PRINCIPAL CURVES: LEARNING, DESIGN, AND APPLICATIONS

BALÁZS KÉGL

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

DECEMBER 1999

0-612-47714-2

To my parents and Agnès

Abstract

Principal Curves: Learning, Design, and Applications

Balázs Kégl, Ph.D.
Concordia University, 2000

The subjects of this thesis are unsupervised learning in general, and principal curves in particular. Principal curves were originally defined by Hastie [Has84] and Hastie and Stuetzle [HS89] (hereafter HS) to formally capture the notion of a smooth curve passing through the "middle" of a d-dimensional probability distribution or data cloud. Based on the definition, HS also developed an algorithm for constructing principal curves of distributions and data sets.

The field has been very active since Hastie and Stuetzle's groundbreaking work. Numerous alternative definitions and methods for estimating principal curves have been proposed, and principal curves were further analyzed and compared with other unsupervised learning techniques. Several applications in various areas including image analysis, feature extraction, and speech processing demonstrated that principal curves are not only of theoretical interest, but they also have a legitimate place in the family of practical unsupervised learning techniques.

Although the concept of principal curves as considered by HS has several appealing characteristics, complete theoretical analysis of the model seems to be rather hard. This motivated us to redefine principal curves in a manner that allowed us to carry out extensive theoretical analysis while preserving the informal notion of principal curves. Our first contribution to the area is, hence, a new *theoretical model* that is analyzed by using tools of statistical learning theory. Our main result here is the first known consistency proof of a principal curve estimation scheme.

The theoretical model proved to be too restrictive to be practical. However, it inspired the design of a new *practical algorithm* to estimate principal curves based on data. The polygonal line algorithm, which compares favorably with previous methods both in terms of performance and computational complexity, is our second contribution to the area of principal curves. To complete the picture, in the last part of the thesis we consider an *application* of the polygonal line algorithm to hand-written character skeletonization.

Acknowledgments

I would like to express my deep gratitude to my advisor, Adam Krzyżak, for his help, trust and invaluable professional support. He suggested the problem, and guided me through the stages of this research. My great appreciation goes to Tamás Linder for leading me through the initial phases of this project, for the fruitful discussions on both the theoretical and the algorithmic issues, and for his constant support in pursuing my ideas. I would also like to thank Tony Kasvand for showing me the initial directions in the skeletonization project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The subjects of this thesis are unsupervised learning in general, and principal curves in particular. It is not intended to be a general survey of unsupervised learning techniques, rather a biased overview of a carefully selected collection of models and methods from the point of view of principal curves. It can also be considered as a case study of bringing a new baby into the family of unsupervised learning techniques, describing her genetic relationship with her ancestors and siblings, and indicating her potential prospects in the future by characterizing her talents and weaknesses. We start the introduction by portraying the family.

1.1 Unsupervised Learning

It is a common practice in general discussions on machine learning to use the dichotomy of supervised and unsupervised learning to categorize learning methods. From a conceptual point of view, supervised learning is substantially simpler than unsupervised learning. In supervised learning, the task is to guess the value of a random variable Y based on the knowledge of a d-dimensional random vector $\mathbf{X}$. The vector $\mathbf{X}$ is usually a collection of numerical observations such as a sequence of bits representing the pixels of an image, and Y represents an unknown nature of the observation such as the numerical digit depicted by the image. If Y is discrete, the problem of guessing Y is called *classification*. Predicting Y means finding a function $f : \mathbb{R}^d \to \mathbb{R}$ such that $f(\mathbf{X})$ is close to Y where "closeness" is measured by a non-negative cost function $q(f(\mathbf{X}),Y)$. The task is then to find a function that minimizes the expected cost, that is,

$$f^* = \arg\min_f E[q(f(\mathbf{X}),Y)].$$

In practice, the joint distribution of $\mathbf{X}$ and Y is usually unknown, so finding f^* analytically is impossible. Instead, we are given $\mathcal{X}_n = \{(\mathbf{X}_1,Y_1),\ldots,(\mathbf{X}_n,Y_n)\} \subset \mathbb{R}^d \times \mathbb{R}$, a sample of n independent and identical copies of the pair $(\mathbf{X},Y)$, and the task is to find a function $\hat{f}_n(\mathbf{X}) = \hat{f}(\mathbf{X},\mathcal{X}_n)$ that predicts Y as well as possible *based on the data set* $\mathcal{X}_n$. The problem is well-defined in the sense that the performance of a predictor $\hat{f}_n$ can be quantified by its *test error*, the average cost measured on an independent test set $\mathcal{X}'_m = \{(\mathbf{X}'_1,Y'_1),\ldots,(\mathbf{X}'_m,Y'_m)\}$ defined by

$$q(\hat{f}) = \frac{1}{m}\sum_{i=1}^{m} q(\hat{f}(\mathbf{X}'_i),Y'_i).$$

As a consequence, the best of two given predictors $\hat{f}_1$ and $\hat{f}_2$ can be chosen objectively by comparing $q(\hat{f}_1)$ and $q(\hat{f}_2)$ on a sufficiently large test sample.

Unfortunately, this is not the case in unsupervised learning. In a certain sense, an unsupervised learner can be considered as a supervised learner where the label Y of the observation $\mathbf{X}$ is the observation itself. In other words, the task is to find a function $f : \mathbb{R}^d \to \mathbb{R}^d$ such that $f(\mathbf{X})$ predicts $\mathbf{X}$ as well as possible. Of course, without restricting the set of admissible predictors, this is a trivial problem. The source of such restrictions is the other objective of unsupervised learning, namely, to represent the mapping $f(\mathbf{X})$ of $\mathbf{X}$ with as few parameters as possible. These two competing objectives of unsupervised learning are called *information preservation* and *dimension reduction*. The trade-off between the two competing objectives depends on the particular problem. What makes unsupervised learning ill-defined in certain applications is that the trade-off is often not specified in the sense that it is possible to find two admissible functions $\hat{f}_1$ and $\hat{f}_2$ such that $\hat{f}_1$ predicts $\mathbf{X}$ better than $\hat{f}_2$, $\hat{f}_2$ compresses $\mathbf{X}$ more efficiently than $\hat{f}_1$, and there is no objective criteria to decide which function performs better *overall*.

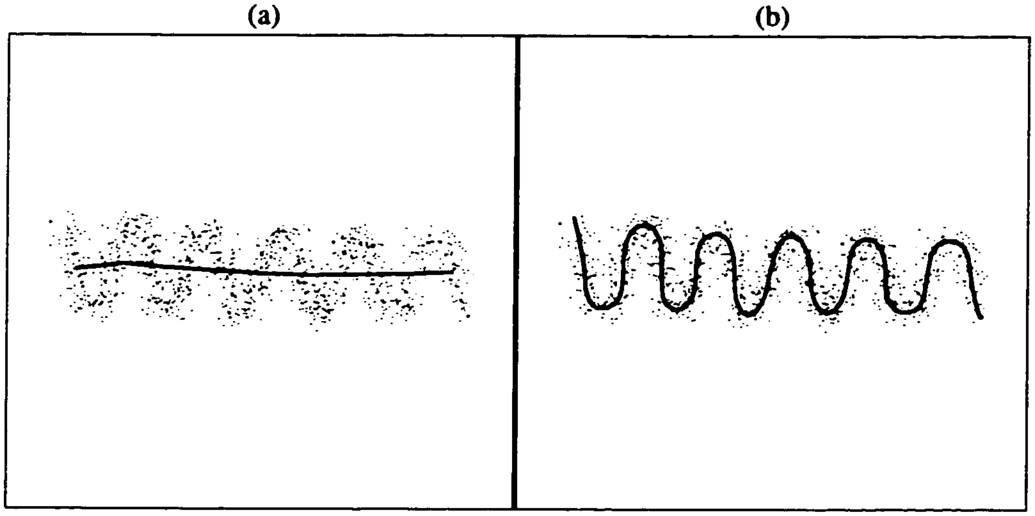


Figure 1: An ill-defined unsupervised learning problem. Which curve describes the data better, (a) a short curve that is "far" from the data, or a (b) long curve that follows the data more closely?

To clarify this ambiguity intrinsic to the unsupervised learning model, consider the simple example depicted by Figure 1. Both images show the same data set and two smooth curves intended to represent the data set in a concise manner. Using the terminology introduced above, f is a function that maps every point in the plane to its projection point on the representing curve. Hence, in this case, the first objective of unsupervised learning means that the representing curve should go through the data cloud as close to the data points, on average, as possible. Obviously, if this is the only objective, then the solution is a "snake" that visits all the data points. For restricting the set of admissible curves, several regularity conditions can be considered. For instance, one can require that the curve be as smooth as possible, or one can enforce a length limit on the representing curve. If the length limit is hard, i.e., the length of the curve *must* be less or equal to a predefined threshold, the problem is well-defined in the sense that the curve that minimizes the average distance from the data cloud exists. In practice, however, it is hard to prescribe such a hard limit. Instead, the length constraint is specified as a soft limit, and the informal objective can be formulated as "find a curve which is as short as possible and which goes through the data as close to the data points, on average, as possible". This "soft" objective clearly makes the problem ill-defined in the sense that without another principle that decides the actual mixing proportion of the two competing objectives, one cannot choose the best of two given representing curve. In our example, we need an outside source that decides between a shorter curve that is farther form the data (Figure 1(a)), or a longer curve that follows the data more closely (Figure 1(b)).

The reason of placing this discussion even before the formal statement of the problem is that it determines our philosophy in developing general purpose unsupervised methods. Since the general problem of unsupervised learning is ill-defined, "turnkey" algorithms cannot be designed. Every unsupervised learning algorithm must come with a set of parameters that can be used to adjust the algorithm to a particular problem or according to a particular principle. From the point of view of the engineer who uses the algorithm, the number of such parameters should be as small as possible, and their effect on the behavior of the algorithm should be as clear as possible.

The intrinsic ambiguity of the unsupervised learning model also limits the possibilities of the theoretical analysis. On the one hand, without imposing some restrictive conditions on the model, it is hard to obtain any meaningful theoretical results. On the other hand, to allow theoretical analysis, these conditions may be so that the model does not exactly refer to any specific practical problem. Nevertheless, it is useful to obtain such results to deepen our insight to the model and also to inspire the development of theoretically well founded practical methods.

In the rest of the section we describe the formal model of unsupervised learning, outline some of the application areas, and briefly review the possible areas of theoretical analysis.

1.1.1 The Formal Model

For the formal description of the problem of unsupervised learning, let $\mathbb{D}$ be the domain of the data and let $\mathcal{F}$ be the set of functions of the form $f : \mathbb{D} \to \mathbb{R}^d$. For each $f \in \mathcal{F}$ we call the range of f the *manifold* generated by f, i.e.,

$$\mathcal{M}_f = f(\mathbb{D}) = \{f(\mathbf{x}) : \mathbf{x} \in \mathbb{D}\}.$$

The set of all manifolds generated by all functions in $\mathcal{F}$ is denoted by $\mathbb{M}$, i.e., we define

$$\mathbb{M} = \{\mathcal{M}_f : f \in \mathcal{F}\}.$$

To measure the distortion caused by the mapping of $\mathbf{x} \in \mathbb{D}$ into $\mathcal{M}_f$ by the function f, we assume that a *distance* $\Delta(\mathcal{M}, \mathbf{x})$ is defined for every $\mathcal{M} \in \mathbb{M}$ and $\mathbf{x} \in \mathbb{D}$. Now consider a random vector $\mathbf{X} \in \mathbb{D}$. The *distance function* or the *loss* of a manifold $\mathcal{M}$ is defined as the expected distance between $\mathbf{X}$ and $\mathcal{M}$, that is,

$$\Delta(\mathcal{M}) = E\left[\Delta(\mathbf{X}, \mathcal{M})\right].$$

The general objective of unsupervised learning is to find a manifold $\mathcal{M}$ such that $\Delta(\mathcal{M})$ is small and $\mathcal{M}$ has a low complexity relative to the complexity of $\mathbb{D}$. The first objective guarantees that the information stored in $\mathbf{X}$ is preserved by the projection whereas the second objective means that $\mathbf{M}$ is an efficient representation of $\mathbf{X}$.

1.1.2 Areas Of Applications

The general model of unsupervised learning has been defined, analyzed, and applied in many different areas under different names. Some of the most important application areas are the following.

- *Clustering* or *taxonomy* in multivariate data analysis [Har75, JD88]. The task is to find a usually hierarchical categorization of entities (for example, species of animals or plants) on the basis of their similarities. It is similar to supervised classification in the sense in that both methods aim to categorize $\mathbf{X}$ into a finite number of classes. The difference is that in a supervised model, the classes are predefined while here the categories are unknown so they must be created during the process.

- *Feature extraction* in pattern recognition [DK82, DGL96]. The objective is to find a relatively small number of features that represent $\mathbf{X}$ well in the sense that they preserve most of the variance of $\mathbf{X}$. Feature extraction is usually used as a pre-processing step before classification to accelerate the learning by reducing the dimension of the input data. Preserving the information stored in $\mathbf{X}$ is important to keep the Bayes error (the error that represents the confusion inherently present in the problem) low.

- *Lossy data compression* in information theory [GG92]. The task is to find an efficient representation of $\mathbf{X}$ for transmitting it through a communication channel or storing it on a storage device. The more efficient the compression, the less time is needed for transmission. Keeping the expected distortion low means that the recovered data at the receiving end resembles the original.

- *Noise reduction* in signal processing [VT68]. It is usually assumed here that $\mathbf{X}$ was generated by a latent additive model,

$$\mathbf{X} = \mathbf{M} + \varepsilon, \tag{1}$$

  where $\mathbf{M}$ is a random vector concentrated to the manifold $\mathcal{M}$, and ε is an independent multivariate random noise with zero mean. The task is to recover $\mathbf{M}$ based on the noisy observation $\mathbf{X}$.

- *Latent-variable models* [Eve84, Mac95, BSW96]. It is presumed that $\mathbf{X}$, although sitting in a high-dimensional space, has a low intrinsic dimension. This is a special case of (1) when the additive noise is zero or nearly zero. In practice, $\mathcal{M}$ is usually highly nonlinear otherwise the problem is trivial. When $\mathcal{M}$ is two-dimensional, using $\mathbf{M}$ for representing $\mathbf{X}$ can serve as an effective visualization tool [Sam69, KW78, BT98].

- *Factor analysis* [Eve84, Bar87] is another special case of (1) when $\mathbf{M}$ is assumed to be a Gaussian random variable concentrated on a linear subspace of $\mathbb{R}^d$, and ε is a Gaussian noise with diagonal covariance matrix.

1.1.3 The Simplest Case

In simple unsupervised models the set of admissible functions $\mathcal{F}$ or the corresponding set of manifolds $\mathbb{M}$ is given independently of the distribution of $\mathbf{X}$. $\mathcal{F}$ is a set of simple functions in the sense that any $f \in \mathcal{F}$ or the corresponding $\mathcal{M}_f \in \mathbb{M}$ can be represented by a few parameters. It is also assumed that any two manifolds in $\mathbb{M}$ have the same intrinsic dimension, so the only objective in this model is to minimize $\Delta(\mathcal{M})$ over $\mathbb{M}$, i.e., to find

$$\mathcal{M}^* = \arg\min_{\mathcal{M}\in\mathbb{M}} E\left[\Delta(\mathbf{X},\mathcal{M})\right].$$

Similarly to supervised learning, the distribution of $\mathbf{X}$ is usually unknown in practice. Instead, we are given $\mathcal{X}_n = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\} \subset \mathbb{R}^d$, a sample of n independent and identical copies of $\mathbf{X}$, and the task is to find a function $\hat{f}_n(\mathbf{X}) = \hat{f}(\mathbf{X}, \mathcal{X}_n)$ *based on the data set* $\mathcal{X}_n$ that minimizes the distance function. Since the the distribution of $\mathbf{X}$ is unknown, we estimate $\Delta(\mathcal{M})$ by the *empirical distance* *function* or *empirical loss* of $\mathcal{M}$ defined by

$$\Delta_n(\mathcal{M}) = \frac{1}{n}\sum_{i=1}^{n}\Delta(\mathbf{X}_i, \mathcal{M}). \tag{2}$$

The problem is well-defined in the sense that the performance of a projection function $\hat{f}_n$ can be quantified by the empirical loss of $\hat{f}_n$ measured on an independent test set $\mathcal{X}'_m = \{\mathbf{X}'_1, \ldots, \mathbf{X}'_m\}$. As a consequence, the best of two given projection functions $\hat{f}_1$ and $\hat{f}_2$ can be chosen objectively by comparing $\Delta_n(\mathcal{M}_{\hat{f}_1})$ and $\Delta_n(\mathcal{M}_{\hat{f}_2})$ on a sufficiently large test sample.

In the theoretical analysis of a particular unsupervised model, the first question to ask is

$$\text{“\textit{Does } } \mathcal{M}^* \text{\textit{ exist in general?}”} \tag{Q1}$$

Clearly, if $\mathcal{M}^*$ does not exist, or it only exists under severe restrictions, the theoretical analysis of any estimation scheme based on finite data is difficult. If $\mathcal{M}^*$ does exist, the next two obvious questions are

$$\text{“\textit{Is} } \mathcal{M}^* \text{ \textit{unique?}”} \tag{Q2}$$

and

$$\text{“\textit{Can we show a concrete example of} } \mathcal{M}^*\text{\textit{?}”} \tag{Q3}$$

Interestingly, even for some of the simplest unsupervised learning models, the answer to Question 3 is no for even the most common multivariate densities. Note, however, that this fact does not make the theoretical analysis of an estimating scheme impossible, and does not make it unreasonable to aim for the optimal loss $\Delta(\mathcal{M}^*)$ in practical estimator design.

