SPLINE ESTIMATION OF PRINCIPAL CURVES

by

Marcel Walther

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Mathematics

at

The University of Wisconsin-Milwaukee

May 2016

ABSTRACT

SPLINE ESTIMATION OF PRINCIPAL CURVES

by

Marcel Walther

The University of Wisconsin-Milwaukee, 2016
Under the Supervision of Professor Daniel Gervini

Finding low-dimensional approximations to high-dimensional data is one of the most important topics in statistics, which has also multiple applications in economics, engineering and science. One suggestion in the literature ,based on kernel smoothing, is a non-linear generalization of principal components. This kernel-based approach comes with several complications. Therefore the purpose of this thesis is to provide an alternative based on spline smoothing which produces more reliable results.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Estimating functions from given data sets is one of the most important topics in statistics, which has also multiple applications in economics, engineering and science. Many problems are linear, that is the reason why estimating linear function out of data points is well studied. Mostly, for this problem the method of least squares is used.

But in some cases it is apparent that the relation between two variables is not linear. That leads to the question of how to estimate non-linear functions. One suggestion in the literature is from Hastie and Stuetzle [1], who developed the algorithm based on principal curves. In general, this model fits non-linear data well, but it comes with several complications. One of the problems is that the procedure does not always converge. Furthermore, it also seems to be possible to improve the quality of the estimation, by reducing the estimation error.

Therefore the purpose of this thesis is to provide an algorithm which produces more reliable results. It develops a procedure based on splines which aims at improving the method approach of Hastie and Stuetzle [1]. First we present the necessary theory, which includes B-splines and the evaluation of derivatives. Then we present the algorithm and compare the methods

of kernel-based principal curves and the new spline based method. Finally, we focus on the parameters of the spline method and their influence on the quality of the algorithm.

2 Principal Curves

First, we review the concept of Principal Curves, introduced in the paper of Hastie and Stuetzle [1]. We will focus on the main definitions and propositions which are needed for the algorithm of Hastie and Stuetzle.

Definition 1

We define the projection index $\lambda_f : R^p \to R^1$ as:

$$\lambda_f(x) = \sup_{\lambda} \{\lambda : \|x - f(\lambda)\| = \inf_{\mu} \|x - f(\mu)\|\}$$

Definition 2

The curve f is called self-consistent if

$$E(X|\lambda_f = \lambda) = f(\lambda) \text{ for a.e. } \lambda.$$

Proposition 1

If a straight line $l(\lambda) = u_0 + \lambda v_0$ is self-consistent, then it is a principal component.

Definition 3

The curve f is called a critical point of the distance function for variations in the class G if

$$\left.\frac{\delta D^2(h, f_t)}{\delta t}\right|_t = 0 \quad \forall g \in G,$$

where $f : d(x, f) = \|x - f(\lambda_f(x))\|$ and $D^2(h, f) = E_h d^2(X, f)$.

Proposition 2

Let G_t denote the class of straight lines $g(\lambda) = a + \lambda b$. A straight line $l_0(\lambda) = a_0 + \lambda b_0$ is a critical point of the distance function for variations in G_t if b_0 is an eigenvector of $cov(X)$ and $a_0 = 0$

Proposition 3

Let G_B denote the class of smooth (C^∞) curves parameterized over Λ, with $\|g\| \leq 1$ and $\|g'\| \leq 1$. Then f is a principal curve of h if f is a critical point of the distance function for perturbations in G_B

With these definitions and propositions the following algorithm was proposed by [1]:

Initialization: Set $f^{(0)}(\lambda) = \bar{x} + a\lambda$, where a is the first linear principal component of h. Set $\lambda^{(0)}(x) = \lambda_{f^{(0)}(x)}$.

Repeat: iteration counter j

1. Set $f^{(j)}(\cdot) = \mathbb{E}(X|\lambda_{f^{(j-1)}}(X) = \cdot)$, which is approximated by kernel smoothing.

2. Define $\lambda^{(j)}(x) = \lambda_{f^{(j)}(x)} \quad \forall x \in h$; transform $\lambda^{(j)}$ so that $f^{(j)}$ is unit speed.

3. Evaluate $D^2(h, f^{(j)}) = \mathbb{E}_{\lambda^{(j)}}\mathbb{E}[\|X - f(\lambda^{(j)}(X))\|^2|\lambda^{(j)}(X)]$

Until: the change in $D^2(h, f^{(j)})$ is below some threshold.

This algorithm is implemented in the program which can be found in Appendix A. For visualization we simulated a principal curve for 500 data points.

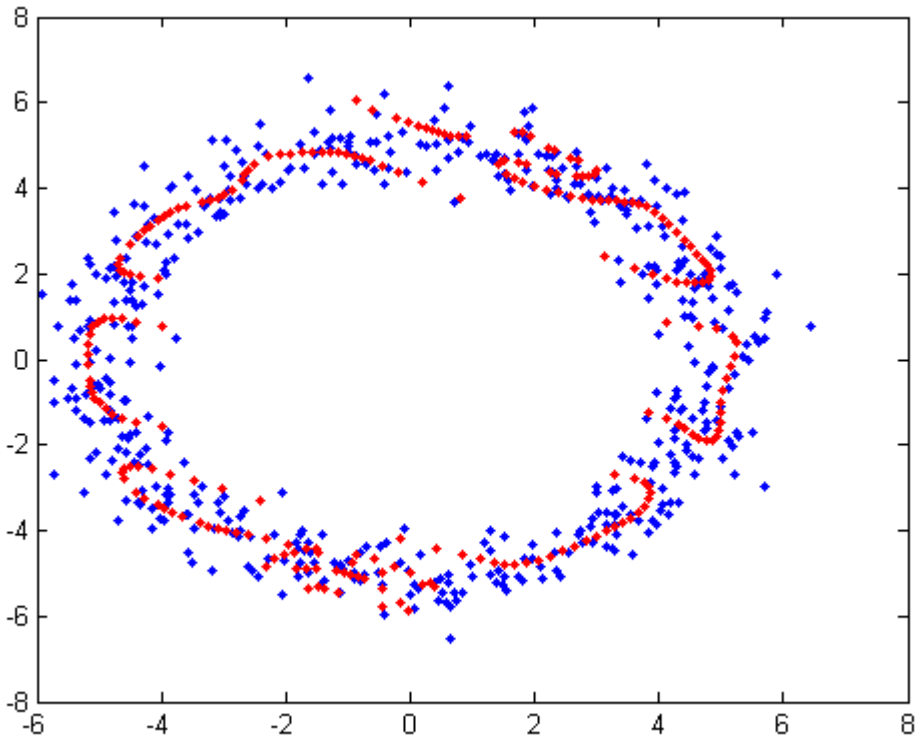

Figure 1: Principal curve of circle data and bandwidth=0.01

The given data is plotted in blue and the estimated data, simulated with the principal curve method, is illustrated in red. We can observe that the algorithm works well, but the estimated points seems to be not very smooth. Also the distance between the estimated data and the real data points can be improved.

