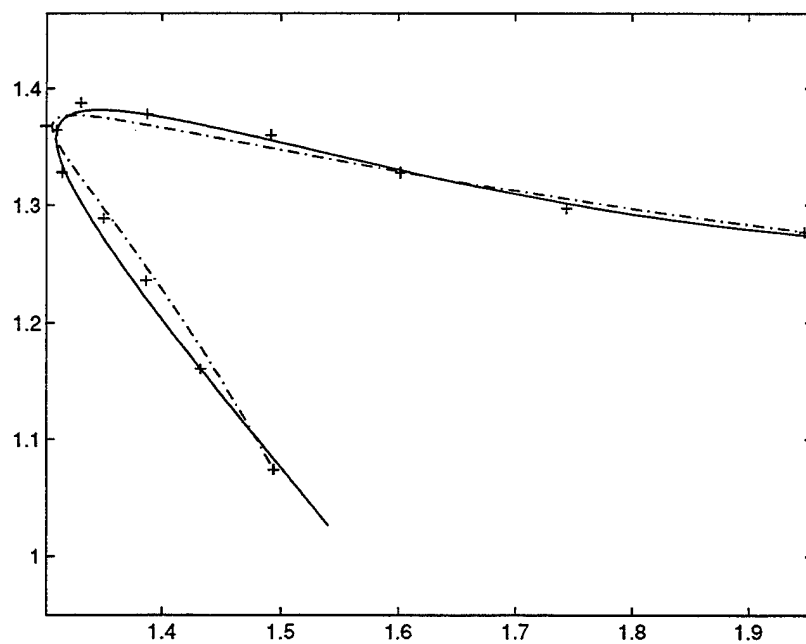


Figure 23. Data Set Two and *grad5.m*

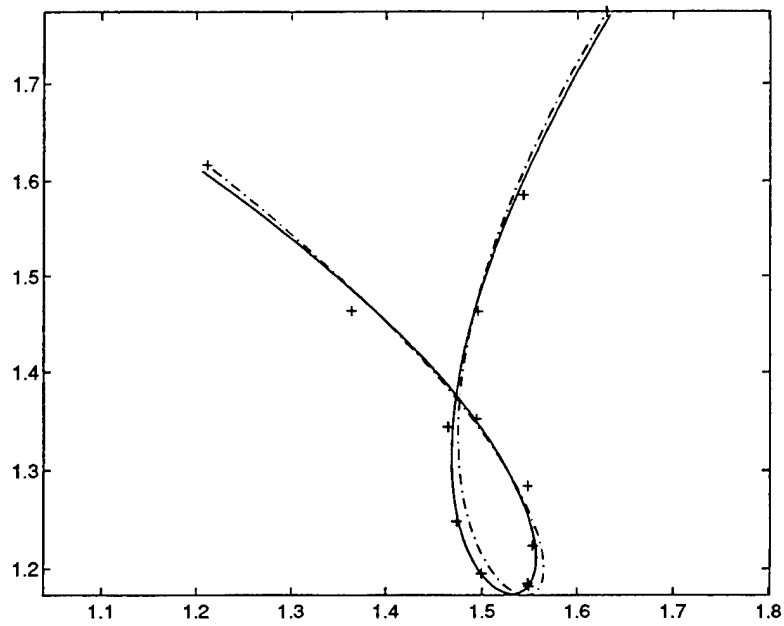


Figure 24. Data Set Three and *grad7.m*

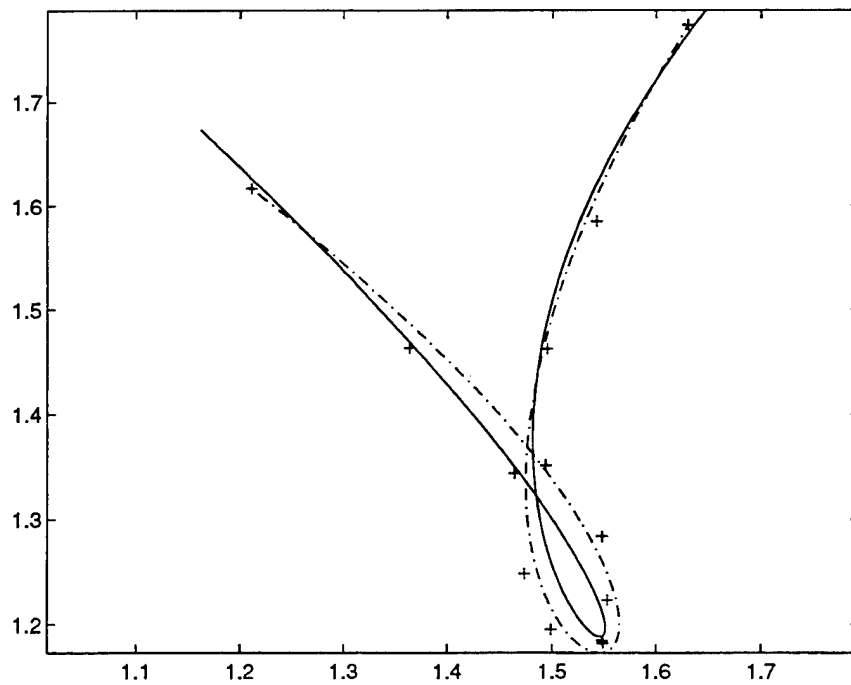


Figure 25. Data Set Three and *grad5.m*

APPENDIX. MATLAB FUNCTIONS

GRAD7.M

```
function [p,t,info]= grad7(d,deg,stop)
```

```
% GRAD7.M This function takes a given set of ordered data and returns
% the parameter values (nodes) and control points which determine the
% Bezier curve that fits the data in the total least squares sense.
% Instead of minimizing the vertical distance to the curve we minimize
% both vertical and horizontal distance. The central feature of this
% function is the use of the Gauss-Newton method to estimate the
% nearest point along the Bezier curve from each data point.
```

```
% Input Arguments:
```

```
%           d      (i x 2) matrix of ordered data points,
%                   i=2,3,...
%           deg     degree of the curve the user wants to fit
%                   to the data
%           stop    stopping criteria number which in the
%                   algorithm becomes 10^(stop)
```

```
% Output Arguments:
```

```
%           p      control points for best fit Bezier curve
%           t      nodes for best fit Bezier curve
%           info    (2 x 1) vector which has the final norm of
%                   the residual and the number of iterations
%                   to convergence
```

```
% Basic Algorithm:
```

```
%           1. Determine and plot the best fit Bezier curve by
%               solving a linear least squares problem using an
%               initial set of nodes based on the
%               'spread' and 'bend' of the data.
%           2. Perform the following until the stopping