The most widely used principle in designing nonparametric estimation schemes is the *empirical loss minimization principle*. In unsupervised learning this means that based on the data set $\mathcal{X}_n$, we pick the manifold $\mathcal{M}_n^* \in \mathbb{M}$ that minimizes the empirical distance function (2), i.e., we choose

$$\mathcal{M}_n^* = \arg\min_{\mathcal{M}\in\mathbb{M}} \frac{1}{n}\sum_{i=1}^{n}\Delta(\mathbf{X}_i, \mathcal{M}). \tag{3}$$

The first property of $\mathcal{M}_n^*$ to analyze is its *consistency*, i.e. the first question is

$$\text{“\textit{Is} } \lim_{n\to\infty}\Delta(\mathcal{M}_n^*) = \Delta(\mathcal{M}^*) \text{ \textit{in probability?}”} \tag{Q4}$$

Consistency guarantees that by increasing the amount of data, the expected loss of $\mathcal{M}_n^*$ gets arbitrarily close to the best achievable loss. Once consistency is established, the next natural question is

$$\text{“\textit{What is the convergence rate of} } \Delta(\mathcal{M}_n^*) \to \Delta(\mathcal{M}^*)\text{\textit{?}”} \tag{Q5}$$

A good convergence rate is important to establish upper bounds for the probability of error for a given data size. From a practical point of view, the next question is

$$\text{“\textit{Is there an efficient algorithm to find} } \mathcal{M}_n^* \text{ \textit{given a data set} } \mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}\text{?”} \tag{Q6}$$

To illustrate this general analysis scheme, we turn to the simplest possible unsupervised learning method. Let the admissible manifolds $\mathcal{M}$ be arbitrary points of the d-dimensional space ($\mathbb{M} = \mathbb{R}^d$), and assume that $\Delta(\mathcal{M}, \mathbf{X})$ is the squared Euclidean distance of $\mathcal{M}$ and $\mathbf{X}$, i.e.,

$$\Delta(\mathcal{M}, \mathbf{X}) = \|\mathcal{M} - \mathbf{X}\|^2.$$

To find $\mathcal{M}_n^*$, we have to minimize $E[\Delta(\mathcal{M}, \mathbf{X})]$ over all $\mathcal{M} \in \mathbb{R}^d$. It is a well known fact that $E[\|\mathcal{M} - \mathbf{X}\|^2]$ is minimized by $E[\mathbf{X}]$ so we have

$$\mathcal{M}^* = E[\mathbf{X}].$$

The answer to all the first three questions is, therefore, yes. According to the empirical loss minimization principle, given $\mathcal{X}_n = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\} \subset \mathbb{R}^d$, a sample of n independent and identical copies of $\mathbf{X}$, the estimator $\mathcal{M}_n^* \in \mathbb{M}$ is the vector that minimizes the empirical loss, i.e.,

$$\mathcal{M}_n^* = \arg\min_{\mathcal{M} \in \mathbb{M}} \frac{1}{n} \sum_{i=1}^{n} \|\mathcal{M} - \mathbf{X}_i\|^2.$$

It is easy to see that the minimizing vector is the *sample mean* or *center of gravity* of $\mathcal{X}_n$, i.e.,

$$\mathcal{M}_n^* = \frac{1}{n} \sum_{i=1}^{n} \mathbf{X}_i.$$

Consistency of $\mathcal{M}_n^*$ follows from the law of large numbers. For the convergence rate note that, if $\mathbf{X}$ has finite second moments,

$$\begin{aligned} \Delta(\mathcal{M}_n^*) - \Delta(\mathcal{M}^*) &= E\left[\|\mathbf{X} - \mathcal{M}_n^*\|^2\right] - E\left[\|\mathbf{X} - \mathcal{M}^*\|^2\right] \\ &= \left(1 + \frac{1}{n}\right)\sigma_{\mathbf{X}}^2 - \sigma_{\mathbf{X}}^2 \\ &= \frac{1}{n}\sigma_{\mathbf{X}}^2 \end{aligned} \tag{4}$$

where $\sigma_{\mathbf{X}}^2$ is the sum of the variances of the components of $\mathbf{X}$. Hence, if $\mathbf{X}$ has finite second moments, the rate of convergence is $\frac{1}{n}$.

Simple unsupervised learning methods of this type are Vector Quantization (admissible manifolds are finite sets of d-dimensional vectors), and Principal Component Analysis (admissible manifolds are linear subspaces of $\mathbb{R}^d$). We analyze these methods in Chapter 2. Theoretical analysis of principal curves with a length constraint in Chapter 4 also follows these lines.

1.1.4 A More Realistic Model

Although amenable for theoretical analysis, simple methods described above are often impractical. In one group of methods the strict restrictions imposed on the admissible manifolds result in that manifolds of $\mathbb{M}$ are not able to describe highly nonlinear data. Unfortunately, there is a problem even if admissible manifolds are rich enough to capture complex data. The problem is that in the simple model described above, the set of admissible manifolds must be specified *independently of the particular application*. Given a set of manifolds $\mathbb{M}$, it is possible that in a certain application $\mathbb{M}$ is too rich, while in another problem manifolds in $\mathbb{M}$ are too simple to capture the data. This problem can be solved by allowing the practitioner to choose from several classes of manifolds of different complexity. Assume, therefore, that a nested sequence of manifold model classes $\mathbb{M}^{(1)} \subset \mathbb{M}^{(2)} \subset \ldots$ is given, such that for a given $j = 1, \ldots$, the intrinsic dimensions of all manifolds in $\mathbb{M}^{(j)}$ are the same. Let c_j be a real number that measures the intrinsic dimension of manifolds in $\mathbb{M}^{(j)}$, such that $c_1 < c_2 < \ldots$. We can also say that c_j measures the *complexity* of $\mathbb{M}^{(j)}$. To define the theoretically best manifold, one can follow the following strategy. Find the optimal manifold in each class to obtain the sequence of manifolds $\mathcal{M}^{(1)^*}, \mathcal{M}^{(2)^*}, \ldots$. Then using a principle corresponding to the particular problem, select the manifold $\mathcal{M}^{(j^*)^*}$ from the j^*th model class that represents the data the best[1].

The same questions can be asked in this complex model as in the simple model described in the previous section. The existence of $\mathcal{M}^{(j^*)^*}$ depends on two conditions. First, optimal manifolds $\mathcal{M}^{(1)^*}, \mathcal{M}^{(2)^*}, \ldots$ must exist for all model classes. Second, the principle that governs the selection of $\mathcal{M}^{(j^*)^*}$ (by choosing the model class j^*) must be well-defined in the sense that it gives a total order over the set of optimal manifolds $\mathcal{M}^{(1)^*}, \mathcal{M}^{(2)^*}, \ldots$.

Estimating $\mathcal{M}^{(j^*)^*}$ can be done by combining the empirical loss minimization principle with the model selection technique described above. Accordingly, one can choose the empirically best manifold for each model class to obtain the sequence $\mathcal{M}_n^{(1)^*}, \mathcal{M}_n^{(2)^*}, \ldots$, and then use the principle corresponding to the particular problem to select the best manifold $\mathcal{M}_n^{(j_n^*)^*}$. Consistency analysis of the model is usually rather hard as one has to not only establish consistency in the model classes, but also to show that when the data size is large, the model class j^* selected in the theoretical model is the same as the model class j_n^* selected in the estimation.

To further complicate the situation, it is usually impractical to follow this scheme since it requires to find the empirically best manifold in several model classes. Instead, practical algorithms usually optimize the two criteria *at the same time*. In most of the algorithms, although not in all of

[1] Note that this approach resembles the method of *complexity regularization* [DGL96] or *structural risk minimization* [Vap98] used in supervised learning. There is a fundamental difference, however. While in supervised learning, complexity regularization is used in the *estimation* phase, here, we use it to define the *theoretically* best manifold. The reason, again, is that the general unsupervised learning problem is inherently ill-posed.

them, the two criteria are combined in one "energy function" of the form

$$G_n(\mathcal{M}) = \Delta_n(\mathcal{M}) + \lambda P(\mathcal{M})$$

where $\Delta_n(\mathcal{M})$ is the empirical distance function of $\mathcal{M}$, as usual, $P(\mathcal{M})$ is a *penalty* or *regularizer* term which penalizes the complexity of the manifold, and λ is a penalty coefficient that determines the trade-off between the accuracy of the approximation and the smoothness of the manifold. The algorithm proceeds by minimizing $G_n(\mathcal{M})$ over all admissible manifolds. In Chapter 3, we present several methods that follow this scheme.

1.2 Principal Curves

The main subject of this thesis is the analysis and applications of principal curves. Principal curves were originally defined by Hastie [Has84] and Hastie and Stuetzle [HS89] (hereafter HS) to formally capture the notion of a smooth curve passing through the "middle" of a d-dimensional probability distribution or data cloud (to form an intuitive image, see Figure 1 on page 2). The original HS definition of principal curves is based on the concept of *self-consistency*. Intuitively, self-consistency means that each point of the curve is the average of all points that project there. Based on the self-consistency property, HS developed a theoretical and a practical algorithm for constructing principal curves of distributions and data sets, respectively.

The field has been very active since Hastie and Stuetzle's groundbreaking work. Numerous alternative definitions and methods for estimating principal curves have been proposed, and principal curves were further analyzed and compared with other unsupervised learning techniques. Several applications in various areas including image analysis, feature extraction, and speech processing demonstrated that principal curves are not only of theoretical interest, but they also have a legitimate place in the family of practical unsupervised learning techniques.

Although the concept of principal curves as considered by HS has several appealing characteristics, complete theoretical analysis of the model seems to be rather hard. This motivated us to redefine principal curves in a manner that allowed us to carry out extensive theoretical analysis while preserving the informal notion of principal curves. Our first contribution to the area is, hence, a new *theoretical model* that can be analyzed along the lines of the general unsupervised learning model described in the previous section. Our main result here is the first known consistency proof of a principal curve estimation scheme.

The theoretical model proved to be too restrictive to be practical. However, it inspired the design of a new *practical algorithm* to estimate principal curves based on data. The polygonal line algorithm, which compares favorably with previous methods both in terms of performance and computational complexity, is our second contribution to the area of principal curves. To complete the picture, in the last part of the thesis we consider an *application* of the polygonal line algorithm to hand-written character skeletonization. We note here that parts of our results have been previously published in [KKLZ98], [KKLZ99], and [KKLZ00].

1.3 Outline of the Thesis

Most of the unsupervised learning algorithms originate from one of the two basic unsupervised learning models, vector quantization and principal component analysis. In Chapter 2 we describe these two models. In Chapter 3, we present the formal definition of the HS principal curves, describe the subsequent extensions and analysis, and discuss the relationship between principal curves and other unsupervised learning techniques.

An unfortunate property of the HS definition is that, in general, it is not known if principal curves exist for a given distribution. This also makes it difficult to theoretically analyze any estimation scheme for principal curves. In Chapter 4 we propose a new definition of principal curves and prove the existence of principal curves in the new sense for a large class of distributions. Based on the new definition, we consider the problem of learning principal curves based on training data. We introduce and analyze an estimation scheme using a common model in statistical learning theory. The main result of this chapter is a proof of consistency and analysis of rate of convergence following the general scheme described in Section 1.1.

Although amenable to analysis, our theoretical algorithm is computationally burdensome for implementation. In Chapter 5 we develop a suboptimal algorithm for learning principal curves. The polygonal line algorithm produces piecewise linear approximations to the principal curve, just as the theoretical method does, but global optimization is replaced by a less complex gradient-based method. We give simulation results and compare our algorithm with previous work. In general, on examples considered by HS, the performance of the new algorithm is comparable with the HS algorithm while it proves to be more robust to changes in the data generating model.

Chapter 6 starts with an overview of existing principal curve applications. The main subject of this chapter is an application of an extended version of the principal curve algorithm to hand-written character skeletonization. The development of the method was inspired by the apparent similarity between the definition of principal curves and the medial axis of a character. A principal curve is a smooth curve that goes through the "middle" of a data set, whereas the medial axis is a set of smooth curves that go equidistantly from the contours of a character. Since the medial axis can be a *set* of connected curves rather then only *one* curve, in Chapter 6 we extend the polygonal line algorithm to find a principal graph of a data set. The extended algorithm also contains two elements specific to the task of skeletonization, an initialization method to capture the approximate topology of the character, and a collection of restructuring operations to improve the structural quality of the skeleton produced by the initialization method. Test results on isolated hand-written digits indicate that the algorithm finds a smooth medial axis of the great majority of a wide variety of character templates. Experiments with images of continuous handwriting demonstrate that the skeleton graph produced by the algorithm can be used for representing hand-written text efficiently.

Chapter 2

Vector Quantization and Principal Component Analysis

Most of the unsupervised learning algorithms originate from one of the two basic unsupervised learning models, vector quantization and principal component analysis. In particular, principal curves are related to both areas: conceptually, they are originated from principal component analysis whereas practical methods to estimate principal curves often resemble to basic vector quantization algorithms. This chapter describes these two models.

2.1 Vector Quantization

Vector quantization is an important topic of information theory. Vector quantizers are used in lossy data compression, speech and image coding [GG92], and clustering [Har75]. Vector quantization can also be considered as the simplest form of unsupervised learning where the manifold to fit to the data is a set of vectors. Kohonen's self-organizing map [Koh97] (introduced in Section 3.2.2) can also be interpreted as a generalization of vector quantization. Furthermore, our new definition of principal curves (to be presented in in Section 4.1) has been inspired by the notion of an *optimal vector quantizer*. One of the most widely used algorithms for constructing locally optimal vector quantizers for distributions or data sets is the Generalized Lloyd (GL) algorithm [LBG80] (also known as the k-means algorithm [Mac67]). Both the HS algorithm (Section 3.1.1) and the polygonal line algorithm (Section 5.1) are similar in spirit to the GL algorithm. This section introduces the concept of optimal vector quantization and describes the GL algorithm.

2.1.1 Optimal Vector Quantizer

A k-point vector quantizer is a mapping $q : \mathbb{R}^d \to \mathbb{R}^d$ that assigns to each input vector $\mathbf{x} \in \mathbb{R}^d$ a *codepoint* $\hat{\mathbf{x}} = q(\mathbf{x})$ drawn from a finite *codebook* $C = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subset \mathbb{R}^d$. The quantizer is completely described by the codebook C together with the partition $\mathcal{V} = \{V_1, \ldots, V_k\}$ of the input space where $V_\ell = q^{-1}(\mathbf{v}_\ell) = \{\mathbf{x} : q(\mathbf{x}) = \mathbf{v}_\ell\}$ is the set of input vectors that are mapped to the ℓth codepoint by q.

The distortion caused by representing an input vector $\mathbf{x}$ by a codepoint $\hat{\mathbf{x}}$ is measured by a non-negative *distortion measure* $\Delta(\mathbf{x}, \hat{\mathbf{x}})$. Many such distortion measures have been proposed in different areas of application. For the sake of simplicity, in what follows, we assume that $\Delta(\mathbf{x}, \hat{\mathbf{x}})$ is the most widely used squared error distortion, that is,

$$\Delta(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2. \tag{5}$$

The performance of a quantizer q applied to a random vector $\mathbf{X} = (X_1, \ldots, X_d)$ is measured by the expected distortion,

$$\Delta(q) = E[\Delta(\mathbf{X}, q(\mathbf{X}))] \tag{6}$$

where the expectation is taken with respect to the underlying distribution of $\mathbf{X}$. The quantizer q^* is *globally optimal* if $\Delta(q^*) \leq \Delta(q)$ for any k-point quantizer q. It can be shown that q^* exists if $\mathbf{X}$ has finite second moments, so the answer to Question 1 in Section 1.1.3 is yes. Interestingly, however, the answers to Questions 2 and 3 are no in general. Finding a globally optimal vector quantizer for a given source distribution or density is a very hard problem. Presently, for $k > 2$ codepoints there seem to be no concrete examples of optimal vector quantizers for even the most common model distributions such as Gaussian, Laplacian, or uniform (in a hypercube) in any dimensions $d > 1$.

Since global optimality is not a feasible requirement, algorithms, even in theory, are usually designed to find *locally optimal* vector quantizers. A quantizer q is said to be locally optimal if $\Delta(q)$ is only a local minimum, that is, slight disturbance of any of the codepoints will cause an increase in the distortion. Necessary conditions for local optimality will be given in Section 2.1.3. We also describe here a theoretical algorithm, the Generalized Lloyd (GL) algorithm [LBG80], to find a locally optimal vector quantizer of a random variable.

In practice, the distribution of $\mathbf{X}$ is usually unknown. Therefore, the objective of *empirical quantizer design* is to find a vector quantizer based on $\mathcal{X}_n = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$, a set of independent and identical copies of $\mathbf{X}$. To design a quantizer with low distortion, most existing practical algorithms attempt to implement the empirical loss minimization principle introduced for the general unsupervised learning model in Section 1.1.3. The performance of a vector quantizer q on $\mathcal{X}_n$ is measured by the *empirical distortion* of q given by

$$\Delta_n(q) = \frac{1}{n} \sum_{i=1}^{n} \Delta(\mathbf{X}_i, q(\mathbf{X}_i)). \tag{7}$$

The quantizer q_n^* is *globally optimal* on the data set $\mathcal{X}_n$ if $\Delta_n(q_n^*) \leq \Delta_n(q)$ for any k-point quantizer q. Finding an empirically optimal vector quantizer is, in theory, possible since the number of different partitions of $\mathcal{X}_n$ is finite. However, the systematic inspection of all different partitions is computationally infeasible. Instead, most practical methods use an iterative approach similar in spirit to the GL algorithm.

It is of both theoretical and practical interest to analyze how the expected loss of the empirically best vector quantizer $\Delta(q_n^*)$ relates to the best achievable loss $\Delta_n(q^*)$, even though q^* is not known and q_n^* is practically infeasible to obtain. Consistency (Question 4 in Section 1.1.3) of the estimation scheme means that the expected loss of the q_n^* converges in probability to the best achievable loss as the number of the data points grows, therefore, if we have a perfect algorithm and unlimited access to data, we can get arbitrarily close to the best achievable loss. A good convergence rate (Question 5) is important to establish upper bounds for the probability of error for a given data size. We start the analysis of the empirical loss minimization principle used for vector quantization design by presenting results on consistency and rate of convergence in Section 2.1.2.

2.1.2 Consistency and Rate Of Convergence

Consistency of the empirical quantizer design under general conditions was proven by Pollard [Pol81, Pol82]. The first rate of convergence results were obtained by Linder et al. [LLZ94]. In particular, [LLZ94] showed that if the distribution of **X** is concentrated on a bounded region, there exists a constant c such that

$$\Delta(q_n^*) - \Delta(q^*) \leq cd^{3/2}\sqrt{\frac{k\log n}{n}}. \tag{8}$$

An extension of this result to distributions with unbounded support is given in [MZ97]. Bartlett et al. [BLL98] pointed out that the $\sqrt{\log n}$ factor can be eliminated from the upper bound in (8) by using an analysis based on sophisticated uniform large deviation inequalities of Alexander [Ale84] or Talagrand [Tal94]. More precisely, it can be proven that there exists a constant c' such that

$$\Delta(q_n^*) - \Delta(q^*) \leq c'd^{3/2}\sqrt{\frac{k\log(kd)}{n}}. \tag{9}$$

There are indications that the upper bound can be tightened to $O(1/n)$. First, in (4) we showed that if $k = 1$, the expected loss of the sample average converges to the smallest possible loss at a rate of $O(1/n)$. Another indication that an $O(1/n)$ rate might be achieved comes from a result of Pollard [Pol82]. He showed if **X** has a specially smooth and regular density, the difference between the codepoints of the empirically designed quantizers and the codepoints of the optimal quantizer obeys a multidimensional central limit theorem. As Chou [Cho94] pointed out, this implies that that within the class of distributions considered by [Pol82], the distortion redundancy decreases at a rate

$O(1/n)$. Despite these suggestive facts, it was showed by [BLL98] that in general, the conjectured $O(1/n)$ distortion redundancy rate does not hold. In particular, [BLL98] proved that for any k-point quantizer q_n which is designed by *any* method from n independent training samples, there exists a distribution on a bounded subset of $\mathbb{R}^d$ such that the expected loss of q_n is bounded away from the optimal distortion by a constant times $1/\sqrt{n}$. Together with (9), this result shows that the minimax (worst-case) distortion redundancy for empirical quantizer design is asymptotically on the order of $1/\sqrt{n}$. As a final note, [BLL98] conjectures that the minimax expected distortion redundancy is some constant times

$$d^a\sqrt{\frac{k^{1-b/d}}{n}}$$

for some values of $a \in [1,3/2]$ and $b \in [2,4]$.