3 Spline Estimation

The previous chapter introduced the theory of principal curves and provided a kernel-based algorithm for estimating the principal curves. But it can be observed that the algorithm of [1] has the following problems:

- It does not always converge
- It is hard to find the optimal bandwidth

For these reasons, we propose an alternative method based on splines that will ameliorate these problems.

3.1 Basic Idea

Because of the problems discussed in the previous subsection, we want to develop an alternative approach. Our idea is to use splines for estimating the curves.

Therefore we consider a set of points $x_1, ..., x_n$ in R^p and we want to estimate a function $f : [0, 1] \to R^p$ which fulfills

$$x_i = f(t_i) + e_i, \quad t_i \in [0, 1],$$

where e_i represents the random error. This equation can also be written with matrices

$$\begin{pmatrix} x_{11} & \dots & x_{1n} \\ \vdots & & \vdots \\ x_{p1} & \dots & x_{pn} \end{pmatrix} = \begin{pmatrix} f_1(t_1) & \dots & f_1(t_n) \\ \vdots & & \vdots \\ f_p(t_1) & \dots & f_p(t_n) \end{pmatrix} + \begin{pmatrix} \epsilon_{11} & \dots & \epsilon_{1n} \\ \vdots & & \vdots \\ \epsilon_{p1} & \dots & \epsilon_{pn} \end{pmatrix}$$

The most common procedure for estimation is the least squares method which minimizes the distance between the points x_i and the function $f(t_i)$. In mathematical terms it can be expressed as:

$$(\hat{f}, \hat{t_1}, ..., \hat{t_n}) = argmin \sum_{i=1}^{n} \|x_i - f(t_i)\|^2 = argmin \sum_{i=1}^{n} \sum_{j=1}^{p} (x_{ji} - f_j(t_i))^2.$$

This problem is very well studied for linear functions. But the interesting ones are the non-linear cases. Here we use splines for estimating the function f. Let $\beta_1(t), ..., \beta_q(t)$ be a spline basis on $[0, 1]$ and $f = (f_1, ..., f_p)$ where each f_j is a univariate function. Then we estimate for each $j = 1, ..., p$ the function f_j with

$$f_j(t) = \sum_{k=1}^{q} a_{jk}\beta_k(t).$$

The splines (β_i) can be chosen, but the coefficients $\{a_{jk}\}$ have to be estimated from the data.

We know from [2] that if the number of knots is very high then f becomes too irregular. A solution for this complication is also provided in [2]. The paper suggests to add a roughness penalty of the form $\lambda \sum_{j=1}^{p} \int_0^1 (f_j'')^2$, where λ can be chosen. Because we use splines for estimating the functions f_j the penalty function can be expressed as

$$\lambda \sum_{j=1}^{p} \int_0^1 (f_j'')^2 = \lambda \sum_{k=1}^{q} \sum_{k'=1}^{q} a_{jk} a_{jk'} \int_0^1 \beta_k''(t)\beta_{k'}''(t)dt.$$

Now, we have the setting for combining the tools we have introduced. The function we want to minimize is

$$F(a_1,\ldots,a_p,t_1,\ldots,t_n)=\sum_{i=1}^{n}\sum_{j=1}^{p}(x_{ji}-f_j(t_i))^2+\lambda\sum_{j=1}^{p}\int_0^1(f_j'')^2= \tag{1}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{p}(x_{ji}-\sum_{k=1}^{q}a_{jk}\beta_k(t_i))^2+\lambda\sum_{j=1}^{p}\sum_{k=1}^{q}\sum_{k'=1}^{q}a_{jk}a_{jk'}\int_0^1\beta_k''(t)\beta_{k'}''(t)dt. \tag{2}$$

The function $F(a_1,\ldots,a_p,t_1,\ldots,t_n)$ depends on $a_1,...,a_p$ and $t_1,...,t_n$. So we have to find the derivatives with respect to these variables and set each equation equal to zero. In the following we focus on the two dimensional case (p=2).

3.2 B-Splines

For the algorithm developed in the next subsection we use a special form of splines which is called B-splines. The theory which is represented can be found in [3]. Let $t_0 \leq t_2 \leq ... \leq t_m$ be knots and n the degree of the splines. Then the spline basis functions satisfy:

$$\beta_{i,0}(x) = \begin{cases} 1 & \text{if} \quad t_i < x < t_{i+1} \\ \\ 0 & else \end{cases}$$

$$\beta_{i,k}(x) = \frac{x - t_i}{t_{i+k-1} - t_i}\beta_{i,k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}}\beta_{i+1,k-1}(x).$$

The number of B-splines is $q = n+m-2$. Each basis function is a polynomial of degree n. Mostly, we use equidistant knots, but the algorithm also works in the general case. The algorithm that computes B-splines in Matlab is given in Appendix B.