%               criteria is met:
%               a. Determine a new set of nodes
%                   which estimates points on the current Bezier
%                   points.
```

```

%
%
%           b. Determine the control points for the Bezier
%           curve which will best fit the data using the
%           new set of nodes.

% ALGORITHM STEP 1.

% Check the hold state so it can be returned to how the user had it.

if (ishold)
    hold_was_off = 0;
else
    hold_was_off = 1;
end

% 'i' is the number of data points and 'j' is the number of control
% points required for the degree of the curve specified by the user.
% aff_angle(d) returns the initial set of nodes, a vector of 'i'
% elements, based on the 'spread' and 'bend' of the data.

i = size(d,1);
j = deg+1;
t = aff_angle(d);

% 'bez_mat' is a (i x j) matrix which is determined by the nodes
% and the degree of the curve desired. Since we would
% like to fit the data exactly, we should solve:
%
%           d = bez_mat * p
%
% for the desired (j x 2) matrix of control points 'p'. This is a
% linear least squares problem since the nodes are known. 'p'
% is determined by:
%
%           p = pinv(bez_mat) * d
%
% which is the same as using the matlab 'backslash' command.

bez_mat = mxbern2(t,deg);
p       = bez_mat \ d;

% Once we have the control points 'p', we can determine the Bezier

```

```

% curve and the points on the curve associated with the initial vector
% t. 'y' is the (i x 2) matrix of points on the Bezier curve
% associated with the initial vector t. 't1' is a closely spaced
% set of parameter values which will produce the Bezier matrix,
% 'bez_mat_1', which will give enough points on the fitted Bezier curve
% for matlab to plot a smooth looking curve: 'y1' is the matrix of
% these points.