2.1.3 Locally Optimal Vector Quantizer

Suppose that we are given a particular codebook C but the partition is not specified. An optimal partition $\mathcal{V}$ can be constructed by mapping each input vector $\mathbf{x}$ to the codepoint $\mathbf{v}_\ell \in C$ that minimizes the distortion $\Delta(\mathbf{x},\mathbf{v}_\ell)$ among all codepoints, that is, by choosing the nearest codepoint to $\mathbf{x}$. Formally, $\mathcal{V} = \{V_1,\ldots,V_k\}$ is the optimal partition of the codebook C if

$$V_\ell = \{\mathbf{x} : \Delta(\mathbf{x},\mathbf{v}_\ell) \le \Delta(\mathbf{x},\mathbf{v}_m),\ m = 1,\ldots,k\}. \tag{10}$$

(A tie-breaking rule such as choosing the codepoint with the lowest index is required if more than one codepoint minimizes the distortion.) V_ℓ is called the *Voronoi region* or *Voronoi set* associated with the codepoint $\mathbf{v}_\ell$.

Conversely, assume that we are given a partition $\mathcal{V} = \{V_1,\ldots,V_k\}$ and an optimal codebook $C = \{\mathbf{v}_1,\ldots,\mathbf{v}_k\}$ is needed to be constructed. To minimize the expected distortion, we have to set

$$\mathbf{v}_\ell = \arg\min_{\mathbf{v}} E[\Delta(\mathbf{X},\mathbf{v})|\mathbf{X} \in V_\ell]. \tag{11}$$

$\mathbf{v}_\ell$ is called the *centroid* or the *center of gravity* of the set V_ℓ, motivated by the fact that for the squared error distortion (5) we have $\mathbf{v}_\ell = E[\mathbf{X}|\mathbf{X} \in V_\ell]$.

It can be shown that the *nearest neighbor condition* (10) and the *centroid condition* (11) must hold for any locally optimal vector quantizer. Another necessary condition of local optimality is that boundary points occur with zero probability, that is,

$$P\{\mathbf{X} : \mathbf{X} \in V_\ell, \Delta(\mathbf{X},\mathbf{v}_\ell) = \Delta(\mathbf{X},\mathbf{v}_m), \ell \ne m\} = 0. \tag{12}$$

If we have a codebook that satisfies all three necessary conditions of optimality, it is widely believed that it is indeed locally optimal. No general theoretical derivation of this result has ever been obtained. For the particular case of discrete distribution, however, it can be shown that under mild restrictions, a vector quantizer satisfying the three necessary conditions is indeed locally optimal [GKL80].

2.1.4 Generalized Lloyd Algorithm

The nearest neighbor condition and the centroid condition suggest a natural algorithm for designing a vector quantizer. The GL algorithm alternates between an expectation and a partition step until the relative improvement of the expected distortion is less than a preset threshold. In the expectation step the codepoints are computed according to (11), and in the partition step the Voronoi regions are set by using (10). It is assumed that an initial codebook $\mathcal{C}^{(0)}$ is given. When the probability density of $\mathbf{X}$ is known, the GL algorithm for constructing a vector quantizer is the following.

Algorithm 1 (The GL algorithm for distributions)

Step 0 *Set* $j = 0$, *and set* $\mathcal{C}^{(0)} = \left\{\mathbf{v}_1^{(0)}, \ldots, \mathbf{v}_k^{(0)}\right\}$ *to an initial codebook.*

Step 1 (Partition) *Construct* $\mathcal{V}^{(j)} = \left\{V_1^{(j)}, \ldots, V_k^{(j)}\right\}$ *by setting*
$V_\ell^{(j)} = \left\{\mathbf{x} : \Delta\left(\mathbf{x}, \mathbf{v}_\ell^{(j)}\right) \leq \Delta\left(\mathbf{x}, \mathbf{v}_m^{(j)}\right),\ m = 1, \ldots, k\right\}$ *for* $\ell = 1, \ldots, k.$

Step 2 (Expectation) *Construct* $\mathcal{C}^{(j+1)} = \left\{\mathbf{v}_1^{(j+1)}, \ldots, \mathbf{v}_k^{(j+1)}\right\}$ *by setting*
$\mathbf{v}_\ell^{(j+1)} = \arg\min_{\mathbf{v}} E\left[\Delta(\mathbf{X}, \mathbf{v}) \middle| \mathbf{X} \in V_\ell^{(j)}\right] = E\left[\mathbf{X} \middle| \mathbf{X} \in V_\ell^{(j)}\right]$ *for* $\ell = 1, \ldots, k.$

Step 3 *Stop if* $\left(1 - \frac{\Delta(q^{(j+1)})}{\Delta(q^{(j)})}\right)$ *is less than or equal to a certain threshold. Otherwise, let* $j = j + 1$ *and go to Step 1.*

Step 1 is complemented with a suitable rule to break ties. When a cell becomes empty in Step 1, one can split the cell with the highest probability, or the cell with the highest partial distortion into two, and delete the empty cell.

It is easy to see that $\Delta(q^{(j)})$ is non-increasing and non-negative, so it must have a limit $\Delta(q^{(\infty)})$. [LBG80] showed that if a limiting quantizer $\mathcal{C}^{(\infty)}$ exists in the sense that $\mathcal{C}^{(j)} \to \mathcal{C}^{(\infty)}$ as $j \to \infty$ (in the usual Euclidean sense), then the codepoints of $\mathcal{C}^{(\infty)}$ are the centroids of the Voronoi regions induced by $\mathcal{C}^{(\infty)}$, so $\mathcal{C}^{(\infty)}$ is a fixed point of the algorithm with zero threshold.

The GL algorithm can easily be adjusted to the case when the distribution of $\mathbf{X}$ is unknown but a set of independent observations $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ of the underlying distribution is known instead. The modifications are straightforward replacements of the expectations by sample averages. In Step 3, the empirical distortion

$$\Delta_n(q) = \frac{1}{n}\sum_{i=1}^{n} \Delta(\mathbf{x}_i, q(\mathbf{x}_i)) = \frac{1}{n}\sum_{\ell=1}^{k}\sum_{\mathbf{x}\in V_\ell} \|\mathbf{v}_\ell - \mathbf{x}\|^2$$

is evaluated in place of the unknown expected distortion $\Delta_n(q)$. The GL algorithm for constructing a vector quantizer based on the data set $\mathcal{X}_n$ is the following.

Algorithm 2 (The GL algorithm for data sets)

Step 0 *Set $j = 0$, and set $\mathcal{C}^{(0)} = \left\{\mathbf{v}_1^{(0)}, \dots, \mathbf{v}_k^{(0)}\right\}$ to an initial codebook.*

Step 1 (Partition) *Construct $\mathcal{V}^{(j)} = \left\{V_1^{(j)}, \dots, V_k^{(j)}\right\}$ by setting*

$$V_\ell^{(j)} = \left\{\mathbf{x} : \Delta\left(\mathbf{x}, \mathbf{v}_\ell^{(j)}\right) \le \Delta\left(\mathbf{x}, \mathbf{v}_m^{(j)}\right),\ m = 1, \dots, k\right\} \textit{ for } \ell = 1, \dots, k.$$

Step 2 (Expectation) *Construct $\mathcal{C}^{(j+1)} = \left\{\mathbf{v}_1^{(j+1)}, \dots, \mathbf{v}_k^{(j+1)}\right\}$ by setting*

$$\mathbf{v}_\ell^{(j+1)} = \arg\min_{\mathbf{v}} \sum_{\mathbf{x} \in V_\ell^{(j)} \cap \mathcal{X}_n} \Delta(\mathbf{x}, \mathbf{v}) = \frac{1}{\left|V_\ell^{(j)}\right|} \sum_{\mathbf{x} \in V_\ell^{(j)} \cap \mathcal{X}_n} \mathbf{x} \textit{ for } \ell = 1, \dots, k.$$

Step 3 *Stop if $\left(1 - \frac{\Delta_n(q^{(j+1)})}{\Delta_n(q^{(j)})}\right)$ is less than a certain threshold. Otherwise, let $j = j + 1$ and go to Step 1.*

For a finite training set, the GL algorithm always converges in a finite number of iterations since the average distortion is non-increasing in both Step 1 and Step 2 and there is only a finite number of ways to partition the training set into k subsets.

2.2 Principal Component Analysis

Principal component analysis (PCA), which is also known as the *Karhunen-Loève transformation*, is perhaps the oldest and best-known technique in multivariate analysis. It was first introduced by Pearson [Pea01], who used it in a biological context. It was then developed by Hotelling [Hot33] in work done on psychometry. It appeared once again quite independently in the setting of probability theory, as considered by Karhunen [Kar47], and was subsequently generalized by Loève. For a full treatment of principal component analysis, see, e.g., [JW92].

Principal component analysis can be considered one of the simplest forms of unsupervised learning when the manifold to fit is a linear subspace. Principal components are also used for initialization in more sophisticated unsupervised learning methods.

The analysis is motivated by the following two problems.

1. Given a random vector $\mathbf{X} \in \mathbb{R}^d$, find the d'-dimensional linear subspace that captures most of the variance of $\mathbf{X}$. This is the problem of *feature extraction* where the objective is to reduce the dimension of the data while retaining most of its information content.

2. Given a random vector $\mathbf{X} \in \mathbb{R}^d$, find the d'-dimensional linear subspace that minimizes the expected distance of $\mathbf{X}$ from the subspace. This problem arises in the area of *data compression* where the task is to represent the data with only a few parameters while keeping low the distortion generated by the projection.

It turns out that the two problems have the same solution, and the solution lies in the eigenstructure of the covariance matrix of $\mathbf{X}$. Before we derive this result in Section 2.2.2, we introduce the definition and show some properties of curves in the d-dimensional Euclidean space in Section 2.2.1. Concepts defined here will be used throughout the thesis. After the analysis, in Section 2.2.3, we summarize some of the properties of the first principal component line. In subsequent definitions of principal curves, these properties will serve as bases for generalization. Finally, in Section 2.2.4 we describe a fast algorithm to find principal components of data sets. The significance of this algorithm is that it is similar in spirit to both the GL algorithm of vector quantization and the HS algorithm (Section 3.1.1) for computing principal curves of data sets.

2.2.1 One-Dimensional Curves

In this section we define curves, lines, and line segments in the d-dimensional Euclidean space. We also introduce the notion of the distance function, the expected Euclidean squared distance of a random vector and a curve. The distance function will be used throughout this thesis as a measure of the distortion when a random vector is represented by its projection to a curve. This section also contains some basic facts on curves that are needed later for the definition and analysis of principal curves (see, e.g., [O'N66] for further reference).

Definition 1 *A* curve *in d-dimensional Euclidean space is a continuous function $\mathbf{f}: I \to \mathbb{R}^d$, where $I = [a, b]$ is a closed interval of the real line.*

The curve $\mathbf{f}$ can be considered as a vector of d functions of a single variable t, $\mathbf{f}(t) = (f_1(t), \dots, f_d(t))$, where $f_1(t), \dots, f_d(t)$ are called the *coordinate functions*.

The Length of a Curve

The *length* of a curve $\mathbf{f}$ over an interval $[\alpha, \beta] \subset [a, b]$, denoted by $l(\mathbf{f}, \alpha, \beta)$, is defined by

$$l(\mathbf{f}, \alpha, \beta) = \sup \sum_{i=1}^{N} \|\mathbf{f}(t_i) - \mathbf{f}(t_{i-1})\|, \qquad (13)$$

where the supremum is taken over all finite partitions of $[\alpha, \beta]$ with arbitrary subdivision points $\alpha = t_0 \leq t_1 < \cdots \leq t_N = \beta$, $N \geq 1$. The length of $\mathbf{f}$ over its entire domain $[a, b]$ is denoted by $l(\mathbf{f})$.

Distance Between a Point and a Curve

Let $\mathbf{f}(t) = (f_1(t), \dots, f_d(t))$ be a curve in $\mathbb{R}^d$ parameterized by $t \in \mathbb{R}$, and for any $\mathbf{x} \in \mathbb{R}^d$ let $t_\mathbf{f}(\mathbf{x})$ denote the parameter value t for which the distance between $\mathbf{x}$ and $\mathbf{f}(t)$ is minimized (see Figure 2). More formally, the *projection index* $t_\mathbf{f}(\mathbf{x})$ is defined by

$$t_\mathbf{f}(\mathbf{x}) = \sup\left\{t : \|\mathbf{x} - \mathbf{f}(t)\| = \inf_{\tau} \|\mathbf{x} - \mathbf{f}(\tau)\|\right\}, \tag{14}$$

where $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^d$. Accordingly, the *projection point* of $\mathbf{x}$ to $\mathbf{f}$ is $\mathbf{f}(t_\mathbf{f}(\mathbf{x}))$. The squared Euclidean distance of $\mathbf{f}$ and $\mathbf{x}$ is the squared distance of $\mathbf{x}$ from its projection point to $\mathbf{f}$, that is,

$$\Delta(\mathbf{x}, \mathbf{f}) = \inf_{a \le t \le b} \|\mathbf{x} - \mathbf{f}(t)\|^2 = \|\mathbf{x} - \mathbf{f}(t_\mathbf{f}(\mathbf{x}))\|^2. \tag{15}$$

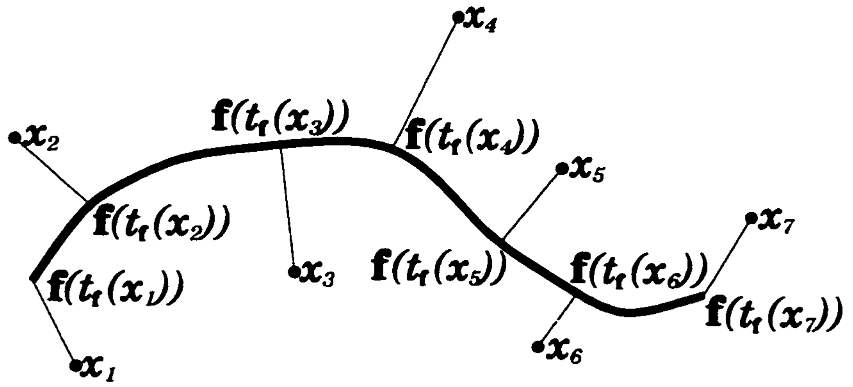


Figure 2: Projecting points to a curve.

Arc Length Parameterization and the Lipschitz Condition

Two curves $\mathbf{f} : [a,b] \to \mathbb{R}^d$ and $\mathbf{g} : [a', b'] \to \mathbb{R}^d$ are said to be *equivalent* if there exist two nondecreasing continuous functions $\phi : [0,1] \to [a,b]$ and $\eta : [0,1] \to [a', b']$ such that

$$\mathbf{f}(\phi(t)) = \mathbf{g}(\eta(t)), \quad 0 \le t \le 1.$$

In this case we write $\mathbf{f} \sim \mathbf{g}$, and it is easy to see that $\sim$ is an equivalence relation. If $\mathbf{f} \sim \mathbf{g}$, then $l(\mathbf{f}) = l(\mathbf{g})$. A curve $\mathbf{f}$ over $[a,b]$ is said to be *parameterized by its arc length* if $l(\mathbf{f}, a, t) = t - a$ for any $a \le t \le b$. Let $\mathbf{f}$ be a curve over $[a,b]$ with length L. It is not hard to see that there exists a unique arc length parameterized curve $\mathbf{g}$ over $[0, L]$ such that $\mathbf{f} \sim \mathbf{g}$.

Let $\mathbf{f}'$ be any curve with length $L' \le L$, and consider the arc length parameterized curve $\mathbf{g} \sim \mathbf{f}'$ with parameter interval $[0, L']$. By definition (13), for all $s_1, s_2 \in [0, L']$ we have $\|\mathbf{g}(s_1) - \mathbf{g}(s_2)\| \le$

$|s_1 - s_2|$. Define $\hat{\mathbf{g}}(t) = \mathbf{g}(L't)$ for $0 \le t \le 1$. Then $\mathbf{f} \sim \hat{\mathbf{g}}$, and $\hat{\mathbf{g}}$ satisfies the Lipschitz condition, i.e., For all $t_1, t_2 \in [0,1]$,

$$\|\hat{\mathbf{g}}(t_1) - \hat{\mathbf{g}}(t_2)\| = \|\mathbf{g}(L't_1) - \mathbf{g}(L't_2)\| \le L'|t_1 - t_2| \le L|t_1 - t_2|. \tag{16}$$

On the other hand, note that if $\hat{\mathbf{g}}$ is a curve over $[0,1]$ which satisfies the Lipschitz condition (16), then its length is at most L.

Note that if $l(\mathbf{f}) < \infty$, then by the continuity of $\mathbf{f}$, its graph

$$G_{\mathbf{f}} = \mathbf{f}([a,b]) = \{\mathbf{f}(t) : a \le t \le b\} \tag{17}$$

is a compact subset of $\mathbb{R}^d$, and the infimum in (15) is achieved for some t. Also, since $G_{\mathbf{f}} = G_{\mathbf{g}}$ if $\mathbf{f} \sim \mathbf{g}$, we also have that $\Delta(\mathbf{x}, \mathbf{f}) = \Delta(\mathbf{x}, \mathbf{g})$ for all $\mathbf{g} \sim \mathbf{f}$.

Geometrical Properties of Curves

Let $\mathbf{f} : [a,b] \to \mathbb{R}^d$ be a differentiable curve with $\mathbf{f} = (f_1, \ldots, f_d)$. The *velocity* of the curve is defined as the vector function

$$\mathbf{f}'(t) = \left(\frac{df_1}{dt}(t), \ldots, \frac{df_d}{dt}(t)\right).$$

It is easy to see that $\mathbf{f}'(t)$ is tangent to the curve at t and that for an arc length parameterized curve $\|\mathbf{f}'(t)\| \equiv 1$. Note that for a differentiable curve $\mathbf{f} : [a,b] \to \mathbb{R}^d$, the length of the curve (13) over an interval $[\alpha, \beta] \subset [a,b]$ can be defined as

$$l(\mathbf{f}, \alpha, \beta) = \int_\alpha^\beta \|\mathbf{f}'(t)\| dt.$$

The vector function

$$\mathbf{f}''(t) = \left(\frac{d^2 f_1}{dt^2}(t), \ldots, \frac{d^2 f_d}{dt^2}(t)\right)$$

is called the *acceleration* of the curve at t. For an arc length parameterized curve $\mathbf{f}''(t)$ is orthogonal to the tangent vector. In this case $\mathbf{f}''(t)/\|\mathbf{f}''(t)\|$ is called the *principal normal* to the curve at t. The vectors $\mathbf{f}'(t)$ and $\mathbf{f}''(t)$ span a plane. There is a unique arc length parameterized circle in this plane that goes through $\mathbf{f}(t)$ and has the same velocity and acceleration at t as the curve itself. The radius $r_{\mathbf{f}}(t) = 1/\|\mathbf{f}''(t)\|$ is called the *radius of curvature* of the curve $\mathbf{f}$ at t. The center $\mathbf{c}_{\mathbf{f}}(t)$ of the circle is called the *center of curvature* of $\mathbf{f}$ at t (Figure 3).