3.3 Determining a Spline Estimator for Two Dimensional Data

Our main task is to develope an algorithm which estimates principal curves in two dimension. For this purpose, we have to minimize the function stated in (1) for the case p=2:

$$\min_{a,t} F(a_1, a_2, t_1, ..., t_n) =$$
$$\min_{a,t} \sum_{i=1}^{n}((x_{1i} - f_1(t_i))^2 + (x_{2i} - f_2(t_i))^2) + \lambda \sum_{j=1}^{p} \int_0^1 (f_j''(t))^2 dt =$$
$$\min_{a,t} \sum_{i=1}^{n}((x_{1i} - \sum_{k=1}^{q} a_{1k}\beta_k(t_i))^2 + (x_{2i} - \sum_{k=1}^{q} a_{2k}\beta_k(t_i))^2)$$
$$+ \lambda \left(\sum_{k=1}^{q}\sum_{k'=1}^{q} a_{1k}a_{1k'} \int_0^1 \beta_k''(t)\beta_{k'}''(t)dt + \sum_{k=1}^{q}\sum_{k'=1}^{q} a_{2k}a_{2k'} \int_0^1 \beta_k''(t)\beta_{k'}''(t)dt \right),$$

where a_1 and a_2 are vectors $a_1 = (a_{11}, ..., a_{1q})$ and

$a_2 = (a_{21}, ..., a_{2q})$. The next step is to find the minimum of the function F. Therefore we have to find the derivative of the function F with respect to a_1, a_2 and t, and set the resulting equations equal to zero. First we want to find

the derivative of F with respect to the variables a_{1l} $(l = 1, ...q)$.

$$\frac{1}{2}\frac{\partial F(a_1, a_2, t_1, \ldots, t_n)}{\partial a_{1l}} = \sum_{i=1}^{n} \beta_l(t_i)(x_{1i} - \sum_{k=1}^{q} a_{1k}\beta_k(t_i)) + \lambda \sum_{k=1}^{q} a_{1k} \int_0^1 \beta_k''(t)\beta_l''(t)dt =$$

$$\begin{pmatrix} \beta_1(t_1) & \ldots & \beta_1(t_n) \\ \vdots & \ldots & \vdots \\ \beta_q(t_1) & \ldots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} x_{11} \\ \vdots \\ x_{1n} \end{pmatrix} - \begin{pmatrix} \beta_1(t_1) & \ldots & \beta_1(t_n) \\ \vdots & \ldots & \vdots \\ \beta_q(t_1) & \ldots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} \beta_1(t_1) & \ldots & \beta_q(t_1) \\ \vdots & \ldots & \vdots \\ \beta_1(t_n) & \ldots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} a_{11} \\ \vdots \\ a_{1q} \end{pmatrix} +$$

$$\lambda \begin{pmatrix} \int_0^1 \beta_1''(t)\beta_1''(t)dt & \ldots & \int_0^1 \beta_1''(t)\beta_q''(t)dt \\ \vdots & \ldots & \vdots \\ \int_0^1 \beta_q''(t)\beta_1''(t)dt & \ldots & \int_0^1 \beta_q''(t)\beta_q''(t)dt \end{pmatrix} \begin{pmatrix} a_{11} \\ \vdots \\ a_{1q} \end{pmatrix}.$$

We can find the derivative of F with respect to a_{2l} with the same procedure

$(l = 1, ...q)$:

$$\frac{1}{2}\frac{\partial F(a_1, a_2, t_1, \ldots, t_n)}{\partial a_{2l}} = \sum_{i=1}^{n} \beta_l(t_i)(x_{2i} - \sum_{k=1}^{q} a_{2k}\beta_k(t_i)) + \lambda \sum_{k=1}^{q} a_{2k} \int_0^1 \beta_k''(t)\beta_l''(t)dt =$$

$$\begin{pmatrix} \beta_1(t_1) & \ldots & \beta_1(t_n) \\ \vdots & \ldots & \vdots \\ \beta_q(t_1) & \ldots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} x_{21} \\ \vdots \\ x_{2n} \end{pmatrix} - \begin{pmatrix} \beta_1(t_1) & \ldots & \beta_1(t_n) \\ \vdots & \ldots & \vdots \\ \beta_q(t_1) & \ldots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} \beta_1(t_1) & \ldots & \beta_q(t_1) \\ \vdots & \ldots & \vdots \\ \beta_1(t_n) & \ldots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} a_{21} \\ \vdots \\ a_{2q} \end{pmatrix} +$$

$$\lambda \begin{pmatrix} \int_0^1 \beta_1''(t)\beta_1''(t)dt & \ldots & \int_0^1 \beta_1''(t)\beta_q''(t)dt \\ \vdots & \ldots & \vdots \\ \int_0^1 \beta_q''(t)\beta_1''(t)dt & \ldots & \int_0^1 \beta_q''(t)\beta_q''(t)dt \end{pmatrix} \begin{pmatrix} a_{21} \\ \vdots \\ a_{2q} \end{pmatrix}.$$

For finding the minimum, we have to set these derivatives equal to zero.

With the matrix notation we can combine these equations and we get.

$$\begin{pmatrix} \beta_1(t_1) & \dots & \beta_1(t_n) \\ \vdots & \dots & \vdots \\ \beta_q(t_1) & \dots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} x_{11} & x_{21} \\ \vdots & \vdots \\ x_{1n} & x_{2n} \end{pmatrix} - \begin{pmatrix} \beta_1(t_1) & \dots & \beta_1(t_n) \\ \vdots & \dots & \vdots \\ \beta_q(t_1) & \dots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} \beta_1(t_1) & \dots & \beta_q(t_1) \\ \vdots & \dots & \vdots \\ \beta_1(t_n) & \dots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} a_{11} & a_{21} \\ \vdots & \vdots \\ a_{1q} & a_{2q} \end{pmatrix}$$
$$+ \lambda \begin{pmatrix} \int_0^1 \beta_1''(t)\beta_1''(t)dt & \dots & \int_0^1 \beta_1''(t)\beta_q''(t)dt \\ \vdots & \dots & \vdots \\ \int_0^1 \beta_q''(t)\beta_1''(t)dt & \dots & \int_0^1 \beta_q''(t)\beta_q''(t)dt \end{pmatrix} \begin{pmatrix} a_{11} & a_{21} \\ \vdots & \vdots \\ a_{1q} & a_{2q} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix}$$

This leads to the following solution for a_1 and a_2

$$\begin{pmatrix} a_{11} & a_{21} \\ \vdots & \vdots \\ a_{1q} & a_{2q} \end{pmatrix} =$$
$$\left(\begin{pmatrix} \beta_1(t_1) & \dots & \beta_1(t_n) \\ \vdots & \dots & \vdots \\ \beta_q(t_1) & \dots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} \beta_1(t_1) & \dots & \beta_q(t_1) \\ \vdots & \dots & \vdots \\ \beta_1(t_n) & \dots & \beta_q(t_n) \end{pmatrix} + \lambda \begin{pmatrix} \int_0^1 \beta_1''(t)\beta_1''(t)dt & \dots & \int_0^1 \beta_1''(t)\beta_q''(t)dt \\ \vdots & \dots & \vdots \\ \int_0^1 \beta_q''(t)\beta_1''(t)dt & \dots & \int_0^1 \beta_q''(t)\beta_q''(t)dt \end{pmatrix} \right)^{-1}$$
$$\begin{pmatrix} \beta_1(t_1) & \dots & \beta_1(t_n) \\ \vdots & \dots & \vdots \\ \beta_q(t_1) & \dots & \beta_q(t_n) \end{pmatrix} \begin{pmatrix} x_{11} & x_{21} \\ \vdots & \vdots \\ x_{1n} & x_{2n} \end{pmatrix} \tag{3}$$

So we get an explicit solution for a_1 and a_2.