```

```

y          = bez_mat * p;
t1         = [0:1/128:1]';
bez_mat_1  = mxbern2(t1,deg);
y1         = bez_mat_1 * p;

```

```

% Now we plot the results for the user. A legend is provided and the
% axes are made 'equal' to eliminate distortions.
% 'Pause' will keep the plot displayed and delay this function until
% the user presses any key on the keyboard.

```

```

figure
plot(y1(:,1),y1(:,2))
hold on
plot(p(:,1),p(:,2),'*')
plot(y(:,1),y(:,2),'o')
plot(d(:,1),d(:,2),'+')
axis('equal')
legend('-', 'Fitted Curve', '*', 'Control Points', ...
       'o', 'Initial Times', '+', 'Data Points')
pause

```

```

% ALGORITHM STEP 2.

```

```

% We will perform steps 2a and 2b within a 'while' loop with stopping
% criteria to meet in order to end the loop. Our stopping criteria
% is based on the relative change of the residual. We also initialize
% the iteration counter to zero. The 'tic' command starts a clock so
% that the user will know how much computer time was required to
% meet the stopping criteria.

```

```

iter      = 0;
resid_old = 0;
resid_new = bez_mat * p - d;

```



```

tic

while norm(resid_new - resid_old)/max(1 , norm(resid_new)) > 10^(stop)

% Algorithm step 2a. Each iteration of the 'while' loop produces a
% new vector t by solving the nonlinear least squares problem
%
%
%                                $B(t) * p = d$ 
%
% where 'p' is the vector of control points, 'd' the matrix of data
% points, and 'B(t)' is the Bernstein matrix with unknown parameter
% values. Matrix 'B(t)' is nonlinear in terms of the parameter
% values. The method used in this function is the Gauss-Newton method
% where we let the residual be  $R(t) = B(t) * p - d$  and we let the
% Jacobian matrix 'J' be such that  $J_{i,j} = dB(t_i)/dt_j$ . I.E., the
% (i,j)-th element of 'J' is the slope along the Bezier curve at the
% i-th parameter value with respect to the j-th parameter
% value. The Gauss-Newton method says that the change in parameter
% values which will minimize the residual is given by
%
%
%                                $\Delta t = -\text{inv}(J' * J) * J' * R$ 
%
% where 'J' and 'R' are evaluated at the current parameter values.

% To compute the gradient at a point on the Bezier curve, we need the
% forward difference of the control points. I.E., we need a
% (j-1 x 2) matrix where the entries are  $p_{i+1} - p_i$  for  $i=1, \dots, j-1$ .
% The slope, 'deriv', is then determined multiplying the Bernstein
% matrix for a degree-minus-one curve by the forward difference matrix
% of control points and then multiplying by the degree of the original
% Bezier curve.

deriv = deg * mxbern2(t,deg-1) * (p(2:j,:) - p(1:j-1,:));

% Now we have what we need for the Gauss-Newton method. Since to use
% this method the residual needs to be a vector, we simply take the
% y-values of the residual and append them to the bottom of the
% x-values. This is done using 'resid(:)'. Similarly, the Jacobian
% matrix's elements must be for the new vector 'resid'. Each element
% of the matrix 'J' is the x-value or y-value of the slope at each