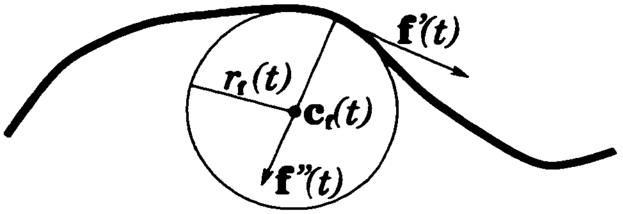


Figure 3: The velocity $\mathbf{f}'(t)$, the acceleration $\mathbf{f}''(t)$, the radius of curvature $r_\mathbf{f}(t)$, and the center of curvature $\mathbf{c}_\mathbf{f}(t)$ of an arc length parameterized curve.

The Distance Function and the Empirical Distance Function

Consider a d-dimensional random vector $\mathbf{X} = (X_1, \ldots, X_d)$ with finite second moments. The *distance function* of a curve $\mathbf{f}$ is defined as the expected squared distance between $\mathbf{X}$ and $\mathbf{f}$, that is,

$$\Delta(\mathbf{f}) = E\left[\Delta(\mathbf{X},\mathbf{f})\right] = E\left[\inf_t \|\mathbf{X} - \mathbf{f}(t)\|^2\right] = E\left[\|\mathbf{X} - \mathbf{f}(t_\mathbf{f}(\mathbf{X}))\|^2\right]. \tag{18}$$

In practical situations the distribution of $\mathbf{X}$ is usually unknown, but a data set $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ drawn independently from the distribution is known instead. In this case, we can estimate the distance function of a curve $\mathbf{f}$ by the *empirical distance function* defined as

$$\Delta_n(\mathbf{f}) = \frac{1}{n}\sum_{i=1}^{n}\Delta(\mathbf{x}_i, \mathbf{f}). \tag{19}$$

Straight Lines and Line Segments

Consider curves of the form

$$\mathbf{s}(t) = t\mathbf{u} + \mathbf{c}$$

where $\mathbf{u}, \mathbf{c} \in \mathbb{R}^d$, and $\mathbf{u}$ is a unit-vector. If the domain of t is the real line, $\mathbf{s}$ is called a *straight line*, or *line*. If $\mathbf{s}$ is defined on a finite interval $[a,b] \subset \mathbb{R}$, $\mathbf{s}$ is called a *straight line segment*, or *line segment*. Note that since $\|\mathbf{u}\| = 1$, $\mathbf{s}$ is arc length parameterized.

By (15), the squared distance of a point $\mathbf{x}$ and a line $\mathbf{s}$ is

$$\begin{aligned}
\Delta(\mathbf{x},\mathbf{s}) &= \inf_{t\in\mathbb{R}} \|\mathbf{x} - \mathbf{s}(t)\|^2 \\
&= \inf_{t\in\mathbb{R}} \|\mathbf{x} - (t\mathbf{u} + \mathbf{c})\|^2 \\
&= \|\mathbf{x} - \mathbf{c}\|^2 + \inf_{t\in\mathbb{R}} \left(t^2 - 2t(\mathbf{x} - \mathbf{c})^T\mathbf{u}\right) \\
&= \|\mathbf{x} - \mathbf{c}\|^2 - ((\mathbf{x} - \mathbf{c})^T\mathbf{u})^2
\end{aligned} \tag{20}$$

where $\mathbf{x}^T$ denotes the transpose of $\mathbf{x}$. The projection point of $\mathbf{x}$ to $\mathbf{s}$ is $\mathbf{c} + ((\mathbf{x} - \mathbf{c})^T\mathbf{u})\mathbf{u}$.

If $\mathbf{s}(t) = t\mathbf{u} + \mathbf{c}$ is a line segment defined over $[a,b] \subset \mathbb{R}$, the way the distance of a point $\mathbf{x}$ and the line segment is measured depends on the value of the projection index $t_\mathbf{s}(\mathbf{x})$. If $t_\mathbf{s}(\mathbf{x}) = a$ or

$t_{\mathbf{s}}(\mathbf{x}) = b$, the distance is measured as the distance of $\mathbf{x}$ and one of the endpoints $\mathbf{v}_1 = a\mathbf{u} + \mathbf{c}$ or $\mathbf{v}_2 = b\mathbf{u} + \mathbf{c}$, respectively. If $\mathbf{x}$ projects to $\mathbf{s}$ between the endpoints, the distance is measured as if $\mathbf{s}$ were a line (Figure 4). Formally,

$$\Delta(\mathbf{x}, \mathbf{s}) = \begin{cases} \|\mathbf{x} - \mathbf{v}_1\|^2 & \text{if } \mathbf{s}(t_{\mathbf{s}}(\mathbf{x})) = \mathbf{v}_1, \\ \|\mathbf{x} - \mathbf{v}_2\|^2 & \text{if } \mathbf{s}(t_{\mathbf{s}}(\mathbf{x})) = \mathbf{v}_2, \\ \|\mathbf{x} - \mathbf{c}\|^2 - ((\mathbf{x} - \mathbf{c})^T \mathbf{u})^2 & \text{otherwise.} \end{cases} \tag{21}$$

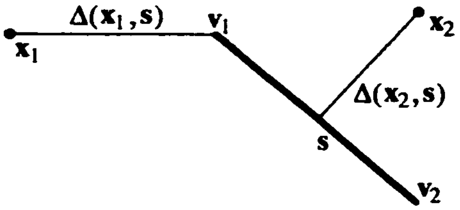


Figure 4: Distance of a point and a line segment. If a point $\mathbf{x}_1$ projects to one of the endpoints $\mathbf{v}_1$ of the line segment $\mathbf{s}$, the distance of $\mathbf{x}_1$ and $\mathbf{s}$ is identical to the distance of $\mathbf{x}_1$ and $\mathbf{v}_1$. If a point $\mathbf{x}_2$ projects to $\mathbf{s}$ between the endpoints, the distance is measured as if $\mathbf{s}$ were a line.

2.2.2 Principal Component Analysis

Consider a d-dimensional random vector $\mathbf{X} = (X_1, \ldots, X_d)$ with finite second moments and zero mean[1]. Let $\mathbf{u} \in \mathbb{R}^d$ be an arbitrary unit vector, and $\mathbf{s}(t) = t\mathbf{u}$ the corresponding straight line. Let $Y = t_{\mathbf{s}}(\mathbf{X}) = \mathbf{X}^T\mathbf{u}$ be the projection index of $\mathbf{X}$ to $\mathbf{s}$. From $E[\mathbf{X}] = 0$ it follows that $E[Y] = 0$, and so the variance of Y can be written as

$$\begin{aligned} \sigma_Y^2 &= E[(\mathbf{X}^T\mathbf{u})^2] = E[(\mathbf{u}^T\mathbf{X})(\mathbf{X}^T\mathbf{u})] \\ &= \mathbf{u}^T E[\mathbf{X}\mathbf{X}^T]\mathbf{u} = \mathbf{u}^T\mathbf{R}\mathbf{u} \\ &= \psi(\mathbf{u}) \end{aligned} \tag{22}$$

where the $d \times d$ matrix $\mathbf{R} = E\left[(\mathbf{X} - E[\mathbf{X}])(\mathbf{X} - E[\mathbf{X}])^T\right] = E\left[\mathbf{X}\mathbf{X}^T\right]$ is the *covariance matrix* of $\mathbf{X}$. Since $\mathbf{R}$ is symmetric, $\mathbf{R} = \mathbf{R}^T$, and so for any $\mathbf{v}, \mathbf{w} \in \mathbb{R}^d$

$$\mathbf{v}^T\mathbf{R}\mathbf{w} = \mathbf{w}^T\mathbf{R}\mathbf{v}. \tag{23}$$

To find stationary values of the projection variance $\psi(\mathbf{u})$, consider a small perturbation $\delta\mathbf{u}$, such that $\|\mathbf{u} + \delta\mathbf{u}\| = 1$. From (22) and (23) it follows that

$$\begin{aligned} \psi(\mathbf{u} + \delta\mathbf{u}) &= (\mathbf{u} + \delta\mathbf{u})^T\mathbf{R}(\mathbf{u} + \delta\mathbf{u}) \\ &= \mathbf{u}^T\mathbf{R}\mathbf{u} + 2(\delta\mathbf{u})^T\mathbf{R}\mathbf{u} + (\delta\mathbf{u})^T\mathbf{R}\,\delta\mathbf{u}. \end{aligned}$$

[1] If $E[\mathbf{X}] \neq 0$, then we subtract the mean from $\mathbf{X}$ before proceeding with the analysis.

Ignoring the second order term $(\delta\mathbf{u})^T\mathbf{R}\delta\mathbf{u}$ and using the definition of $\psi(\mathbf{u})$ again, we have

$$\begin{aligned}\psi(\mathbf{u}+\delta\mathbf{u}) &= \mathbf{u}^T\mathbf{R}\mathbf{u}+2(\delta\mathbf{u})^T\mathbf{R}\mathbf{u}\\ &= \psi(\mathbf{u})+2(\delta\mathbf{u})^T\mathbf{R}\mathbf{u}.\end{aligned} \tag{24}$$

If $\mathbf{u}$ is such that $\psi(\mathbf{u})$ has a stationary value, to a first order in $\delta\mathbf{u}$, we have

$$\psi(\mathbf{u}+\delta\mathbf{u}) = \psi(\mathbf{u}). \tag{25}$$

Hence, (25) and (24) imply that

$$(\delta\mathbf{u})^T\mathbf{R}\mathbf{u} = 0. \tag{26}$$

Since $||\mathbf{u}+\delta\mathbf{u}||^2 = ||\mathbf{u}||^2+2(\delta\mathbf{u})^T\mathbf{u}+||\delta\mathbf{u}||^2 = 1$, we require that, to a first order in $\delta\mathbf{u}$,

$$(\delta\mathbf{u})^T\mathbf{u} = 0. \tag{27}$$

This means that the perturbation $\delta\mathbf{u}$ must be orthogonal to $\mathbf{u}$. To find a solution of (26) with the constraint (27), we have to solve

$$(\delta\mathbf{u})^T\mathbf{R}\mathbf{u} - \lambda(\delta\mathbf{u})^T\mathbf{u} = 0,$$

or, equivalently,

$$(\delta\mathbf{u})^T(\mathbf{R}\mathbf{u} - \lambda\mathbf{u}) = 0. \tag{28}$$

For the condition (28) to hold, it is necessary and sufficient that we have

$$\mathbf{R}\mathbf{u} = \lambda\mathbf{u}. \tag{29}$$

The solutions of (29), $\lambda_1,\ldots,\lambda_d$, are the *eigenvalues* of R, and the corresponding unit vectors, $\mathbf{u}_1,\ldots,\mathbf{u}_d$, are the *eigenvectors* of R. For the sake of simplicity, we assume that the eigenvalues are distinct, and they are indexed in decreasing order, i.e.,

$$\lambda_1 > \ldots > \lambda_d.$$

Define the $d \times d$ matrix $\mathbf{U}$ as

$$\mathbf{U} = [\mathbf{u}_1,\ldots,\mathbf{u}_d],$$

and let Λ be the diagonal matrix

$$\Lambda = \mathrm{diag}[\lambda_1,\ldots,\lambda_d].$$

Then the d equations of form (29) can be summarized in

$$\mathbf{RU} = \mathbf{U}\Lambda. \tag{30}$$

The matrix $\mathbf{U}$ is orthonormal so $\mathbf{U}^{-1} = \mathbf{U}^T$, and therefore (30) can be written as

$$\mathbf{U}^T\mathbf{RU} = \Lambda. \tag{31}$$

Thus, from (22) and (31) it follows that the principal directions along which the projection variance is stationary are the eigenvectors of the covariance matrix $\mathbf{R}$, and the stationary values themselves are the eigenvalues of $\mathbf{R}$. (31) also implies that the maximum value of the projection variance is the largest eigenvalue of $\mathbf{R}$, and the principal direction along which the projection variance is maximal is the eigenvector associated with the largest eigenvalue. Formally,

$$\max_{\|\mathbf{u}\|=1} \psi(\mathbf{u}) = \lambda_1, \tag{32}$$

and

$$\arg\max_{\|\mathbf{u}\|=1} \psi(\mathbf{u}) = \mathbf{u}_1. \tag{33}$$

The straight lines $\mathbf{s}_i(t) = t\mathbf{u}_i, i = 1, \ldots, d$ are called the *principal component lines* of $\mathbf{X}$. Since the eigenvectors form an orthonormal basis of $\mathbb{R}^d$, any data vector $\mathbf{x} \in \mathbb{R}^d$ can be represented uniquely by its projection indices $t_i = \mathbf{u}_i^T\mathbf{x}, i = 1, \ldots, d$ to the principal component lines. The projection indices $t_i, \ldots, t_d$ are called the *principal components* of $\mathbf{x}$. The construction of the vector $\mathbf{t} = [t_i, \ldots, t_d]^T$ of the principal components,

$$\mathbf{t} = \mathbf{U}^T\mathbf{x},$$

is the principal component analysis of $\mathbf{x}$. To reconstruct the original data vector $\mathbf{x}$ from $\mathbf{t}$, note again that $\mathbf{U}^{-1} = \mathbf{U}^T$ so

$$\mathbf{x} = (\mathbf{U}^T)^{-1}\mathbf{t} = \mathbf{Ut} = \sum_{i=1}^{d} t_i\mathbf{u}_i. \tag{34}$$

From the perspective of feature extraction and data compression, the practical value of principal component analysis is that it provides an effective technique for dimensionality reduction. In particular, we may reduce the number of parameters needed for effective data representation by discarding those linear combinations in (34) that have small variances and retain only those terms that have large variances. Formally, let $\mathcal{S}_{d'}$ be the d'-dimensional linear subspace spanned by the first d' eigenvectors of $\mathbf{R}$. To approximate $\mathbf{X}$, we define

$$\mathbf{X}' = \sum_{i=1}^{d'} t_j\mathbf{u}_j,$$

the projection of $\mathbf{X}$ to $\mathcal{S}_{d'}$. It can be shown by using (33) and induction that $\mathcal{S}_{d'}$ maximizes the variance of $\mathbf{X}'$,

$$E\left[\mathbf{X}'^2\right] = \sum_{i=1}^{d'} \psi(\mathbf{u}_j) = \sum_{i=1}^{d'} \lambda_j,$$

and minimizes the variance of $\mathbf{X} - \mathbf{X}'$,

$$E\left[(\mathbf{X} - \mathbf{X}')^2\right] = \sum_{i=d'+1}^{d} \psi(\mathbf{u}_j) = \sum_{i=d'+1}^{d} \lambda_j,$$

among all d'-dimensional linear subspaces. In other words, the solutions of both Problem 1 and Problem 2 are the subspace which is spanned by the first d' eigenvectors of $\mathbf{X}$'s covariance matrix.

2.2.3 Properties of the First Principal Component Line

The *first principal component line* (Figure 5) of a random variable $\mathbf{X}$ with zero mean is defined as the straight line $\mathbf{s}_1 = t\mathbf{u}_1$ where $\mathbf{u}_1$ is the eigenvector which belongs to the largest eigenvalue λ_1 of $\mathbf{X}$'s correlation matrix. The first principal component line has the following properties.

1. The first principal component line maximizes the variance of the projection of $\mathbf{X}$ to a line among all straight lines.

2. The first principal component line minimizes the distance function among all straight lines.

3. If the distribution of $\mathbf{X}$ is elliptical, the first principal component line is *self-consistent*, that is, any point of the line is the conditional expectation of $\mathbf{X}$ over those points of the space which project to this point. Formally,

$$\mathbf{s}_1(t) = E\left[\mathbf{X} \mid t_f(\mathbf{X}) = t\right].$$

Property 1 is a straightforward consequence of (33). To show Property 2, note that if $\mathbf{s}(t) = t\mathbf{u} + \mathbf{c}$ is an arbitrary straight line, then by (18) and (20),

$$\begin{aligned}
\Delta(\mathbf{s}) &= E\left[\Delta(\mathbf{X},\mathbf{s})\right] \\
&= E\left[\|\mathbf{X} - \mathbf{c}\|^2 - ((\mathbf{X} - \mathbf{c})^T\mathbf{u})^2\right] \\
&= E\left[\|\mathbf{X}\|^2\right] + \|\mathbf{c}\|^2 - E\left[(\mathbf{X}^T\mathbf{u})^2\right] - (\mathbf{c}^T\mathbf{u})^2 \qquad (35) \\
&= \sigma_{\mathbf{X}}^2 - \psi(\mathbf{u}) + \|\mathbf{c}\|^2 - (\mathbf{c}^T\mathbf{u})^2 \\
&\le \sigma_{\mathbf{X}}^2 - \psi(\mathbf{u}), \qquad (36)
\end{aligned}$$

where (35) follows from $E[\mathbf{X}] = 0$. On the one hand, in (36) equality holds if and only if $\mathbf{c} = t\mathbf{u}$ for some $t \in \mathbb{R}$. Geometrically, it means that the minimizing line must go through the origin. On

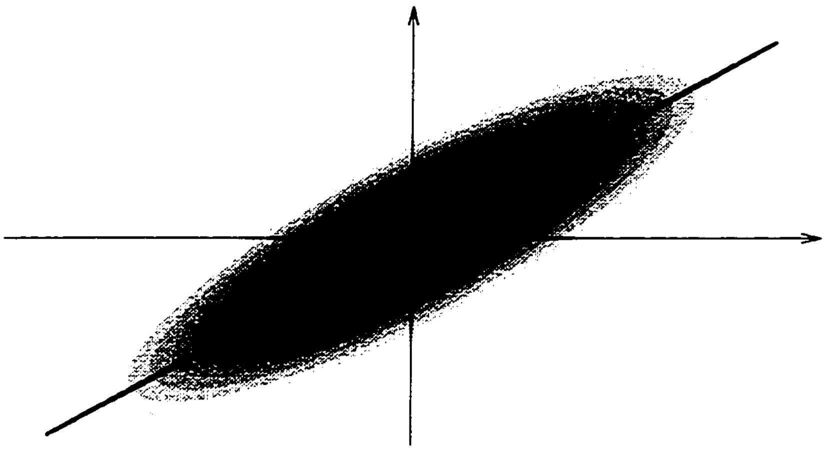


Figure 5: The first principal component line of an elliptical distribution in the plane.

the other hand, $\sigma_{\mathbf{X}}^2 - \psi(\mathbf{u})$ is minimized when $\psi(\mathbf{u})$ is maximized, that is, when $\mathbf{u} = \mathbf{u}_1$. These two conditions together imply Property 2. Property 3 follows from the fact that the density of a random variable with an elliptical distribution is symmetrical about the principal component lines.

2.2.4 A Fast PCA Algorithm for Data Sets

In practice, principal component analysis is usually applied for sets of data points rather than distributions. Consider a data set $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$, such that $\frac{1}{n}\sum_{i=1}^n \mathbf{x}_n = 0$. The first principal component line of $\mathcal{X}_n$ is a straight line $\mathbf{s}_1(t) = t\mathbf{u}_1$ that minimizes the empirical distance function (19),

$$\Delta_n(\mathbf{s}) = \frac{1}{n}\sum_{i=1}^{n}\Delta(\mathbf{x}_i, \mathbf{s}),$$

among all straight lines. The solution lies in the eigenstructure of the sample covariance matrix of the data set, which is defined as $\mathbf{R}_n = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_n\mathbf{x}_n^T$. Following the derivation of PCA for distributions previously in this section, it can be shown easily that the unit vector $\mathbf{u}_1$ that defines the minimizing line $\mathbf{s}_1$ is the eigenvector which belongs to the largest eigenvalue of $\mathbf{R}_n$.