Now we want to find a solution for $t_i \;\; (i = 1, ...n)$. Therefore we derive F with respect to $t_i \;\; (i = 1, ..., n)$ and set the equations equal to zero.

$$\frac{1}{2}\frac{\partial F(a_1, a_2, t_1, \ldots, t_n)}{\partial t_i} = f_1'(t_i)(x_{1i} - f_1(t_i)) + f_2'(t_i)(x_{2i} - f_2(t_i)) =$$
$$\left(\sum_{k=1}^{q} a_{1k}\beta_k'(t_i)\right)\left(x_{1i} - \sum_{k=1}^{q} a_{1k}\beta_k(t_i)\right) + \left(\sum_{k=1}^{q} a_{2k}\beta_k'(t_i)\right)\left(x_{2i} - \sum_{k=1}^{q} a_{2k}\beta_k(t_i)\right) \stackrel{!}{=} 0$$

We recognize that there is no explicit solution for this equation. That is the reason why we have to use a numerical procedure. We are going to use Newton-Raphson [4]. Therefore, we define for $i = 1, ..., n$

$$g(t_i) = f_1'(t_i)(x_{1i} - f_1(t_i)) + f_2'(t_i)(x_{2i} - f_2(t_i)).$$

For the algorithm, we need the derivative of g with respect to $t_i \;\; (i = 1, ..., n)$, which is

$$g'(t_i) = f_1''(t_i)(x_{1i} - f_1(t_i)) - (f_1'(t_i))^2 + f_2''(t_i)(x_{2i} - f_2(t_i)) - (f_2'(t_i))^2.$$

Now, we can state the Newton-Raphson algorithm and we get a numerical solution for t_i: The k-th-step update of t_i is

$$t_i^{(k+1)} = t_i^{(k)} + \frac{g(t_i^{(k)})}{g'(t_i^{(k)})}. \tag{4}$$

We have created the tools for the algorithm, which is the main purpose of this thesis. The only question which has to be answered is the initial vector t. We use the equidistant points in [0,1]. To summarize, we get the following algorithm:

1. Initialize t with $t_i = \frac{i}{n+1}$, $a = (a_1, a_2)$ with the equation (3) and $a_{old} = 0$
2. Repeat while $\|a_{old} - a\|$ is greater than a threshold:
   (a) $a_{old} = a$
   (b) $t_i = t_i + \frac{g(t_i)}{g'(t_i)}$, with the functions g and g' computed in this chapter
   (c) compute a with (3) and the new a.
3. Compute t with (4) until t converges.

The algorithm implemented in Matlab can be found in Appendix C. To get a first impression of the algorithm, we show an example in Figure 2.

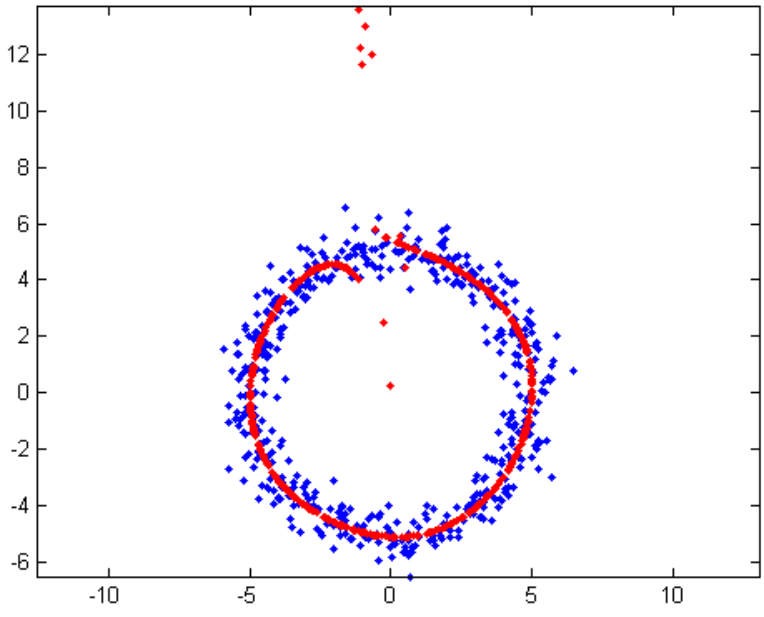

Figure 2: Spline estimation for 500 circle data

We used the same data set as in Figure 1. It can be observed that the estimated curve looks smoother than the one in Figure 1. It also seems that the average distance is smaller. In the next chapter we verify this in a more systematic way by simulation.

4 Validation of the Spline Estimation Procedure

In the last section we developed a spline-based algorithm, for principal curve estimation.

In this section, we want to verify that spline estimation produces better results than the approach of Hastie and Stuetzle [1]. For this reason, we have to create a homogeneous setting to compare the algorithms reasonably. The first step is to define an interesting data set which shows the behavior of the approaches. Our task is to create a procedure which estimates non-linear data well. Therefore we check the algorithm with the following two dimensional data sets, which represent a circle:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} sin(t) \\ cos(t) \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \end{pmatrix},$$

where t is uniformly distributed in $[0, 2\pi)$ and e_1 and e_2 are independent $N(0, \sigma)$. If σ is high, also the fluctuation of the points increases. This gives us the opportunity to test our algorithms in different settings. The bigger σ is, the more challenging it is to estimate the functions properly.