```

```

% parameter's point. Since the slope at any point on the curve
% doesn't change with any parameter other than it's own, most of the
% entries in 'J' are zero.

% 't', the new nodes are given by 't = t - delta_t'
% using the formula above. Because '(J' * J)' and 'J' have a form we
% know in advance, we can form 'delta_t' using less computer time and
% flops.

t = t - (deriv(:,1).*resid_new(:,1) + deriv(:,2).*resid_new(:,2)) ...
        ./ [deriv(:,1).^2 + deriv(:,2).^2] ;

% Now we have a new vector t, but we want to make sure the values
% are between 0 and 1. In most cases with well behaved data, the
% following rescaling of the nodes also results in the endpoints
% being associated with the nodes t_1=0 and t_m=1.

t = -min(t)*ones(i,1) + t;
t = t/max(t);

% With ordered nodes we now want the new control points so
% that we can reproduce the points 'tau' on the curve for the
% next iteration of the 'while' loop or to be used in the final plot
% if the stopping criteria is met. Note that if the condition is met
% we can also use the below 'bez_mat' matrix for the final plot. We
% also compute a new value for 'resid_new' and update the iteration
% counter.

bez_mat = mxbern2(t,deg);
p       = bez_mat \ d;

resid_old = resid_new;
resid_new = bez_mat * p - d;

iter     = iter + 1;

% End 'while' loop and stop computer time clock to show user how long
% it took to converge.

end

toc

```

```
% Now that we have the best fit vector t and the associated
% matrix 'p' of control points, we are ready to plot the final
% solution for the user. 'y' are the points on the Bezier curve
% associated with the vector t. 'y1' are the closely spaced
% points on the Bezier curve which matlab will use to plot a smooth
% looking curve.
```

```
y  = bez_mat * p;
y1 = bez_mat_1 * p;
```

```
% Finally, we clear the current plot and plot the results.
```

```
figure
plot(y1(:,1),y1(:,2))
hold on
plot(p(:,1),p(:,2),'*')
plot(y(:,1),y(:,2),'o')
plot(d(:,1),d(:,2),'+')
axis('equal')
```

```
% Include legend on final plot.
```

```
legend('-', 'Fitted Curve', '*', 'Control Points', ...
        'o', 'Nodes', '+', 'Data Points')
```

```
% Return hold state to however the user had it.
```

```
if (hold_was_off)
    hold off;
end
```

```
info = [norm(bez_mat*p-d), iter];
```

```
% End GRAD7.M
```

MXBERN2.M

```
function [B] = mxbern2(t,d)

% MXBERN2.M This function creates a Bernstein matrix of degree d
% using the values in the column vector t. A Bernstein matrix
% is a generalized Vandermonde matrix whose (i,j) entry is
%  $B_{j-1}^d(t_i)$ . The vector t must be a column vector with values
% between 0 and 1. Copyright (c) 3 December 1994 by Carlos F. Borges.
% All rights reserved. Modified by permission for this paper.

[n m] = size(t);
if (m ~= 1)
    error('t must be a column vector.');
```

end

% Check to see if nodes are within [0,1].

```
if min(t) < 0 | max(t) > 1
    error('nodes are not within [0,1]')
end
```

% Build the Bernstein matrix.

```
ct = 1 - t;
B = zeros(n,d+1);

for i = 0 : d
    B(:,i+1) = (t.^i).*(ct.^(d-i));
end
```

% If d > 22 we form the Bernstein matrix differently to
% avoid roundoff.

```
if d < 23
    B = B*diag( [1 cumprod(d:-1:1)./cumprod(1:d)] );
else
    B = B*diag(diag(fliplr(pascal(d+1))));
end
```