An obvious algorithm to minimize $\Delta_n(\mathbf{s})$ is therefore to find the eigenvectors and eigenvalues of $\mathbf{R}_n$. The crude method, direct diagonalization of $\mathbf{R}_n$, can be extremely costly for high-dimensional data since it takes $O(nd^3)$ operations. More sophisticated techniques, for example the power method (e.g., see [Wil65]), exist that perform matrix diagonalization in $O(nd^2)$ steps if only the first leading eigenvectors and eigenvalues are required. Since the $d \times d$ covariance matrix $\mathbf{R}_n$ must explicitly be computed, $O(nd^2)$ is also the theoretical lower limit of the computational complexity of this approach.

To break the $O(nd^2)$ barrier, several approximative methods were proposed (e.g., [Oja92],

[RT89], [Föl89]). The common approach of these methods is to start from an arbitrary line, and to iteratively optimize the orientation of the line using the data so that it converges to the first principal component line. The characterizing features of these algorithms are the different learning rules they use for the optimization in each iteration.

The algorithm we introduce here is of the same genre. It was proposed recently, independently by Roweis [Row98] and Tipping and Bishop [TB99]. The reason we present it here is that there is a strong analogy between this algorithm designed for finding the first principal component line[2], and the HS algorithm (Section 3.1.1) for computing principal curves of data sets. Moreover, we also use a similar method in the inner iteration of the polygonal line algorithm (Section 5.1) to optimize the locations of vertices of the polygonal principal curve.

The basic idea of the algorithm is the following. Start with an arbitrary straight line, and project all the data points to the line. Then *fix the projection indices*, and find a new line that optimizes the distance function. Once the new line has been computed, restart the iteration, and continue until convergence.

Formally, let $\mathbf{s}^{(j)}(t) = t\mathbf{u}^{(j)}$ be the line produced by the jth iteration, and let $\mathbf{t}^{(j)} = \left[t_1^{(j)}, \ldots, t_n^{(j)}\right]^T = \left[\mathbf{x}_1^T\mathbf{u}^{(j)}, \ldots, \mathbf{x}_n^T\mathbf{u}^{(j)}\right]^T$ be the vector of projection indices of the data points to $\mathbf{s}^{(j)}$. The distance function of $\mathbf{s}(t) = t\mathbf{u}$ assuming the fixed projection vector $\mathbf{t}^{(j)}$ is defined as

$$\begin{aligned}\Delta_n\left(\mathbf{s}\,\middle|\,\mathbf{t}^{(j)}\right) = &= \sum_{i=1}^{n}\left\|\mathbf{x}_i - t_i^{(j)}\mathbf{u}\right\|^2 \\ &= \sum_{i=1}^{n}\|\mathbf{x}_i\|^2 + \|\mathbf{u}\|^2\sum_{i=1}^{n}\left(t_i^{(j)}\right)^2 - 2\mathbf{u}^T\sum_{i=1}^{n}t_i^{(j)}\mathbf{x}_i. \end{aligned} \tag{37}$$

Therefore, to find the optimal line $\mathbf{s}^{(j+1)}$, we have to minimize (37) with the constraint that $\|\mathbf{u}\| = 1$. It can be shown easily that the result of the constrained minimization is

$$\mathbf{u}^{(j+1)} = \arg\min_{\|\mathbf{u}\|=1}\Delta\left(\mathbf{s}\,\middle|\,\mathbf{t}^{(j)}\right) = \frac{\sum_{i=1}^{n}t_i^{(j)}\mathbf{x}_i}{\left\|\sum_{i=1}^{n}t_i^{(j)}\mathbf{x}_i\right\|},$$

and so $\mathbf{s}^{(j+1)}(t) = t\mathbf{u}^{(j+1)}$.

The formal algorithm is the following.

Algorithm 3 (The RTB algorithm)

Step 0 *Let* $\mathbf{s}^{(0)}(t) = t\mathbf{u}^{(0)}$ *be an arbitrary line. Set* $j = 0$.

Step 1 *Set* $\mathbf{t}^{(j)} = \left[t_1^{(j)}, \ldots, t_n^{(j)}\right]^T = \left[\mathbf{x}_1^T\mathbf{u}^{(j)}, \ldots, \mathbf{x}_n^T\mathbf{u}^{(j)}\right]^T$.

[2] The original algorithm in [Row98] and [TB99] can compute the first d' principal components simultaneously. For the sake of simplicity, we present it here only for the first principal component line.

Step 2 *Define* $\mathbf{u}^{(j+1)} = \frac{\sum_{i=1}^{n} t_i^{(j)} \mathbf{x}_i}{\left\|\sum_{i=1}^{n} t_i^{(j)} \mathbf{x}_i\right\|}$, *and* $\mathbf{s}^{(j+1)}(t) = t\mathbf{u}^{(j+1)}$.

Step 3 *Stop if* $\left(1 - \frac{\Delta_n(\mathbf{s}^{(j+1)})}{\Delta_n(\mathbf{s}^{(j)})}\right)$ *is less than a certain threshold. Otherwise, let* $j = j + 1$ *and go to Step 1.*

The standard convergence proof for the Expectation-Minimization (EM) algorithm [DLR77] applies to the RTB algorithm so it can be shown that $\mathbf{s}^{(j)}$ has a limit $\mathbf{s}^{(\infty)}$, and that the distance function $\Delta_n(\mathbf{s})$ has a local maximum in $\mathbf{s}^{(\infty)}$. Furthermore, [TB99] showed that the only stable local extremum is the global maximum so $\mathbf{s}^{(\infty)}$ is indeed the first principal component line.

Chapter 3

Principal Curves and Related Areas

In this chapter we introduce the original HS definition of principal curves and summarize some of the subsequent research. We also describe the connection of principal curves to some of the related unsupervised learning models. Section 3.1 introduces the HS definition of principal curves, and describes the HS algorithm for probability distributions and data sets. Section 3.2 summarizes some of the subsequent results on principal curves and highlights the relationship between principal curves, self-organizing maps and nonlinear principal component analysis.

3.1 Principal Curves with Self-Consistency Property

3.1.1 The HS Definition

Property 3 in Section 2.2 states that for elliptical distributions the first principal component is self-consistent, i.e., any point of the line is the conditional expectation of **X** over those points of the space which project to this point. HS generalized the self-consistency property of principal components and defined principal curves as follows.

Definition 2 *The smooth curve* $\mathbf{f}(t)$ *is a principal curve if the following hold:*

(i) $\mathbf{f}$ *does not intersect itself,*

(ii) $\mathbf{f}$ *has finite length inside any bounded subset of* $\mathbb{R}^d$,

(iii) $\mathbf{f}$ *is self-consistent, i.e.,* $\mathbf{f}(t) = E\left[\mathbf{X}|t_{\mathbf{f}}(\mathbf{X}) = t\right]$.

Intuitively, self-consistency means that each point of **f** is the average (under the distribution of **X**) of all points that project there. Thus, principal curves are smooth self-consistent curves which pass through the "middle" of the distribution and provide a good one-dimensional nonlinear summary of the data (see Figure 6).

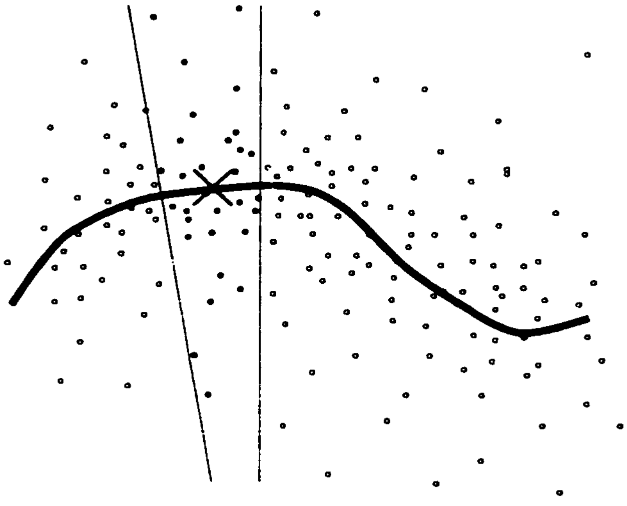


Figure 6: Self-consistency. Each point of the curve is the average of points that project there.

It follows from the discussion in Section 2.2 that the principal component lines are stationary points of the distance function. HS proved an analogous result for principal curves. Formally, let $\mathbf{f}$ be a smooth (infinitely differentiable) curve, and for $\lambda \in \mathbb{R}$ consider the perturbation $\mathbf{f}+\lambda\mathbf{g}$ of $\mathbf{f}$ by a smooth curve $\mathbf{g}$ such that $\sup_t \|\mathbf{g}(t)\| \leq 1$ and $\sup_t \|\mathbf{g}'(t)\| \leq 1$. Then $\mathbf{f}$ is a principal curve if and only if $\mathbf{f}$ is a stationary point of the distance function in the sense that for all such $\mathbf{g}$,

$$\left.\frac{\partial\Delta(\mathbf{f}+\lambda\mathbf{g})}{\partial\lambda}\right|_{\lambda=0} = 0.$$

In this sense the HS principal curve definition is a natural generalization of principal components.

Existence of the HS Principal Curves

The existence of principal curves defined by the self-consistency property is in general an open question. Until recently, the existence of principal curves had been proven only for some special distributions, such as elliptical or spherical distributions, or distributions concentrated on a smooth curve. The first results on principal curves of non-trivial distributions are due to Duchamp and Stuetzle [DS96a] who studied principal curves in the plane. They showed that principal curves are solutions of a differential equation. By solving this differential equation for uniform densities on rectangles and annuli, they found oscillating principal curves besides the obvious straight and circular ones, indicating that principal curves in general will not be unique. They also showed that if a density has several principal curves, they have to cross, a property somewhat analogous to the orthogonality of principal components.

The HS Algorithm for Distributions

Based on the self-consistency property, HS developed an algorithm for constructing principal curves. Similar in spirit to the GL algorithm of vector quantizer design (Section 2.1), and the RTB algorithm (Section 2.2.4) of principal component analysis, the HS algorithm iterates between a projection step and an expectation step until convergence. In the projection step, projection indices of the data to the curve are computed. In the expectation step, a new curve is computed. For every point $\mathbf{f}^{(j)}(t)$ of the previous curve, a point of the new curve is defined as the expectation of the data that project to $\mathbf{f}^{(j)}(t)$. When the probability density of $\mathbf{X}$ is known, the formal algorithm for constructing principal curves is the following.

Algorithm 4 (The HS algorithm)

Step 0 *Let* $\mathbf{f}^{(0)}(t)$ *be the first principal component line for* $\mathbf{X}$. *Set* $j = 0$.

Step 1 (Projection) *Set* $t_{\mathbf{f}^{(j)}}(\mathbf{x}) = \max\{t : \|\mathbf{x} - \mathbf{f}(t)\| = \min_\tau \|\mathbf{x} - \mathbf{f}(\tau)\|\}$ *for all* $\mathbf{x} \in \mathbb{R}^d$.

Step 2 (Expectation) *Define* $\mathbf{f}^{(j+1)}(t) = E\left[\mathbf{X} | t_{\mathbf{f}^{(j)}}(\mathbf{X}) = t\right]$.

Step 3 *Stop if* $\left(1 - \frac{\Delta(\mathbf{f}^{(j+1)})}{\Delta(\mathbf{f}^{(j)})}\right)$ *is less than a certain threshold. Otherwise, let* $j = j + 1$ *and go to Step 1.*

Although HS is unable to prove that the algorithm converges, they have the following evidence in its favor:

1. By definition, principal curves are fixed points of the algorithm.
2. Assuming that each iteration is well defined and produces a differentiable curve, the expected squared distance $\Delta(\mathbf{f})$ converges.
3. If Step 1 is replaced by fitting a least squares straight line, then the procedure converges to the largest principal component.

Unfortunately, the fact that $\Delta(\mathbf{f})$ converges does not mean that $\mathbf{f}$ converges to any meaningful solution. Among the principal components, the largest principal component minimizes the distance function, the smallest principal component maximizes it, and all the others are saddle points. Interestingly, there is no such distinction between different principal curves of a distribution. [DS96b] showed that all principal curves are saddle points of the distance function. In this sense, any algorithm that aims to find a principal curve by minimizing the distance function will fail to converge to a stable solution without further restricting the set of admissible curves. This fact is one of the motivations behind our new definition of principal curves in Chapter 4.

3.1.2 The HS Algorithm for Data Sets

Similarly to the GL and RTB algorithms, the HS algorithm can be extended to data sets. Unlike in the former two cases, however, this case requires more than simple replacements of expectations by the corresponding sample averages. A general issue is the representation of the curve by a finite number of parameters. A more serious problem arises in the expectation step: in general there is at most one point that projects to a given point of the curve. In this section we give a detailed treatment of the modifications proposed by HS, and analyze the algorithm.

Assume that a set of points $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ is given. Project the data points to an arbitrary curve $\mathbf{f}$, and index the points so that their projection indices $t_1, \ldots, t_n$ are in increasing order. We can represent the curve $\mathbf{f}$ by a polygonal curve of n vertices by connecting pairs of consecutive projection points $(\mathbf{f}(t_i), \mathbf{f}(t_{i+1})), i = 1, \ldots, n-1$ by line segments. In the discussion below we assume that all curves produced by the algorithm are such polygonal curves. We also assume that all curves are arc length parameterized, so the parameters $t_i, i = 1, \ldots, n$ can be defined recursively by

$$\begin{aligned} &1. \quad t_1 = 0, \\ &2. \quad t_i = t_{i-1} + \|\mathbf{f}(t_i) - \mathbf{f}(t_{i-1})\|, \quad i = 2, \ldots, n. \end{aligned} \tag{38}$$

In Step 0, $\mathbf{f}^{(0)}(t)$ is the first principal component line of the data set $\mathcal{X}_n$. In the stopping condition in Step 3, the distance function $\Delta\left(\mathbf{f}^{(j+1)}\right)$ is replaced by the empirical distance function,

$$\Delta_n\left(\mathbf{f}^{(j+1)}\right) = \frac{1}{n}\sum_{i=1}^{n}\left\|\mathbf{x}_i - \mathbf{f}^{(j+1)}\left(t_i^{(j)}\right)\right\|^2.$$

In the projection step (Step 1), the new projection indices $t_i^{(j)}, i = 1, \ldots, n$ are computed by projecting the data points to $\mathbf{f}^{(j)}(t)$. When we reach this step for the first time, the projection indices can be set by projecting the data points to the first principal component line. After the jth iteration, the current curve is represented as a polygonal curve of vertices $\mathbf{f}^{(j)}\left(t_1^{(j-1)}\right), \ldots, \mathbf{f}^{(j)}\left(t_n^{(j-1)}\right)$. Let $\mathbf{s}_\ell$ be the line segment that connects the vertices $\mathbf{f}^{(j)}\left(t_\ell^{(j-1)}\right)$ and $\mathbf{f}^{(j)}\left(t_{\ell+1}^{(j-1)}\right)$, $\ell = 1, \ldots, n-1$. To compute the projection index of a data point $\mathbf{x}_i$, we have to find the nearest line segment to $\mathbf{x}_i$, denoted by $\mathbf{s}_{\ell(i)}$, where the index $\ell(i)$ is defined by

$$\ell(i) = \underset{\ell=1,\ldots,n-1}{\arg\min}\ \Delta(\mathbf{x}_i, \mathbf{s}_\ell).$$

If $\mathbf{s}_{\ell(i)}$ is defined over $\left[t_{\ell(i)}^{(j-1)}, t_{\ell(i)+1}^{(j-1)}\right]$, the new projection index is identical to the projection index of $\mathbf{x}_i$ to $\mathbf{s}_{\ell(i)}$, that is,

$$t_i^{(j)} = t_{\mathbf{s}_{\ell(i)}}(\mathbf{x}_i).$$

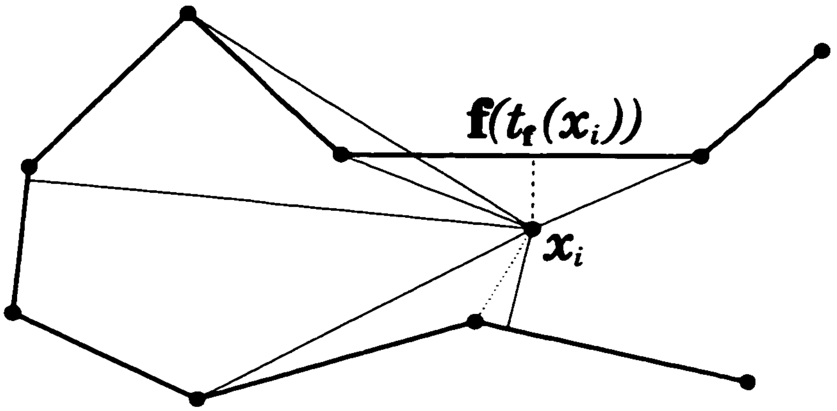


Figure 7: Computing projection points. $\mathbf{x}_i$ is projected to the line segments (along the solid thin lines), and the nearest projection point is chosen (connected to $\mathbf{x}_i$ by dashed line). Note that the nearest vertex (connected to $\mathbf{x}_i$ by dotted line) is not the endpoint of the nearest line segment.

Figure 7 illustrates the method for a data point.

There seems to be a simpler approach to find the projection point of a data point $\mathbf{x}_i$. Instead of searching through the line segments, one could find the nearest vertex to $\mathbf{x}_i$, project the point to the vertex and the two incident line segments, and pick the nearest projection. Figure 7 clearly indicates that this approach can yield a wrong result if the nearest vertex of the polygonal curve is not the endpoint of nearest line segment. Although this configuration occurs quite rarely, in general it cannot be excluded.

Before proceeding with expectation step, the data points are reindexed in increasing order by their projection indices.

In the expectation step (Step 2), finite points of the new curve $\mathbf{f}^{(j+1)}(t) = E\left[\mathbf{X}|t_{\mathbf{f}^{(j)}}(\mathbf{X}) = t\right]$ are estimated at the n projection indices $t = t_1^{(j)}, \ldots, t_n^{(j)}$ found in the projection step. In general, the only observation that projects to $\mathbf{f}^{(j)}(t)$ at $t_i^{(j)}$ is $\mathbf{x}_i$. Using this one point in the averaging would result in a curve that visits all the data points after the first iteration. To tackle this problem, HS proposed two approaches. In the first, $E\left[\mathbf{X}|t_{\mathbf{f}^{(j)}}(\mathbf{X}) = t_i^{(j)}\right]$ is estimated by averaging over observations that *project close* to $t_i^{(j)}$. HS used the *locally weighted running-lines smoother* [Cle79] for local averaging. In the second approach, a non-parametric regression estimate (*cubic smoothing splines* [Sil85]) is used to minimize a data-dependent criteria.