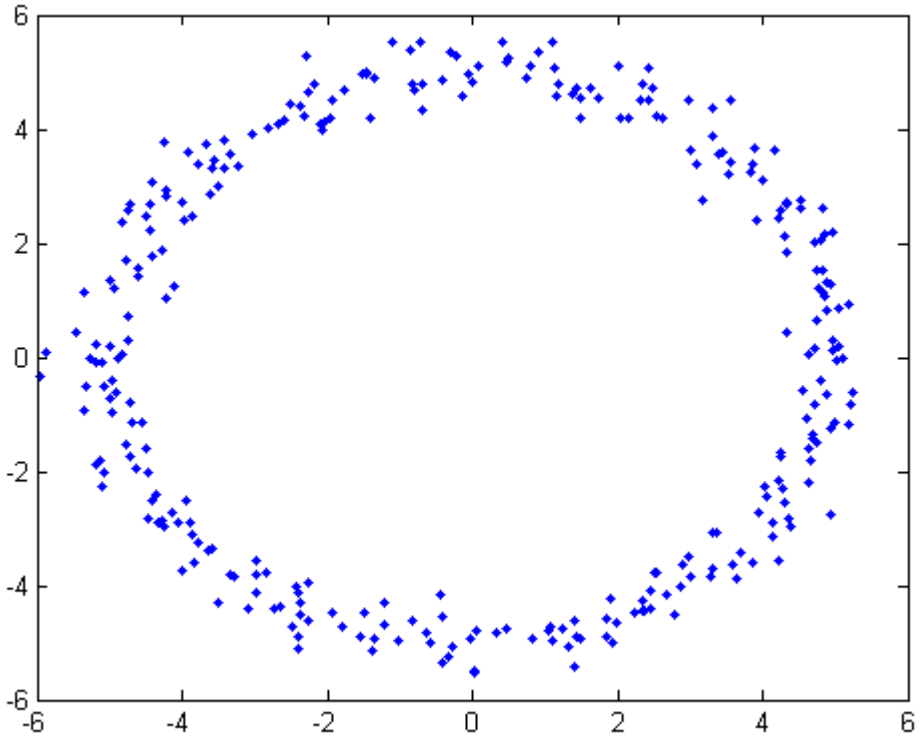

Figure 3: Circle data with $\sigma = 0.3$ and $n = 300$

Furthermore, we have to define an empirical indicator for comparing the results of the different methods. Intuitively, we want to compare the distance between the original data and the estimated curves.

This leads to the following error measure for different simulations (j):

$$err_j = \sqrt{\int (\hat{f}_1 - f_{01}(t))^2 dt + \int (\hat{f}_2(t) - f_{02}(t))^2 dt},$$

where $\hat{f} = (\hat{f}_1, \hat{f}_2)$ is the estimated curve and $f_0 = (f_{01}, f_{02})$ is the true curve. Because the algorithms estimate only finite values, the integral becomes a sum:

$$err_j = \sqrt{\sum_{i=1}^{n} (\hat{f}_1(t_i) - f_{01}(t_i))^2 + \sum_{i=1}^{n} (\hat{f}_2(t_i) - f_{02}(t_i))^2}.$$

Because we simulate random data, it happens that we get different errors for the same σ. So we simulate the procedure several times and average the errors. Therefore we end up with the mean error (ME)

$$ME = \frac{1}{m} \sum_{j=1}^{m} err_j,$$

where m is the number of simulations. In the following, we set them equal to 500.

4.1 Comparing the Kernel-Based and Spline-Based Methods of Principal Curves

In both algorithms there are variables which also have to be estimated. In the spline estimation procedure, we have to choose λ and for the principal curve method we have to find the best bandwidth. In general, these questions need further theoretical work. Therefore we choose the parameters in a heuristic way. For these data, the bandwidth is in the interval of $[10^{-1}, 10^{-6}]$ and the parameter λ can usually be found in the range of $[10^{-1}, 0]$, where $\lambda = 0$ means that no penalty is necessary for the spline estimation. These considerations lead to the idea of picking the best parameters out of the intervals for each algorithm.

In conclusion, we simulate both algorithms with the following variations of the data set:

- Different standard derivation σ of the random errors
- Different sample sizes n

The respective ME's are shown in Table 1 and Table 2.

	n=50	n=100	n=300
$\sigma = 0.1$	0.6034	0.9388	1.6994
$\sigma = 0.5$	2.9754	4.5132	8.4360
$\sigma = 1$	5.7585	8.7661	16.7219

Table 1: Error of spline algorithm with different σ and n

	n=50	n=100	n=300
$\sigma = 0.1$	44.8342	63.5606	109.7636
$\sigma = 0.5$	42.664	59.5541	111.3093
$\sigma = 1$	46.4490	58.8301	113.4822

Table 2: Error of kernel-based algorithm with different σ and n

It can be observed that all errors of the kernel-based estimator are bigger than the ones of the spline estimator. This table confirms the graphical observation we made in the previous section. For better illustration, we can also look at the boxplot of the single errors (err). As a representative we take the boxplot of the data set with $\sigma = 0.5$ and $n = 300$.

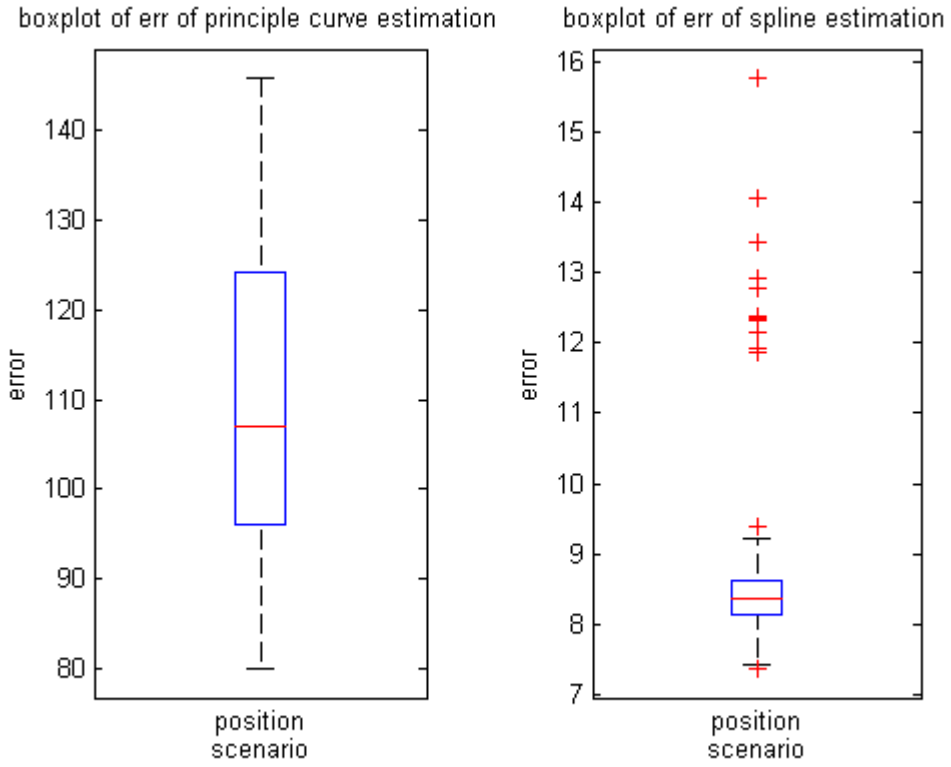

Figure 4: Boxplot of errors of principal curve estimation and spline estimation ($n = 300$, $\sigma = 0.1$)