% End MXBERN2.M

AFF_ANGLE.M

```
function [h] = aff_angle(X)

% AFF_ANGLE.M This function returns an affine invariant vector of
% nodes for a given set of ordered data X. It is
% assumed that the user sends this function the data arranged in a
% (n x 2) matrix ordered from row one to row n.

% Obtaining the covariance matrix A is from a function AFF_KNT.M which
% is copyrighted on 3 December 1994 by Carlos F. Borges and appears
% here with his approval. The affine invariant angle method of
% obtaining nodes is found in a paper by Thomas A. Foley
% and Gregory M. Nielson, "Knot Selection for Parametric Spline
% Interpolation", in the book "Mathematical Methods in Computer Aided
% Geometric Design", Academic Press, Inc., 1989. Notation from this
% paper is used to annotate the steps in this algorithm.

n = size(X,1);

Xbar = X - ones(size(X)) * diag(mean(X));
Xcov = Xbar' * Xbar/n;
A = inv(Xcov);

% Obtain the node spacing values using the metric
%
% 
$$t_i = M[X](X_i, X_{i+1}).$$

%

V = X(2:n,:) - X(1:n-1,:);

t = diag(V * A * V') .^ (1/2);

% Obtain the values for
%
% 
$$M^2[X](X_{i-1}, X_{i+1})$$

%

V_2 = X(3:n,:) - X(1:n-2,:);

t_2 = diag(V_2 * A * V_2');
```

```

% Get theta_i values. This is what takes into account the bending
% of the data.

theta = zeros(n-1,1);

for j = 2:n-1

    theta(j) = min( pi - acos( (t(j-1)^2 + t(j)^2 - ...
                                t_2(j-1))/(2*t(j)*t(j-1)) ) , pi/2);

end

% Obtain the affine invariant angle node spacing values h_i.

h = zeros(n-1,1);

h(1) = t(1)*( 1 + (1.5*theta(2)*t(2))/(t(1) + t(2)) );

for j = 2:n-2

    h(j) = t(j) * ( 1 + (1.5*theta(j)*t(j-1))/(t(j-1)+t(j)) +...
                    (1.5*theta(j+1)*t(j+1))/(t(j)+ t(j+1)) );

end

h(n-1) = t(n-1) * ( 1 + (1.5*theta(n-1)*t(n-2))/(t(n-2)+t(n-1)) );

% Now that we have the node spacing values, we want to normalize
% them so that they are within [0,1], with the first data point being
% associated with the value zero and the last data point with the
% value one.

h = [0;h];

h = cumsum(h);

h = h / h(n);

% End AFF_ANGLE.M

```


LIST OF REFERENCES

- [1] R. T. Farouki and V. T. Rajan. On the Numerical Condition of Polynomials in Bernstein Form. *Computer Aided Graphic Design*, 4:191-216, 1987.
- [2] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, Inc., San Diego, California, 1993.
- [3] Gregory M. Nielson and Thomas A. Foley. A Survey of Applications of an Affine Invariant Norm. *Mathematical Methods in Computer Aided Geometric Design*, pages 445-467, 1989.
- [4] Thomas A. Foley and Gregory M. Nielson. Knot Selection for Parametric Spline Interpolation. *Mathematical Methods in Computer Aided Geometric Design*, pages 261-271, 1989.
- [5] Philip J. Schneider. Solving the Nearest-Point-on-Curve Problem. *Graphics Gems*, pages 607-611, 1990.
- [6] Charles S. Beightler, Don T. Phillips and Douglass J. Wilde. *Foundations of Optimization*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.
- [7] J. E. Dennis, JR. and Robert D. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
8725 John J. Kingman Rd., STE 0944
Ft. Belvoir, Virginia 22060-6218
2. Dudley Knox Library 2
Naval Postgraduate School
411 Dyer Rd.
Monterey, California 93943-5101
3. Director, Training and Education 1
MCCDC, Code C46
1019 Elliot Rd.
Quantico, Virginia 22134-5027
4. Director, Marine Corps Research Center 2
MCCDC, Code C40RC
2040 Broadway Street
Quantico, Virginia 22134-5107
5. Director, Studies and Analysis Division 1
MCCDC, Code C45
3300 Russell Road
Quantico, Virginia 22134-5130
6. Marine Corps Representative 1
Naval Postgraduate School
Code 037, Bldg. 234, HA-220
699 Dyer Road
Monterey, California 93940
7. Marine Corps Tactical Systems Support Activity 1
Technical Advisory Branch
Attn: Major J.C. Cummiskey
Box 555171
Camp Pendleton, California 92055-5080
8. Superintendent 1
ATTN: Professor Carlos Borges
(Code MA/Bc)
Naval Postgraduate School
Monterey, California 93943-5000

9. Superintendent 1
ATTN: Professor Richard Franke
(Code MA/Fe)
Naval Postgraduate School
Monterey, California 93943-5000