Locally Weighted Running-Lines Smoother

Consider the estimation of the single coordinate function $E\left[X|t_{\mathbf{f}^{(j)}}(X) = t_i^{(j)}\right]$ based on the sample of n pairs $(t_1^{(j)}, x_1), \ldots, (t_n^{(j)}, x_n)$. To estimate this quantity, the smoother fits a straight line to the first wn observations $\{x_k\}$ of which the projection index $t_k^{(j)}$ is the closest to $t_i^{(j)}$. The estimate is taken to be the fitted value of the line at $t_i^{(j)}$. The fraction w of points in the neighborhood is called the *span*, and w is a parameter of the smoother. In fitting the line, weighted least squares regression is used. The weights are derived from a symmetric kernel centered at $t_i^{(j)}$ that goes smoothly to 0 within the neighborhood. Formally, let $t_w^{(j)}$ denote the wnth nearest projection index to $t_i^{(j)}$, and define the weight w_{ik} of the observation x_k by

$$w_{ik} = \begin{cases} \left(1 - \left|\frac{t_k^{(j)} - t_i^{(j)}}{t_w^{(j)} - t_i^{(j)}}\right|^3\right)^{1/3} & \text{if } \left|t_k^{(j)} - t_i^{(j)}\right| < \left|t_w^{(j)} - t_i^{(j)}\right|, \\ 0 & \text{otherwise.} \end{cases} \tag{39}$$

Cubic Smoothing Splines

The algorithm to estimate principal curves for data sets is motivated by the algorithm for finding principal curves of distributions, so it is designed to find a stationary point of the average squared distance, $\Delta_n(\mathbf{f}) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{f}(t_\mathbf{f}(\mathbf{x}_i))\|^2$. To obtain a smooth curve solution, motivated by cubic smoothing splines [Sil85], HS suggested to minimize a penalized average squared distance criterion to define principal curves. Formally, let

$$G(\mathbf{f}) = \Delta_n(\mathbf{f}) + \mu P(\mathbf{f}), \tag{40}$$

where $P(\mathbf{f}) = \int_0^1 \|\mathbf{f}''(\tau)\|^2 d\tau$ measures the total curvature of the curve, and the penalty coefficient μ is a parameter of the algorithm. Note that the parameter of the curve is rescaled to lie in the interval $[0, 1]$. In the expectation step the criteria (40) is minimized by minimizing separately the d coordinate functions,

$$G(f_\ell) = \frac{1}{n}\sum_{i=1}^{n} |x_{i\ell} - f_\ell(t_i)|^2 + \mu \int_0^1 |f_\ell''(\tau)|^2 d\tau, \quad \ell = 1, \ldots, d.$$

Computational Complexity of the HS Algorithm

In the projection step the distance between n line segments and n data points is computed, so the complexity of the step is $O(n^2)$. The computational complexity of the expectation step is $O(n^2)$ for the kernel type smoothers, and $O(n)$ for the smoothing spline. The complexity of the sorting routine after the projection step is $O(n \log n)$. Hence, the computational complexity of the HS algorithm, dominated by the complexity of the projection step, is $O(n^2)$.[1]

[1] [Has84] argues that the computational complexity of the projection step can be improved by using spatial data structures. [YMMS92] claims similar results. However, both [Has84] and [YMMS92] attempt to find the projection point $\mathbf{x}$ of a data point by finding the nearest *vertex* to $\mathbf{x}$, and projecting $\mathbf{x}$ to the two line segments incident to the vertex. As we showed earlier in this section, in general, this approach can yield a wrong result.

3.1.3 The Bias of the HS Algorithm

HS observed two sources of bias in the estimation process. *Model bias* occurs when data points are generated by the additive model

$$\mathbf{X} = \mathbf{f}(Y) + \mathbf{e} \tag{41}$$

where Y is uniformly distributed over the domain of the smooth curve $\mathbf{f}$, and $\mathbf{e}$ is bivariate additive noise which is independent of Y. In general, if $\mathbf{f}$ has a curvature, it is not self-consistent so it is not a principal curve of the distribution of $\mathbf{X}$. The self-consistent curve lies *outside* $\mathbf{f}$ from the point of view of the center of the curvature. This bias goes to 0 with the ratio of the noise variance and the radius of the curvature.

Estimation bias occurs because the scatterplot smoothers average over neighborhoods. The estimation bias points towards the center of curvature so usually it has a flattening effect on the estimated curve. Unlike the model bias, the estimation bias can be affected by parameters of the algorithm. The larger the span coefficient w of the running-lines smoother or the penalty coefficient μ of the spline smoother, the larger is the bias. So, in theory it is possible to set these parameters so that the two bias components cancel each other.

HS proposed a simple model for the quantitative analysis of the two bias components. Let $\mathbf{f}$ be an arc length parameterized circle with constant curvature $1/r$, i.e, let

$$\mathbf{f}(t) = \begin{bmatrix} r\cos(t/r) \\ r\sin(t/r) \end{bmatrix}$$

for $t \in I = [-r\pi, r\pi)$. Let the random variable $\mathbf{X}$ be defined by (41). Assume that the noise $\mathbf{e}$ has zero mean and σ^2 variance in both coordinates. HS showed that in this situation the radius r^* of the self-consistent circle $\mathbf{f}^*$ is larger than r. The intuitive explanation of this is that the model (41) seems to generate more mass outside the circle $\mathbf{f}$ than inside (Figure 8(a)). In a quantitative analysis, HS showed that, under certain conditions,

$$r^* \approx r + \frac{\sigma^2}{2r} \tag{42}$$

so the bias inherent in the model (41) is $\sigma^2/2r$. It also follows from the analysis that the distance function at $\mathbf{f}^*$ is

$$\Delta(\mathbf{f}^*) \approx \sigma^2 - \frac{\sigma^4}{4r^2} = \Delta(\mathbf{f}) - \frac{\sigma^4}{4r^2}. \tag{43}$$

The source of the estimation bias is the local averaging procedure used in the HS algorithm designed for data sets. Assume that the principal curve at $t = 0$ is estimated by using data that projects to the curve in the interval $I_\theta = [-r\theta, r\theta]$ (Figure 8(b)). The smoother fits a straight line to

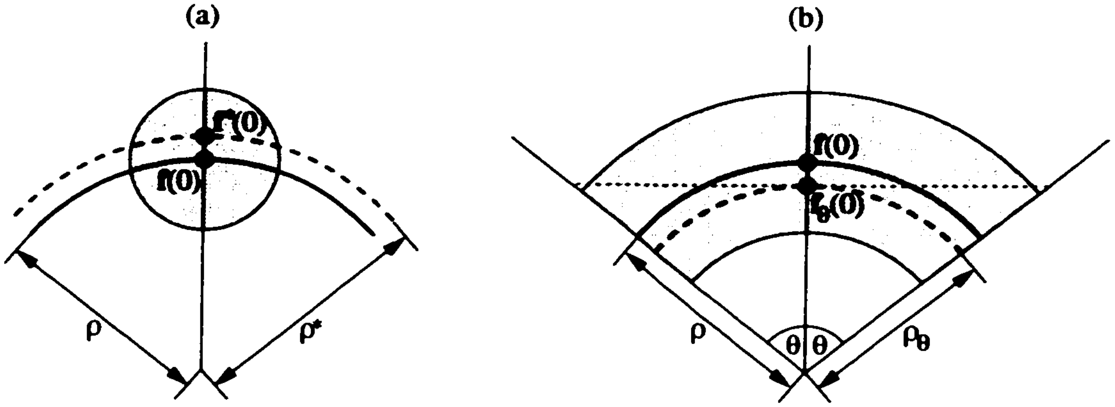


Figure 8: (a) The model bias. There is more mass outside **f** than inside so the self-consistent circle has a larger radius than the generating curve. (b) The estimation bias. A straight line (dotted line) is fitted to the data. The estimated point $\mathbf{f}_\theta(0)$ is *inside* the generating circle.

the data, and the estimate is taken to be the fitted value of the line at $t = 0$. Clearly, the estimate will be *inside* the generating curve. HS showed that under certain conditions the radius of the estimated curve is

$$r_\theta = r^* \frac{\sin(\theta/2)}{\theta/2}. \tag{44}$$

Reducing the Estimation Bias

It follows from (42) and (44) that the span θ can be chosen so that the estimation bias and the model bias are approximately balanced. Unfortunately, for moderate sample sizes, the obtained span tends to be too small. In other words, if the span size is set to an appropriate value for a *given sample size*, the estimation bias tends to be much larger than the model bias. This observation naturally created a demand for procedures to reduce the estimation bias.

Banfield and Raftery [BR92] (hereafter BR) proposed the following modifications to the algorithm. The expectation step (Step 3) in the HS algorithm can be rewritten as

$$\mathbf{f}^{(j+1)}(t) = \mathbf{f}^{(j)}(t) + \mathbf{b}^{(j)}(t)$$

where

$$\mathbf{b}^{(j)}(t) = E\left(\mathbf{X} - \mathbf{f}^{(j+1)}(t) \middle| t_{\mathbf{f}^{(j)}}(\mathbf{X}) = t\right)$$

can be considered as a measure of the bias of $\mathbf{f}^{(j+1)}$ at t. Let

$$\mathbf{p}_i^{(j)} = \mathbf{x}_i - \mathbf{f}^{(j)}\left(t_i^{(j)}\right)$$

denote the *projection residual* of the data point $\mathbf{x}_i$ projected onto $\mathbf{f}^{(j)}$. The bias measure $\mathbf{b}^{(j)}(t)$ is the expected value of the projection residuals of the data points that project onto $\mathbf{f}^{(j)}$ at t. [BR92] suggested that, in the algorithm for data sets, the projection residuals of the data points in $\mathcal{X}$, rather then the data points themselves, should be used to calculate $\mathbf{f}^{(j+1)}(t)$. Accordingly, let

$$\overline{\mathbf{p}}_i^{(j)} = \frac{\sum_{k=1}^n w_{ik}\mathbf{p}_k^{(j)}}{\sum_{k=1}^n w_{ik}}$$

be the weighted average of the projection residuals of the data points that project close to $t_i^{(j)}$. By using $\overline{\mathbf{p}}_i^{(j)}$ as the estimation of the bias $\mathbf{b}^{(j)}\left(t_i^{(j)}\right)$, the new point of the curve is estimated by

$$\mathbf{f}^{(j+1)}\left(t_i^{(j)}\right) = \mathbf{f}^{(j)}\left(t_i^{(j)}\right) + \overline{\mathbf{p}}_i^{(j)}.$$

[BR92] also extended the HS algorithm to closed curves. Experimental results on simulated data are given in Section 5.2, where the HS algorithm with smoothing splines is compared to the BR algorithm and the polygonal line algorithm introduced in Section 5.1.

3.2 Alternative Definitions and Related Concepts

3.2.1 Alternative Definitions of Principal Curves

Two substantially different approaches to principal curves have been proposed subsequent to Hastie and Stuetzle's groundbreaking work. Tibshirani [Tib92] introduced a semi-parametric model for principal curves. The motivation of [Tib92] to redefine principal curves is the unsettling property of the HS principal curves that if the distribution of the data is defined by the additive model $\mathbf{X} = \mathbf{f}(Y) + \mathbf{e}$ (see (41)), $\mathbf{f}$ is not the principal curve of $\mathbf{X}$ in general. To solve this problem, [Tib92] derives principal curves from the additive model (41). Consider a d-dimensional random vector $\mathbf{X} = (X_1, \ldots, X_d)$ with density $\mu_{\mathbf{X}}$. Now imagine that $\mathbf{X}$ was generated in two stages. In the first step, a point on a curve $\mathbf{f}(Y)$ is generated according to some distribution μ_Y, and in the second step, $\mathbf{X}$ is generated from a conditional distribution $\mu_{\mathbf{X}|Y}$ where the mean of $\mu_{\mathbf{X}|Y}$ is $\mathbf{f}(Y)$, and $X_1, \ldots, X_d$ are conditionally independent given Y. Using this model, [Tib92] defines principal curves as follows.

Definition 3 *The* principal curve *of a random variable* $\mathbf{X}$ *is a triplet* $\{\mu_Y, \mu_{\mathbf{X}|Y}, \mathbf{f}\}$ *satisfying the following conditions:*

(i) μ_Y *and* $\mu_{\mathbf{X}|Y}$ *are consistent with* $\mu_{\mathbf{X}}$*, that is,* $\mu_{\mathbf{X}}(\mathbf{x}) = \int \mu_{\mathbf{X}|Y}(\mathbf{x}|y)\mu_Y(y)dy$.

(ii) $X_1, \ldots, X_d$ *are conditionally independent given* Y.

(iii) $\mathbf{f}(t)$ *is a curve in* $\mathbb{R}^d$ *parameterized over a closed interval in* $\mathbb{R}$ *satisfying* $\mathbf{f}(t) = E[\mathbf{X}|Y = t]$.

It is easy to see that if the distribution of the data is defined by the additive model (41), the generating curve $\mathbf{f}$ is indeed the principal curve of $\mathbf{X}$ in the sense of Definition 3. Based on this definition, [Tib92] proposed a semi-parametric scheme for estimating principal curves of data sets. In the model, μ_Y is left completely unspecified, while $\mu_{\mathbf{X}|Y}$ is assumed to be from a parametric family. Therefore, at a certain parameter y, one has to estimate the point of the curve $\mathbf{f}(y)$ and the parameters $\Sigma(y)$ of $\mu_{\mathbf{X}|Y}$. Given a data set $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$, [Tib92] uses maximum likelihood estimation to find the unknown parameters. The log-likelihood

$$l(\mathbf{f}, \Sigma) = \sum_{i=1}^{n} \log \int \mu_{\mathbf{X}|Y}(\mathbf{x}_i | \mathbf{f}(y), \Sigma(y)) \mu_Y(y) dy$$

was minimized by using the EM algorithm [DLR77]. The algorithm was tested on several simulated and real data sets and compared to the HS algorithm. Although Definition 3 has some theoretical advantages over the HS definition, the resulting estimation procedure does not produce better results than the HS algorithm.

Recently, Delicado [Del98] proposed yet another definition based on a property of the first principal components of multivariate normal distributions. Consider a d-dimensional random vector $\mathbf{X} = (X_1, \ldots, X_d)$. [Del98] calls a point $\mathbf{x}^* \in R^d$ a *principal oriented point* if there exists a direction $\mathbf{u}^*(\mathbf{x}^*) \in R^d$ such that $\mathbf{x}^*$ is the conditional mean of $\mathbf{X}$ in the hyperplane orthogonal to $\mathbf{u}^*(\mathbf{x}^*)$ that contains $\mathbf{x}^*$, i.e.,

$$\mathbf{x}^* = E\left[\mathbf{X} | (\mathbf{X} - \mathbf{x}^*)^T \mathbf{u}^*(\mathbf{x}) = 0\right].$$

A curve $\mathbf{f}: [a, b] \to R^d$ is called a *principal curve of oriented points* of $\mathbf{X}$ if for all $t \in [a, b]$, the points of the curve $\mathbf{f}(t)$ are principal oriented points. The definition can be considered as a generalization of PCA since the first principal component of a multivariate normal distribution is a principal curve of oriented points. It can be seen easily that if the curve $\mathbf{f}$ satisfies certain regularity conditions, namely that no points of the support of the distribution of $\mathbf{X}$ can be projected orthogonally to more than one points of $\mathbf{f}$, then $\mathbf{f}$ is a principal curve of oriented points if and only if it is a principal curve in the HS sense. [Del98] also proposed an algorithm to find a principal curve of oriented points of a given data set. Examples indicate that the curves produced by the procedure tend to be less smooth than the curves produced by the HS algorithm.

3.2.2 The Self-Organizing Map

Kohonen's self-organizing map (SOM) [Koh82] is one of the most widely used and most extensively studied unsupervised learning method. The basic idea of the algorithm was inspired by the way the brain forms topology preserving neural representations or maps of various sensory impressions. Keys of the success of the SOM among practitioners are its simplicity, efficiency, and low computational complexity.

In a certain sense, the SOM algorithm can be considered as a generalization of both the GL algorithm and the HS algorithm (with local averaging). The relation of SOM and vector quantization is a widely known fact (see, e.g. [Koh97]), whereas the similarities between SOM and principal curves were pointed out recently by [RMS92] and [MC95]. The SOM algorithm is usually formulated as a stochastic learning algorithm. To emphasize its similarities to the HS and GL algorithms, we present it here as a batch method as it was first formulated by Luttrell [Lut90].

In its original form, the SOM is a nearest neighbor vector quantizer equipped with a topology. Similarly to vector quantization, we are given a set of codepoints $C = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \subset \mathbb{R}^d$. In addition, there is a weighted graph defined over C by a $k \times k$ matrix of weights $\mathbf{W} = \{w_{\ell,m}\}$. The weights $w_{\ell,m}$ $(\ell, m = 1, \ldots, k)$ are usually defined as a monotonically decreasing function of the Euclidean distance between the initial codepoints $\mathbf{v}_\ell$ and $\mathbf{v}_m$. In this sense, $\mathbf{W}$ can be considered as a topology over C. In the simplest case, codepoints are organized in a one-dimensional topology, typically along a line. In practice, the topology is usually two-dimensional, i.e., initial codepoints are placed in a rectangular or hexagonal grid. Three or more-dimensional topologies are rarely used.

The objective of the SOM algorithm is to fit the map to a data set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$ while preserving the predefined topology of the codepoints. On the one hand, the concept of "fitting" suggests that the algorithm minimize a global distortion function or some sort of average distance between the data points and their projections. On the other hand, preserving the topology means that some of the accuracy of the quantization is traded for keeping the smoothness of the topological mapping. In an ideal situation the two criteria can be combined into an objective function which is then minimized by an algorithm. Although in some special cases such objective functions can be defined, in general, no such function exists for the SOM algorithm.

Similarly to all algorithms presented in this chapter, the SOM algorithm alternates between a projection and an expectation step. The projection step is identical to the projection step of the GL algorithm, i.e., each data point is placed into the Voronoi-set of its nearest codepoint. In the expectation step the codepoints are relocated. In the GL algorithm, the new codepoint $\mathbf{v}_\ell$ is set to the center of gravity of the corresponding Voronoi-set V_ℓ. In the SOM algorithm, the new codepoint is a weighted average of *all* data points where the weight of a data point $\mathbf{x}_i$ in effecting the update of $\mathbf{v}_\ell$ depends on how close it projects to $\mathbf{v}_\ell$. Here, "closeness" is measured in the topology defined by $\mathbf{W}$. Formally, let $\ell(i)$ denote the index of the nearest codepoint to $\mathbf{x}_i$ in the jth iteration, that is,

$$\ell(i) = \arg\min_{\ell=1,\ldots,k} \left\| \mathbf{v}_\ell^{(j)} - \mathbf{x}_i \right\|^2.$$

Then the new codepoint is given by

$$\mathbf{v}_\ell^{(j+1)} = \frac{\sum_{i=1}^n w_{\ell,\ell(i)} \mathbf{x}_i}{\sum_{i=1}^n w_{\ell,\ell(i)}}.$$

Although there exists no theoretical proof of the convergence of the algorithm, in practice it is observed to converge. Since there is no known objective function that is minimized by the iteration of the two steps, convergence of the algorithm is, in theory, difficult to determine. The average distortion (7) used in vector quantizer design is guaranteed to decrease in the projection step but may increase in the expectation step. The weighted average distortion defined by

$$\Delta_n(C, \mathbf{W}) = \sum_{\ell=1}^{k} \frac{\sum_{i=1}^{n} w_{\ell,\ell(i)} \|\mathbf{v}_\ell - \mathbf{x}_i\|^2}{\sum_{i=1}^{n} w_{\ell,\ell(i)}}$$

is minimized in the expectation step but may increase in the projection step. In the formal description of the algorithm below we use the "general" distance function Δ indicating that the exact convergence criteria is unknown.