The boxplot shows that all individual errors of the spline estimation are smaller than the ones from the kernel-based estimation. Now, the remaining question is what the plots look like. This can be observed in the next figure. In this picture the given data is plotted in blue, the estimation of the kernel-based curve is green and the spline estimation is colored in red.

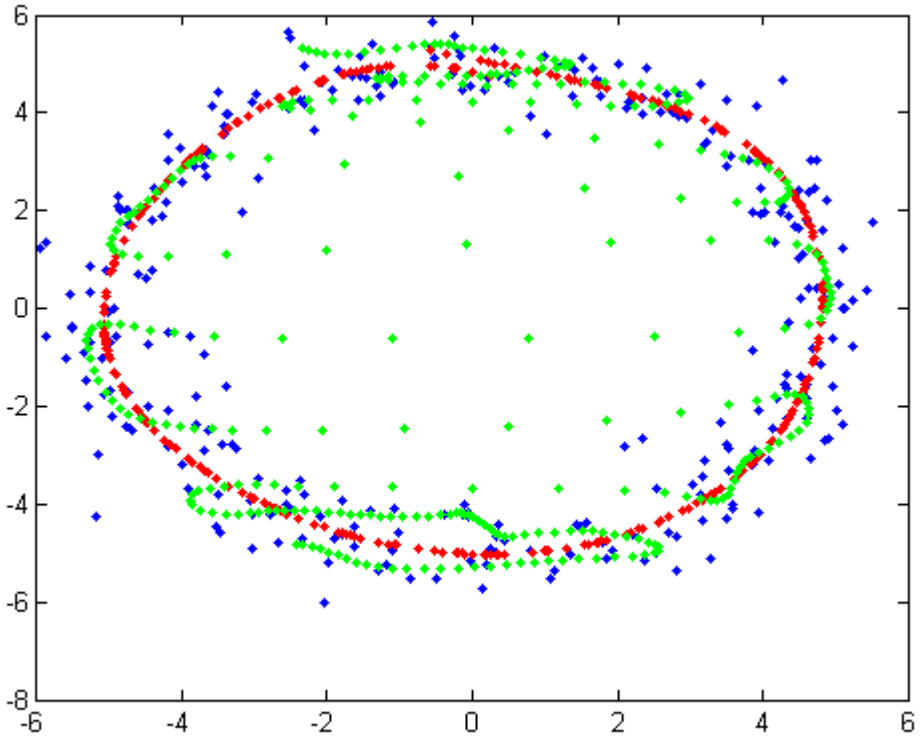

Figure 5: plot of estimated points of principal curve estimation and spline estimation ($n = 300$, $\sigma = 0.1$)

Also this graphic confirms the fact that the spline estimation produces better estimators than the approach of [1]. But it is very surprising that the kernel-based curve is that worse. This seems to be a contradiction to Figure 1, where the estimations seem to be closer on the real data and in fact with a different bandwidth we get better plots. But if we look closer on the estimated data we realize that in some points $(f_1(\hat{t}_i), f_2(\hat{t}_i))$ the algorithm of

Hastie and Stuetzle [1] does not produce a solution. For example we can get the following plot for a different bandwidth (=0.0001).

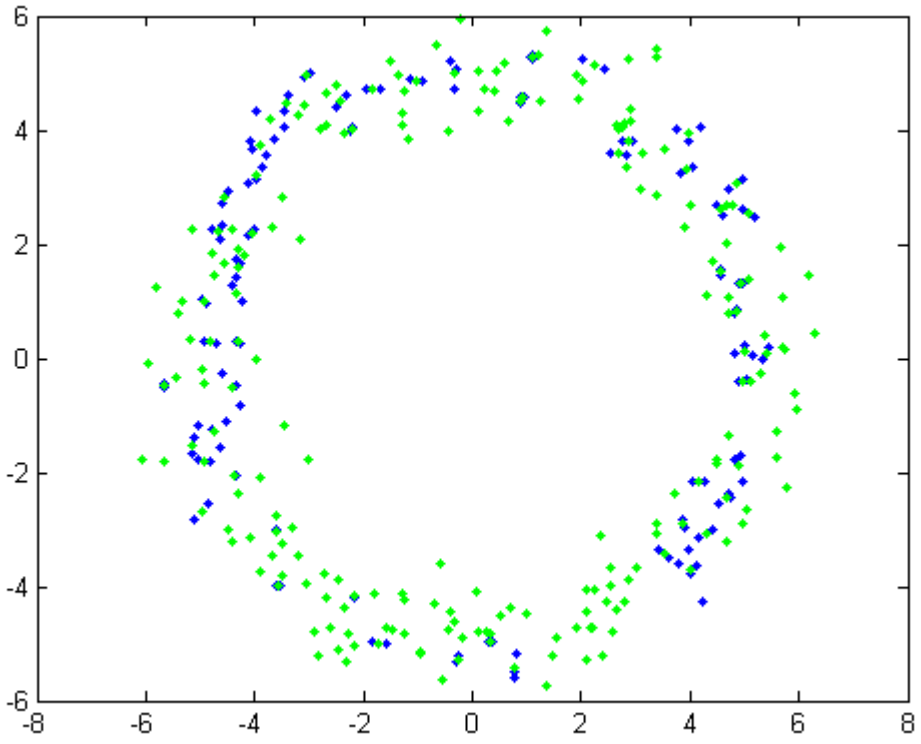

Figure 6: Principal curve of 300 data points and bandwidth=0.0001

We realize that the estimated data seems to fit better than the points in Figure 4. But the algorithm does not converge in three points for the bandwidth of 0.0001. Therefore we cannot compare it with the spline estimation properly. Nevertheless, also the result of the principal curve with band-

width=0.0001 seems to be worse than the estimated data of the spline estimation.