Algorithm 5 (The SOM algorithm)

Step 0 *Set $j = 0$, and set $\mathcal{C}^{(0)} = \left\{\mathbf{v}_1^{(0)}, \ldots, \mathbf{v}_k^{(0)}\right\}$ to an initial codebook.*

Step 1 (Partition) *Construct $\mathcal{V}^{(j)} = \left\{V_1^{(j)}, \ldots, V_k^{(j)}\right\}$ by setting*
$V_i^{(j)} = \left\{\mathbf{x} : \Delta\left(\mathbf{x}, \mathbf{v}_i^{(j)}\right) \leq \Delta\left(\mathbf{x}, \mathbf{v}_m^{(j)}\right),\ m = 1, \ldots, k\right\}$ *for $i = 1, \ldots, k$.*

Step 2 (Expectation) *Construct $\mathcal{C}^{(j+1)} = \left\{\mathbf{v}_1^{(j+1)}, \ldots, \mathbf{v}_k^{(j+1)}\right\}$ by setting*
$\mathbf{v}_\ell^{(j+1)} = \frac{\sum_{i=1}^{n} w_{\ell,\ell(i)} \mathbf{x}_i}{\sum_{i=1}^{n} w_{\ell,\ell(i)}}$ *for $\ell = 1, \ldots, k$ where $\ell(i) = \arg\min_{\ell=1,\ldots,k} \left\|\mathbf{v}_\ell^{(j)} - \mathbf{x}_i\right\|^2$.*

Step 3 *Stop if $\left|1 - \frac{\Delta^{(j+1)}}{\Delta^{(j)}}\right|$ is less than a certain threshold. Otherwise, let $j = j + 1$ and go to Step 1.*

Note that if the weight matrix $\mathbf{W}$ is the identity matrix, the SOM algorithm is identical to the GL algorithm. In practice, the neighborhood width of the codepoints is usually decreased as the optimization proceeds. In the final steps of the algorithm, $\mathbf{W}$ is usually set to the identity matrix, so the SOM and the GL algorithms are equivalent at this point. However, that this does not imply that the resulting final codebooks generated by the algorithms are equivalent.

To illuminate the connection between self-organizing maps and principal curves, consider the HS algorithm with a locally weighted running-line smoother used for local averaging. In the expectation step of the HS algorithm, an $n \times n$ weight matrix is defined by (39) where the weight $w_{\ell,m}$ determines the effect of $\mathbf{x}_m$ in estimating the curve at the projection point of $\mathbf{x}_\ell$. Now consider the SOM algorithm with $k = n$ codepoints running side by side with the HS algorithm on the same data set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Assume that after the jth iteration the n projection points to the principal curve $\mathbf{f}^{(j)}\left(t_1^{(j-1)}\right), \ldots, \mathbf{f}^{(j)}\left(t_n^{(j-1)}\right)$ are identical to the n codepoints $\left\{\mathbf{v}_1^{(j)}, \ldots, \mathbf{v}_n^{(j)}\right\}$ of the SOM, and that the weight matrix $\mathbf{W}$ of the SOM is defined by (39). In this case the estimation procedures in the following expectation steps of the two algorithms are almost identical. The only difference is that the HS algorithm uses weighted least square regression, while the SOM algorithm applies weighted average in computing the new codepoint.

This practically negligible difference originates from a more important conceptual difference, namely, that the objective of the HS algorithm is to find an optimal *curve*, whereas the SOM algorithm optimizes a *set of vertices* equipped with a one-dimensional topology (in this case). The practical similarity of the actual methods then emerges from following two facts. First, the HS algorithm approximates the curve by a polygonal curve so the task is then to optimize the vertices of the polygonal curve. Second, the codepoints produced by the SOM algorithm, when depicted, are usually connected by line segments. The connections here are based on the neighborhood relations generated by the weight matrix $\mathbf{W}$ (such that each "inner" codepoint is connected to its two nearest neighbors, while each "endpoint" is connected to its nearest neighbor), and serve as a tool to visualize the topology. The line segments are not by any means part of the manifold fitted to the data. This conceptual difference is also the source of a major practical difference of the two methods when we consider the *entire optimization*, not only one projection step for which a scenario described above created artificially. This major difference is that the weights of the SOM algorithm are either kept unchanged during the optimization or they are modified deterministically in a data-independent fashion (i.e., the neighborhoods of the codepoints are shrunk as described above in connection with the GL algorithm), whereas the weights (39) of the HS algorithm are reset in every iteration based on the relative positions of the projections points.

We note here that the conceptual difference between principal curves and self-organizing maps will result in a major practical difference between the SOM algorithm and the polygonal line algorithm to be introduced in Chapter 5 for estimating principal curves of data sets. This practical difference and its implications will be discussed in Section 5.1.9.

Limitations of the SOM Algorithm and Principled Alternatives

Despite its simplicity and efficiency, the SOM algorithm has several weaknesses that make its theoretical analysis difficult and limit its practical usefulness. The first and probably most important limitation of the SOM algorithm is that there does not exist any objective function that is minimized by the training process as showed by Erwin et al. [EOS92]. Not only has this limitation theoretical consequences, namely that it is hard to show any analytical properties of the resulting map, but it also makes experimental evaluation difficult. Nevertheless, several recent studies attempt to objectively compare the SOM algorithm to other methods from different aspects. These studies suggest that it is hard to find any criteria under which the SOM algorithm performs better than the traditional techniques used for comparison.

In a study on the efficiency of the SOM algorithm for *data clustering*, Waller et al. [WKIM98] compared the SOM algorithm to five different clustering algorithms on 2850 artificial data sets. In these experiments, zero neighborhood width was used in the final iterations of the SOM algorithm, consequently, it was found that the SOM and the k-means clustering algorithms (the stochastic version of the GL algorithm) performed equally well in terms of the number of misclassified data points (both being better than the other hierarchical clustering methods). The significance of this result is that the nonzero neighborhood width applied in the beginning of the SOM iteration does not improve the clustering performance of the SOM algorithm. It was also shown by Balakrishnan et al. [BCJL94], who compared the SOM algorithm to k-means clustering on 108 multivariate normal clustering problems, that if the neighborhood width does not decrease to zero, the SOM algorithm performs significantly worse than the k-means clustering algorithm.

Evaluating the *topology preservation* capability of the SOM algorithm, Bezdek and Nikhil [BN95] compared the SOM algorithm to traditional multidimensional scaling techniques on seven artificial data sets with different numbers of points and dimensionality, and different shapes of source distributions. The degree of topology preservation of the data was measured via a Spearman rank correlation between the distances of points in the input space and the distances of their projections in the two-dimensional space. [BN95] found that the traditional statistical methods preserve the distances much more effectively than the SOM algorithm. This result was also confirmed by Flexer [Fle99].

In an empirical study on SOM's ability to do *both clustering and topology preservation* in the same time, Flexer [Fle97, Fle99] compared the SOM algorithm to a combined technique of k-means clustering plus Sammon mapping [Sam69] (a traditional statistical method used for multidimensional scaling) on the cluster centers. If zero neighborhood width was used in the final iterations of the SOM algorithm, the SOM algorithm performed almost equally well to the combined algorithm in terms of the number of misclassified data points (confirming the results of [WKIM98]). However, the SOM algorithm performed substantially worse than the combined method in preserving the topology as a consequence of the restriction of the planar grid topology of the SOM. Using a nonzero neighborhood width at the end of the training did not improve the performance of the SOM algorithm significantly.

There have been several attempts to overcome the limitations of the SOM algorithm. Here we briefly describe two alternative models which we selected on the basis of their strong connection to principal curves. The Generative Topographic Mapping (GTM) of Bishop et al. [BSW98] is a principled alternative to SOM. Similarly to Tibshirani's semi-parametric model [Tib92] described in Section 3.2.1, it is assumed that the data was generated by adding an independent Gaussian noise to a vector generated on a nonlinear manifold according to an underlining distribution. To develop a model similar in spirit to the SOM and to make the optimization problem tractable, the latent manifold is assumed to be a set of points of a regular grid. In this sense the GTM can be considered as a "discretized" version of Tibshirani's model (in which the nonlinear manifold is a curve). Similarly to Tibshirani's algorithm, the yielded optimization problem is solved by an EM algorithm. An interesting relationship between the HS algorithm, Tibshirani's algorithm, the GTM algorithm and our polygonal line algorithm is pointed out in Section 5.1.9, after the latter method is described.

Tc overcome the limitations of the SOM algorithm caused by the predefined topology of the cluster centers, Balzuweit et al. [BDHW97, DBH96] proposed a method to adaptively modify the topology during the training process. The basic idea is to set the weight w_{ij} proportional to the number of data points whose nearest and the second nearest neighbors are the cluster centers $\mathbf{v}_i$ and $\mathbf{v}_j$. The resulting dynamic topology is similar to the topology induced by the local averaging rule in the HS algorithm. The main advantage of the method is that it allows the formation of loops and forks during the training process as opposed to the single curve topology of the HS algorithm.

3.2.3 Nonlinear Principal Component Analysis

In the nonlinear PCA model of Kramer [Kra91] the empirical loss minimization principle described in Section 1.1.3 is slightly modified. According to the principle formalized in (3), given a data set $\mathcal{X}_n = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^d$ and a set $\mathcal{F}$ of curves[2], we pick the curve that minimizes the average distance between the data points and the curve. Formally, we minimize

$$\sum_{i=1}^{n} \Delta(\mathbf{x}_i, \mathbf{f}) = \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{f}(t(\mathbf{x}_i))\|^2 \tag{45}$$

over all curves $\mathbf{f} \in \mathcal{F}$ where the projection index $t(\mathbf{x}_i) = t_{\mathbf{f}}(\mathbf{x}_i)$ is a *fixed* function of $\mathbf{f}$ and $\mathbf{x}_i$ defined in (14). On the other hand, in nonlinear PCA the projection index is also subject to optimization, i.e., we minimize (45) with respect to all functions $\mathbf{f} \in \mathcal{F}$ and $t \in \mathcal{T}$. The function classes $\mathcal{F}$ and $\mathcal{T}$ contain continuous smooth functions tailored to the gradient-based optimization method usually used to carry out the optimization of (45). In particular, [Kra91] uses functions of the form

$$t(\mathbf{x}_i) = \sum_{j=1}^{k_1} w_j^{(1)} \sigma\left(\mathbf{w}_j^{(2)} \mathbf{x}_i + b_j^{(2)}\right)$$

and

$$f_i(t) = \sum_{j=1}^{k_2} w_j^{(3)} \sigma\left(w_j^{(4)} t + b_j^{(4)}\right), i = 1, \dots, n$$

where σ is any continuous and monotonically increasing function with $\sigma(x) \to 1$ as $x \to +\infty$ and $\sigma(x) \to 0$ as $x \to -\infty$, and (45) is optimized with respect to the unknown parameters $w_j^{(1)}, b_j^{(2)} \in \mathbb{R}, \mathbf{w}_j^{(2)} \in \mathbb{R}^d, j = 1, \dots, k_1$ and $w_j^{(3)}, w_j^{(4)}, b_j^{(4)} \in R, j = 1, \dots, k_2$. Comparing nonlinear PCA to principal curves, Malthouse et al. [MMT95] pointed out that the main difference between the two

[2] The original model of [Kra91] is more general in the sense that it allows arbitrary-dimensional manifolds. Our purpose here is to compare nonlinear PCA to principal curves, so, for the sake of simplicity and without loss of generality, we describe nonlinear PCA as a curve-fitting method.

models is that principal curves allow the projection index $t(\mathbf{x})$ to be discontinuous at certain points. The required continuity of the projection index causes the nonlinear PCA optimization to find a suboptimal solution $(\hat{\mathbf{f}},\hat{t})$ in the sense that in general, the projection of a point $\mathbf{x}$ will not be the nearest point of $\hat{\mathbf{f}}$ to $\mathbf{x}$, i.e.,

$$\|\mathbf{x}-\hat{\mathbf{f}}(\hat{t}(\mathbf{x}))\| > \inf_t \|\mathbf{x}-\hat{\mathbf{f}}(t)\|.$$

Chapter 4

Learning Principal Curves with a Length Constraint

An unfortunate property of the HS definition is that in general, it is not known if principal curves exist for a given source density. This also makes it difficult to theoretically analyze any estimation scheme for principal curves. In Section 4.1 we propose a new concept of principal curves and prove their existence in the new sense for a large class of source densities. In Section 4.2 we consider the problem of principal curve design based on training data. We introduce and analyze an estimation scheme using a common model in statistical learning theory.

4.1 Principal Curves with a Length Constraint

One of the defining properties of the first principal component is that it minimizes the distance function (18) among all straight lines (Property 2 in Section 2.2.3). We wish to generalize this property of the first principal component and define principal curves so that they minimize the expected squared distance over a class of curves rather than only being critical points of the distance function. To do this it is necessary to constrain the length of the curve since otherwise for any $\mathbf{X}$ with a density and any $\varepsilon > 0$ there exists a smooth curve $\mathbf{f}$ such that $\Delta(\mathbf{f}) \leq \varepsilon$, and thus a minimizing $\mathbf{f}$ has infinite length. On the other hand, if the distribution of $\mathbf{X}$ is concentrated on a polygonal line and is uniform there, the infimum of the squared distances $\Delta(\mathbf{f})$ is 0 over the class of smooth curves but no smooth curve can achieve this infimum. For this reason, we relax the requirement that $\mathbf{f}$ should be differentiable but instead we constrain the length of $\mathbf{f}$. Note that by the definition of curves, $\mathbf{f}$ is still continuous. We give the following new definition of principal curves.

Definition 4 *A curve $\mathbf{f}^*$ is called a* principal curve *of length L for $\mathbf{X}$ if $\mathbf{f}^*$ minimizes $\Delta(\mathbf{f})$ over all curves of length less than or equal to L.*

The relation of our definition and the HS definition (Definition2) is analogous to the relation of a globally optimal vector quantizer and a locally optimal vector quantizer (Section 2.1). Locally optimal vector quantizers are fixed points of the expected distortion $\Delta(q)$ while self-consistent principal curves are fixed points of the distance function $\Delta(\mathbf{f})$. This similarity is further illuminated by a recent work [TLF95] which defines k points $\mathbf{y}_1, \ldots, \mathbf{y}_k$ to be self-consistent if

$$\mathbf{y}_i = E[\mathbf{X} | \mathbf{X} \in V_i]$$

where $V_1, \ldots, V_k$ are the Voronoi regions associated with $\mathbf{y}_1, \ldots, \mathbf{y}_k$. In this sense, our principal curves correspond to globally optimal vector quantizers ("principal points" by the terminology of [TLF95]) while the HS principal curves correspond to self-consistent points.

A useful advantage of the new definition is that principal curves of length L always exist if $\mathbf{X}$ has finite second moments as the next result shows.

Theorem 1 *Assume that $E\|\mathbf{X}\|^2 < \infty$. Then for any $L > 0$ there exists a curve $\mathbf{f}^*$ with $l(\mathbf{f}^*) \leq L$ such that*

$$\Delta(\mathbf{f}^*) = \inf\{\Delta(\mathbf{f}) : l(\mathbf{f}) \leq L\}.$$

Proof Define

$$\Delta^* = \inf\{\Delta(\mathbf{f}) : l(\mathbf{f}) \leq L\}.$$

First we show that the above infimum does not change if we add the restriction that all $\mathbf{f}$ lie inside a closed sphere $S(r) = \{\mathbf{x} : \|\mathbf{x}\| \leq r\}$ of large enough radius r and centered at the origin. Indeed, without excluding nontrivial cases, we can assume that $\Delta^* < E\|\mathbf{X}\|^2$. Denote the distribution of $\mathbf{X}$ by μ and choose $r > 3L$ large enough such that

$$\int_{S(r/3)} \|\mathbf{x}\|^2 \mu(d\mathbf{x}) > \Delta^* + \varepsilon \tag{46}$$

for some $\varepsilon > 0$. If $\mathbf{f}$ is such that $G_\mathbf{f}$ (the graph of $\mathbf{f}$ defined by 17) is not entirely contained in $S(r)$, then for all $\mathbf{x} \in S(r/3)$ we have $\Delta(\mathbf{x}, \mathbf{f}) > \|\mathbf{x}\|^2$ since the diameter of $G_\mathbf{f}$ is at most L. Then (46) implies that

$$\Delta(\mathbf{f}) \geq \int_{S(r/3)} \Delta(\mathbf{x}, \mathbf{f}) \mu(d\mathbf{x}) > \Delta^* + \varepsilon$$

and thus

$$\Delta^* = \inf\{\Delta(\mathbf{f}) : l(\mathbf{f}) \leq L, G_\mathbf{f} \subset S(r)\}. \tag{47}$$

In view of (47) there exists a sequence of curves $\{\mathbf{f}_n\}$ such that $l(\mathbf{f}_n) \leq L$, $G_{\mathbf{f}_n} \subset S(r)$ for all n, and $\Delta(\mathbf{f}_n) \to \Delta^*$. By the discussion preceding (16) in Section 2.2.1, we can assume without loss of generality that all $\mathbf{f}_n$ are defined over $[0,1]$ and

$$\|\mathbf{f}_n(t_1) - \mathbf{f}_n(t_2)\| \leq L|t_1 - t_2| \tag{48}$$

for all $t_1, t_2 \in [0,1]$. Consider the set of all curves $\mathcal{C}$ over $[0,1]$ such that $\mathbf{f} \in \mathcal{C}$ if and only if $\|\mathbf{f}(t_1) - \mathbf{f}(t_2)\| \leq L|t_1 - t_2|$ for all $t_1, t_2 \in [0,1]$ and $G_{\mathbf{f}} \subset S(r)$. It is easy to see that $\mathcal{C}$ is a closed set under the uniform metric $d(\mathbf{f},\mathbf{g}) = \sup_{0 \leq t \leq 1} \|\mathbf{f}(t) - \mathbf{g}(t)\|$. Also, $\mathcal{C}$ is an equicontinuous family of functions and $\sup_t \|\mathbf{f}(t)\|$ is uniformly bounded over $\mathcal{C}$. Thus $\mathcal{C}$ is a compact metric space by the Arzela-Ascoli theorem (see, e.g., [Ash72]). Since $\mathbf{f}_n \in \mathcal{C}$ for all n, it follows that there exists a subsequence $\mathbf{f}_{n_k}$ converging uniformly to an $\mathbf{f}^* \in \mathcal{C}$.