So, we can conclude that the spline estimation leads to better results than the kernel-based approach. Either the mean error of the spline estimation is smaller, or the error for the kernel-based curve estimation cannot be determined because the algorithm does not converge at every point.

4.2 Relationship between the Number of Knots and λ

In the previous subsection, we figured out that the spline method estimates the initial data better than the procedure based on kernels. Another interesting question is the relationship between the number of knots and the optimal λ and if a higher number of knots leads to better results. For this reason we compare the mean error, defined in the last subsection, for different numbers of knots.

	$k = 5$	$k = 10$	$k = 20$
$\sigma = 0.1$	1.6988 $(\lambda = 0)$	1.6784 $(\lambda = 0)$	1.6992 $(\lambda = 0)$
$\sigma = 0.5$	8.4360 $(\lambda = 10^{-4})$	8.2443 $(\lambda = 10^{-6})$	16.7764 $(\lambda = 10^{-4})$
$\sigma = 1$	16.7764 $(\lambda = 10^{-4})$	18.984 $(\lambda = 10^{-3})$	18.893 $(\lambda = 10^{-4})$

Table 3: Connection between number of knots and the ME

The table shows that the number of knots does not have a huge effect on the mean error. The errors are approximately in the same range and can be explained with a randomness of the simulated data. The other interesting question is how the penalty parameter (λ) and the amount of knots (k) is connected. Also in this case a connection cannot be observed.

But we see a relationship between λ and σ. If σ gets larger, also λ increases. This means that if data gets more variable the algorithm needs a

higher penalty parameter. Now, we want to have a look on the plots of the simulations.

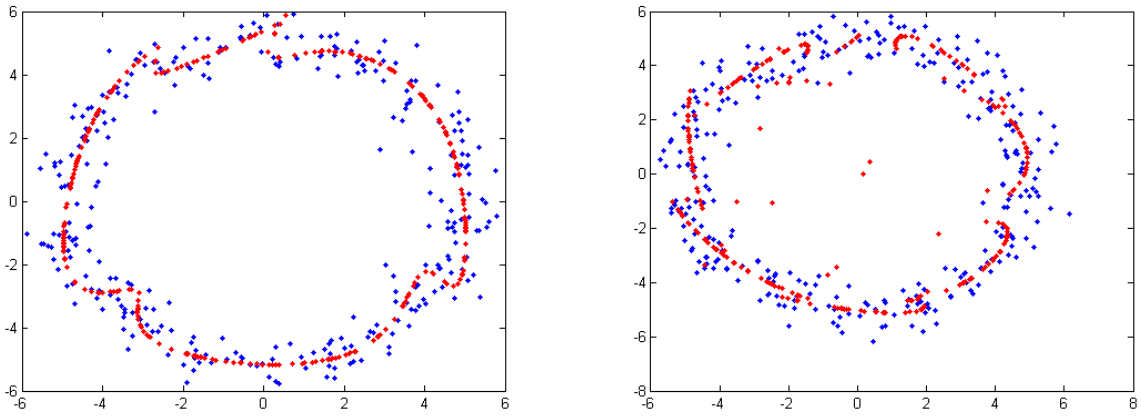

Figure 7: Two plots with different amount of knots (left k=10, right k=20)

The figures show that the plot with 10 knots seems to be smoother than the plot with 20 knots.

5 Conclusion

In this thesis we presented a new method for principal curve estimation based on B-splines. This method was based on Hastie and Stuetzle [1] whose principal curve algorithm produces results which can be improved in several aspects.

One problem of the kernel-based principal curve is the quality of the estimators. The distance between the estimated points and the data is very large. We solved this issue by developing a spline-based estimation method. We showed in Section 4 that the new approach is sometimes better than the algorithm of [1].

Another problem which was the fact that in the procedure of Hastie and Stuetzle [1] the algorithm does not converge in every point, could be solved. This issue does not occur with the new algorithm presented in this thesis.

Therefore, we can conclude that the spline estimation algorithm solves the main problems which can be observed in the kernel-based principal curve algorithm.

References

[1] Trevor Hastie and Werner Stuetzle. Principal curves. *Journal of the American Statistical Association*, 1989.

[2] B. W. Silverman. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society.*, 1985.

[3] Carl de Boor. *A Practical Guide to Splines.* Springer-Verlag, 1978.

[4] A. Hohmann P. Deuflhard. *Numerische Mathematik I. Eine algorithmisch orientierte Einführung. 3. überarbeitete und erweiterte Auflage.* de Gruyter, 2002.

Appendices

A Kernel-Based Principal Curve

```
function [t,f,sc,fsc] = princurve(x,h)
% [t,f,sc,fsc] = princurve(x,h)
% Principal curve
% Input: x (n x p) data matrix, h (scalar>0) bandwidth
% Output: t (m x 1) grid, f (m x p) values of f on the grid,
%         sc (n x 1) individual scores, fsc (n x p) values of f on the scores

[n,p] = size(x);
m = 300;
t = linspace(0,1,m)';
fdot = zeros(m,p);

% Initial estimator (first PC)
[U,S] = svd(cov(x));
z = (x-repmat(mean(x),[n 1]))*U(:,1);
fsc = repmat(mean(x),[n 1]) + z*U(:,1)';
DX = sqrt(sum((x-fsc).^2,2));
a = min(z);
b = max(z);
sc = (z-a)/(b-a);
f = repmat(mean(x),[m 1]) + (a+(b-a)*t)*U(:,1)';
if p==2
plot(x(:,1),x(:,2),'.',fsc(:,1),fsc(:,2),'o',f(:,1),f(:,2))
pause(.3)
end

% Iterations
err = 1;
iter = 0;
while err>1e-3 && iter<50
iter = iter + 1;
DX0 = DX;
```

```
% Update function and derivatives
for j = 1:m
w = normpdf((sc-t(j))/h)/h;
f(j,:) = (w'*x)/sum(w);
w1 = ((sc-t(j))/h).*normpdf((sc-t(j))/h)/h^2;
fdot(j,:) = (w1'*x)/sum(w) - (w'*x)*sum(w1)/sum(w)^2;
end
% Update scores
for i = 1:n
InnProd = sum((repmat(x(i,:),[m 1])-f).*fdot,2);
D = sum((repmat(x(i,:),[m 1])-f).^2,2);
[minD,jmin] = min(D);
sc(i) = t(jmin);
fsc(i,:) = f(jmin,:);
DX(i) = norm(x(i,:)-fsc(i,:));
end
err = abs(mean(DX)-mean(DX0))/mean(DX0);
disp(['Iteration ' num2str(iter) ', Mean DX = ' num2str(mean(DX)) ', Error ' num2s
if p==2
plot(x(:,1),x(:,2),'.',fsc(:,1),fsc(:,2),'.r')
pause(.3)
end
end


\end{appendices}
```

B B-Splines

```
function y = bspl(x,k,t,r)
%function y = bspl(x,k,t,r)
%
%B-spline basis functions and their derivatives
%
%INPUT:
%   x   (m x 1 or 1 x m)    Input grid.
%   k   (scalar)             Spline order.
%   t   (n x 1 or 1 x n)    Knots, must be a strictly increasing sequence
%                              and must INCLUDE interval endpoints.
%   r (scalar)            Order of derivative.
%
%OUTPUT:
%   y   (m x n+k-2)         Basis function (or derivative) values at X
%
% Version: May 2010

if nargin<4
error('Not enough input arguments')
end
if size(t,1)>1
t = t';
end

m = length(x);
n = length(t);
y = zeros(m,n+k-2);

if r==0

tt = [t(1)*ones(1,k-1), t, t(n)*ones(1,k-1)];
n = length(tt);
b = zeros(1,k);
dr = zeros(1,k-1);
dl = zeros(1,k-1);
for l = 1:m
```

```
b(1) = 1;
i = find(tt<=x(l),1,'last');
if i==n, i = n-k; end
for j = 1:k-1
%x
%tt
dr(j) = tt(i+j)-x(l);
dl(j) = x(l)-tt(i+1-j);
saved = 0;
for o = 1:j
term = b(o)/(dr(o)+dl(j+1-o));
b(o) = saved + dr(o)*term;
saved = dl(j+1-o)*term;
end
b(j+1) = saved;
end
y(l,i-k+1:i) = b;
end

else

tt = [repmat(t(1),1,k-2), t, repmat(t(n),1,k-2)];
B = bspl(x,k-1,t,r-1);
msp = ((k-1)./(ones(m,1)*(tt(k:n+2*(k-2))-tt(1:n+k-3)))).*B;
y(:,1) =  - msp(:,1);
y(:,2:n+k-3) = msp(:,1:n+k-4) - msp(:,2:n+k-3);
y(:,n+k-2) =  msp(:,n+k-3);

end
```

C Spline Estimation

```
function [f] = Main(x,kn,d,lambda)
s=size(x);
k=linspace(0,1,kn+2);
t=(1:s(1))'/(s(1)+1);
smooth= zeros(d+kn,d+kn);
beta=bspl(t,d,k,0); %% spline vektor \beta
t0 = linspace(0,1,500);
dbeta=bspl(t0,d,k,2);
for i=1:d+kn-2
for j=1:d+kn-2
smooth(i,j)= sum(dbeta(:,i).*dbeta(:,j)*(t0(2)-t0(1)));
end
end
a=sola(x,t,k,d,lambda,smooth);
iter = 0;
relerr = 1;
while relerr>0.001 && iter<100
iter = iter + 1;
aold=a;
t=solt(x,k,d,t,aold,0);
a=sola(x,t,k,d,lambda,smooth);
relerr = norm(a-aold)/norm(aold);
%f=bspl(t,d,k,0)*a+a(:,1)'*smooth*a(:,1)+a(:,2)'*smooth*a(:,2)
disp(['Iter: ' num2str(iter) ', RelErr: ' num2str(relerr)])

end
told=t;
t=solt(x,k,d,t,a,1);
f=bspl(t,d,k,0)*a;
end
```

```
function a= sola(x,t,k,d,lambda,smooth)
%% input x which is the data vector (n,p)
%% beta are the basisfunctions
%% kn is the number of knots for the splines
%% t are the points which gives us the other algorithm %%
s=size(x);
beta=bspl(t,d,k,0); %% spline vektor \beta
%%% finding optimal as (without \lambda)
a= (beta'*beta+lambda*smooth)\(beta'*x);
end
```

```
function t = solt(x,k,d,t,a,p)
Iterr=0;
n=1;
s= size(x);
f=bspl(t,d,k,0)*a;
df=bspl(t,d,k,1)*a;
d2f=bspl(t,d,k,2)*a;
%%% defining the function g which we have to set equal to zero %%%
g=df(:,1).*(x(:,1)-f(:,1))+df(:,2).*(x(:,2)-f(:,2));
dg= d2f(:,1).*(x(:,1)-f(:,1))-df(:,1).^2+ d2f(:,2).*(x(:,2)-f(:,2))-df(:,2).^2;
%first loop
if p==0
for i=1:s(1)

tnew=t(i)-g(i)/dg(i);
if tnew>=0 && tnew<=1
t(i) = tnew;
end
f(i,:)=bspl(t(i),d,k,0)*a;
df(i,:)=bspl(t(i),d,k,1)*a;
d2f(i,:)=bspl(t(i),d,k,2)*a;
g=df(:,1).*(x(:,1)-f(:,1))+df(:,2).*(x(:,2)-f(:,2));
dg= d2f(:,1).*(x(:,1)-f(:,1))-df(:,1).^2+ d2f(:,2).*(x(:,2)-f(:,2))-df(:,2).^2;


end
%% final loop %%
else
Iterr=0;
tnew=-1;
for i=1:s(1)
while abs(g(i))>0.0001 && Iterr < 100
while n<4 && tnew ~= t(i)
tnew=t(i)-g(i)/(dg(i)*2^n);
if tnew>=0 && tnew<=1
t(i) = tnew;
end
n=n+1;
```

```
end
n=1;
Iterr=Iterr+1;
end
Iterr=0;
f(i,:)=bspl(t(i),d,k,0)*a;
df(i,:)=bspl(t(i),d,k,1)*a;
d2f(i,:)=bspl(t(i),d,k,2)*a;
g=df(:,1).*(x(:,1)-f(:,1))+df(:,2).*(x(:,2)-f(:,2));
dg= d2f(:,1).*(x(:,1)-f(:,1))-df(:,1).^2+ d2f(:,2).*(x(:,2)-f(:,2))-df(:,2).^2;
end
end
end
```