To simplify the notation let us rename $\{\mathbf{f}_{n_k}\}$ as $\{\mathbf{f}_n\}$. Fix $\mathbf{x} \in \mathbb{R}^d$, assume $\Delta(\mathbf{x},\mathbf{f}_n) \geq \Delta(\mathbf{x},\mathbf{f}^*)$, and let $t_{\mathbf{x}}$ be such that $\Delta(\mathbf{x},\mathbf{f}^*) = \|\mathbf{x} - \mathbf{f}^*(t_{\mathbf{x}})\|^2$. Then by the triangle inequality,

$$\begin{aligned}
|\Delta(\mathbf{x},\mathbf{f}^*) - \Delta(\mathbf{x},\mathbf{f}_n)| &= \Delta(\mathbf{x},\mathbf{f}_n) - \Delta(\mathbf{x},\mathbf{f}^*) \\
&\leq \|\mathbf{x} - \mathbf{f}_n(t_{\mathbf{x}})\|^2 - \|\mathbf{x} - \mathbf{f}^*(t_{\mathbf{x}})\|^2 \\
&\leq (\|\mathbf{x} - \mathbf{f}_n(t_{\mathbf{x}})\| + \|\mathbf{x} - \mathbf{f}^*(t_{\mathbf{x}})\|)\,\|\mathbf{f}_n(t_{\mathbf{x}}) - \mathbf{f}^*(t_{\mathbf{x}})\|.
\end{aligned}$$

By symmetry, a similar inequality holds if $\Delta(\mathbf{x},\mathbf{f}_n) < \Delta(\mathbf{x},\mathbf{f}^*)$. Since $G_{\mathbf{f}^*}, G_{\mathbf{f}_n} \subset S(r)$, and $E\|\mathbf{X}\|^2$ is finite, there exists $A > 0$ such that

$$E\,|\Delta(\mathbf{X},\mathbf{f}_n) - \Delta(\mathbf{X},\mathbf{f}^*)| \leq A \sup_{0 \leq t \leq 0} \|\mathbf{f}_n(t) - \mathbf{f}^*(t)\|$$

and therefore

$$\Delta^* = \lim_{n\to\infty} \Delta(\mathbf{f}_n) = \Delta(\mathbf{f}^*).$$

Since the Lipschitz condition on $\mathbf{f}^*$ guarantees that $l(\mathbf{f}^*) \leq L$, the proof is complete. □

Note that we have dropped the requirement of the HS definition that principal curves be non-intersecting. In fact, Theorem 1 does not hold in general for non-intersecting curves of length L without further restricting the distribution of $\mathbf{X}$ since there are distributions for which the minimum of $\Delta(\mathbf{f})$ is achieved only by an intersecting curve even though non-intersecting curves can arbitrarily approach this minimum. Note also that neither the HS nor our definition guarantees the uniqueness of principal curves. In our case, there might exist several principal curves for a given length constraint L but each of these will have the same (minimal) squared loss.

Finally, we note that although principal curves of a given length always exist, it appears difficult to demonstrate concrete examples unless the distribution of $\mathbf{X}$ is discrete or it is concentrated on a curve. It is presently unknown what principal curves look like with a length constraint for even the simplest continuous multivariate distributions such as the Gaussian. However, this fact in itself does not limit the operational significance of principal curves. The same problem occurs in the theory of optimal vector quantizers (Section 2.1.1) where, except for the scalar case ($d = 1$), the structure of optimal quantizers with $k > 2$ codepoints is unknown for even the most common multivariate densities. Nevertheless, algorithms for quantizer design attempting to find near optimal vector quantizers are of great theoretical and practical interest.

4.2 Learning Principal Curves

Suppose that n independent copies $\mathbf{X}_1, \ldots, \mathbf{X}_n$ of $\mathbf{X}$ are given. These are called the *training data* and they are assumed to be independent of $\mathbf{X}$. The goal is to use the training data to construct a curve of length at most L whose expected squared loss is close to that of a principal curve for $\mathbf{X}$.

Our method is based on a common model in statistical learning theory (e.g., see [Vap98]). We consider classes $\mathcal{S}_1, \mathcal{S}_2, \ldots$ of curves of increasing complexity. Given n data points drawn independently from the distribution of $\mathbf{X}$, we choose a curve as the estimator of the principal curve from the kth model class $\mathcal{S}_k$ by minimizing the empirical error. By choosing the complexity of the model class appropriately as the size of the training data grows, the chosen curve represents the principal curve with increasing accuracy.

We assume that the distribution of $\mathbf{X}$ is concentrated on a closed and bounded convex set $K \subset \mathbb{R}^d$. The following lemma shows that there exists a principal curve of length L *inside* K, and so we will only consider curves in K.

Lemma 1 *Assume that $P\{X \in K\} = 1$ for a closed and convex set K, and let $\mathbf{f}$ be a curve with $l(\mathbf{f}) \leq L$. Then there exists a curve $\hat{\mathbf{f}}$ such that $G_{\hat{\mathbf{f}}} \subset K$, $l(\hat{\mathbf{f}}) \leq L$, and*

$$\Delta(\hat{\mathbf{f}}) \leq \Delta(\mathbf{f}).$$

Proof For each t in the domain of $\mathbf{f}$, let $\hat{\mathbf{f}}(t)$ be the unique point in K such that $\|\mathbf{f}(t) - \hat{\mathbf{f}}(t)\| = \min_{\mathbf{x} \in K} \|\mathbf{f}(t) - \mathbf{x}\|$. It is well known that $\hat{\mathbf{f}}(t)$ satisfies

$$(\mathbf{f}(t) - \hat{\mathbf{f}}(t))^T (\mathbf{x} - \hat{\mathbf{f}}(t)) \leq 0, \quad \text{for all } \mathbf{x} \in K. \tag{49}$$

Then for all t_1, t_2 we have

$$\begin{aligned} \|\mathbf{f}(t_1) - \mathbf{f}(t_2)\|^2 &= \|\hat{\mathbf{f}}(t_1) - \hat{\mathbf{f}}(t_2)\|^2 + \|\mathbf{f}(t_1) - \hat{\mathbf{f}}(t_1) + \hat{\mathbf{f}}(t_2) - \mathbf{f}(t_2)\|^2 + \\ &\quad 2(\hat{\mathbf{f}}(t_1) - \hat{\mathbf{f}}(t_2))^T (\mathbf{f}(t_1) - \hat{\mathbf{f}}(t_1)) + 2(\hat{\mathbf{f}}(t_1) - \hat{\mathbf{f}}(t_2))^T (\hat{\mathbf{f}}(t_2) - \mathbf{f}(t_2)) \\ &\geq \|\hat{\mathbf{f}}(t_1) - \hat{\mathbf{f}}(t_2)\|^2 \end{aligned}$$

where the inequality follows from (49) since $\hat{\mathbf{f}}(t_1), \hat{\mathbf{f}}(t_2) \in K$. Thus $\hat{\mathbf{f}}(t)$ is continuous (it is a curve) and $l(\hat{\mathbf{f}}) \leq l(\mathbf{f}) \leq L$. A similar inequality shows that for all t and $\mathbf{x} \in K$,

$$\|\mathbf{x} - \hat{\mathbf{f}}(t)\|^2 \leq \|\mathbf{x} - \mathbf{f}(t)\|^2$$

so that $\Delta(\hat{\mathbf{f}}) \leq \Delta(\mathbf{f})$. □

Let $\mathcal{S}$ denote the family of curves taking values in K and having length not greater than L. For $k \geq 1$ let $\mathcal{S}_k$ be the set of polygonal (piecewise linear) curves in K which have k segments and whose lengths do not exceed L. Note that $\mathcal{S}_k \subset \mathcal{S}$ for all k. Let $\Delta(\mathbf{x}, \mathbf{f})$ denote the squared distance between a point $\mathbf{x} \in \mathbb{R}^d$ and the curve $\mathbf{f}$ as defined in (15). For any $\mathbf{f} \in \mathcal{S}$ the empirical squared error of $\mathbf{f}$ on the training data is the sample average

$$\Delta_n(\mathbf{f}) = \frac{1}{n} \sum_{i=1}^{n} \Delta(\mathbf{X}_i, \mathbf{f}) \tag{50}$$

where we have suppressed in the notation the dependence of $\Delta_n(\mathbf{f})$ on the training data. Let our theoretical algorithm[1] choose an $\mathbf{f}_{k,n} \in \mathcal{S}_k$ which minimizes the empirical error, i.e,

$$\mathbf{f}_{k,n} = \arg\min_{\mathbf{f} \in \mathcal{S}_k} \Delta_n(\mathbf{f}). \tag{51}$$

We measure the efficiency of $\mathbf{f}_{k,n}$ in estimating $\mathbf{f}^*$ by the difference $J(\mathbf{f}_{k,n})$ between the expected squared loss of $\mathbf{f}_{k,n}$ and the optimal expected squared loss achieved by $\mathbf{f}^*$, i.e., we let

$$J(\mathbf{f}_{k,n}) = \Delta(\mathbf{f}_{k,n}) - \Delta(\mathbf{f}^*) = \Delta(\mathbf{f}_{k,n}) - \min_{\mathbf{f} \in \mathcal{S}} \Delta(\mathbf{f}).$$

Since $\mathcal{S}_k \subset \mathcal{S}$, we have $J(\mathbf{f}_{k,n}) \geq 0$. Our main result in this chapter proves that if the number of data points n tends to infinity, and k is chosen to be proportional to $n^{1/3}$, then $J(\mathbf{f}_{k,n})$ tends to zero at a rate $J(\mathbf{f}_{k,n}) = O(n^{-1/3})$.

Theorem 2 *Assume that $P\{\mathbf{X} \in K\} = 1$ for a bounded and closed convex set K, let n be the number of training points, and let k be chosen to be proportional to $n^{1/3}$. Then the expected squared loss of the empirically optimal polygonal line with k segments and length at most L converges, as $n \to \infty$, to the squared loss of the principal curve of length L at a rate*

$$J(\mathbf{f}_{k,n}) = O(n^{-1/3}).$$

The proof of the theorem is given below. To establish the result we use techniques from statistical learning theory (e.g., see [DGL96]). First, the approximating capability of the class of curves

[1] The term "hypothetical algorithm" might appear to be more accurate since we have not shown that an algorithm for finding $\mathbf{f}_{k,n}$ exists. However, an algorithm clearly exists which can approximate $\mathbf{f}_{k,n}$ with arbitrary accuracy in a finite number of steps (consider polygonal lines whose vertices are restricted to a fine rectangular grid). The proof of Theorem 2 shows that such approximating curves can replace $\mathbf{f}_{k,n}$ in the analysis.

$\mathcal{S}_k$ is considered, and then the estimation (generalization) error is bounded via covering the class of curves $\mathcal{S}_k$ with ε accuracy (in the squared distance sense) by a discrete set of curves. When these two bounds are combined, one obtains

$$J(\mathbf{f}_{k,n}) \le \sqrt{\frac{kC(L,D,d)}{n}} + \frac{DL+2}{k} + O(n^{-1/2}) \tag{52}$$

where the term $C(L,D,d)$ depends only on the dimension d, the length L, and the diameter D of the support of $\mathbf{X}$, but is independent of k and n. The two error terms are balanced by choosing k to be proportional to $n^{1/3}$ which gives the convergence rate of Theorem 2.

Remarks

1. Although the constant hidden in the O notation depends on the dimension d, the exponent of n is dimension-free. This is not surprising in view of the fact that the class of curves $\mathcal{S}$ is equivalent in a certain sense to the class of Lipschitz functions $\mathbf{f}\colon [0,1] \to K$ such that $\|\mathbf{f}(x) - \mathbf{f}(y)\| \le L|x-y|$ (see (16) in Section 2.2.1). It is known that the ε-entropy, defined by the logarithm of the ε covering number, is roughly proportional to $1/\varepsilon$ for such function classes [KT61]. Using this result, the convergence rate $O(n^{-1/3})$ can be obtained by considering ε-covers of $\mathcal{S}$ directly (without using the model classes $\mathcal{S}_k$) and picking the empirically optimal curve in this cover. The use of the classes $\mathcal{S}_k$ has the advantage that they are directly related to the practical implementation of the algorithm given in the next section.

2. Even though Theorem 2 is valid for any given length constraint L, the theoretical algorithm itself gives little guidance about how to choose L. This choice depends on the particular application and heuristic considerations are likely to enter here. One example is given in Chapter 5 where a practical implementation of the polygonal line algorithm is used to recover a "generating curve" from noisy observations.

3. The proof of Theorem 2 also provides information on the distribution of the expected squared error of $\mathbf{f}_{k,n}$ given the training data $\mathbf{X}_1,\ldots,\mathbf{X}_n$. In particular, it is shown at the end of the proof that for all n and k, and δ such that $0<\delta<1$, with probability at least $1-\delta$ we have

$$E\left[\Delta(\mathbf{X},\mathbf{f}_{k,n})|\mathbf{X}_1,\ldots,\mathbf{X}_n\right] - \Delta(\mathbf{f}^*) \le \sqrt{\frac{kC(L,D,d) - D^4\log(\Delta/2)}{n}} + \frac{DL+2}{k} \tag{53}$$

where log denotes natural logarithm and $C(L,D,d)$ is the same constant as in (52).

4. Recently, Smola et al. [SWS98] obtained $O(n^{-1/(2+\alpha)})$ convergence rate using a similar but more general model where the value of α depends on the particular regularizer used in the model. [SWS98] pointed out that although there exist regularizers with $\alpha < 1$, in the particular case of a length constraint, $\alpha = 2$ so the obtained convergence rate is $O(n^{-1/4})$.

Proof of Theorem 2 Let $\mathbf{f}_k^*$ denote the curve in $\mathcal{S}_k$ minimizing the squared loss, i.e.,

$$\mathbf{f}_k^* = \arg\min_{\mathbf{f}\in\mathcal{S}_k} \Delta(\mathbf{f}).$$

The existence of a minimizing $\mathbf{f}_k^*$ can easily be shown using a simpler version of the proof of Lemma 1. Then $J(\mathbf{f}_{k,n})$ can be decomposed as

$$J(\mathbf{f}_{k,n}) = (\Delta(\mathbf{f}_{k,n}) - \Delta(\mathbf{f}_k^*)) + (\Delta(\mathbf{f}_k^*) - \Delta(\mathbf{f}^*))$$

where, using standard terminology, $\Delta(\mathbf{f}_{k,n}) - \Delta(\mathbf{f}_k^*)$ is called the *estimation error* and $\Delta(\mathbf{f}_k^*) - \Delta(\mathbf{f}^*)$ is called the *approximation error*. We consider these terms separately first, and then choose k as a function of the training data size n to balance the obtained upper bounds in an asymptotically optimal way.

Approximation Error

For any two curves $\mathbf{f}$ and $\mathbf{g}$ of finite length define their (nonsymmetric) distance by

$$\rho(\mathbf{f},\mathbf{g}) = \max_t \min_s \|\mathbf{f}(t) - \mathbf{g}(s)\|.$$

Note that $\rho(\hat{\mathbf{f}},\hat{\mathbf{g}}) = \rho(\mathbf{f},\mathbf{g})$ if $\hat{\mathbf{f}} \sim \mathbf{f}$ and $\hat{\mathbf{g}} \sim \mathbf{g}$, i.e., $\rho(\mathbf{f},\mathbf{g})$ is independent of the particular choice of the parameterization within equivalence classes. Next we observe that if the diameter of K is D, and $G_\mathbf{f}, G_\mathbf{g} \in K$, then for all $\mathbf{x} \in K$,

$$\Delta(\mathbf{x},\mathbf{g}) - \Delta(\mathbf{x},\mathbf{f}) \le 2D\rho(\mathbf{f},\mathbf{g}), \tag{54}$$

and therefore

$$\Delta(\mathbf{g}) - \Delta(\mathbf{f}) \le 2D\rho(\mathbf{f},\mathbf{g}). \tag{55}$$

To prove (54), let $\mathbf{x} \in K$ and choose t' and s' such that $\Delta(\mathbf{x},\mathbf{f}) = \|\mathbf{x} - \mathbf{f}(t')\|^2$ and $\min_s \|\mathbf{g}(s) - \mathbf{f}(t')\| = \|\mathbf{g}(s') - \mathbf{f}(t')\|$. Then

$$\begin{aligned}
\Delta(\mathbf{x},\mathbf{g}) - \Delta(\mathbf{x},\mathbf{f}) &\le \|\mathbf{x} - \mathbf{g}(s')\|^2 - \|\mathbf{x} - \mathbf{f}(t')\|^2 \\
&= \left(\|\mathbf{x} - \mathbf{g}(s')\| + \|\mathbf{x} - \mathbf{f}(t')\|\right)\left(\|\mathbf{x} - \mathbf{g}(s')\| - \|\mathbf{x} - \mathbf{f}(t')\|\right) \\
&\le 2D\|\mathbf{g}(s') - \mathbf{f}(t')\| \\
&\le 2D\rho(\mathbf{f},\mathbf{g}).
\end{aligned}$$

Let $\mathbf{f} \in \mathcal{S}$ be an arbitrary arc length parameterized curve over $[0,L']$ where $L' \le L$. Define $\mathbf{g}$ as a polygonal curve with vertices $\mathbf{f}(0), \mathbf{f}(L'/k), \ldots, \mathbf{f}((k-1)L'/k), \mathbf{f}(L')$. For any $t \in [0,L']$, we have $|t - iL'/k| \le L/(2k)$ for some $i \in \{0,\ldots,k\}$. Since $\mathbf{g}(s) = \mathbf{f}(iL'/k)$ for some s, we have

$$\begin{aligned}
\min_s \|\mathbf{f}(t) - \mathbf{g}(s)\| &\le \|\mathbf{f}(t) - \mathbf{f}(iL'/k)\| \\
&\le |t - iL'/k| \le \frac{L}{2k}.
\end{aligned}$$

Note that $l(\mathbf{g}) \leq L'$, by construction, and thus $\mathbf{g} \in \mathcal{S}_k$. Thus for every $\mathbf{f} \in \mathcal{S}$ there exists a $\mathbf{g} \in \mathcal{S}_k$ such that $\rho(\mathbf{f}, \mathbf{g}) \leq L/(2k)$. Now let $\mathbf{g} \in \mathcal{S}_k$ be such that $\rho(\mathbf{f}^*, \mathbf{g}) \leq L/(2k)$. Then by (55) we conclude that the approximation error is upper bounded as

$$\begin{aligned} \Delta(\mathbf{f}_k^*) - \Delta(\mathbf{f}^*) &\leq \Delta(\mathbf{g}) - \Delta(\mathbf{f}^*) \\ &\leq 2D\rho(\mathbf{f}^*, \mathbf{g}) \\ &\leq \frac{DL}{k}. \end{aligned} \tag{56}$$

Estimation Error

For each $\varepsilon > 0$ and $k \geq 1$ let $\mathcal{S}_{k,\varepsilon}$ be a *finite* set of curves in K which form an ε-cover of $\mathcal{S}_k$ in the following sense. For any $\mathbf{f} \in \mathcal{S}_k$ there is an $\mathbf{f}' \in \mathcal{S}_{k,\varepsilon}$ which satisfies

$$\sup_{\mathbf{x} \in K} |\Delta(\mathbf{x}, \mathbf{f}) - \Delta(\mathbf{x}, \mathbf{f}')| \leq \varepsilon. \tag{57}$$

The explicit construction of $\mathcal{S}_{k,\varepsilon}$ is given below in Lemma 2. Since $\mathbf{f}_{k,n} \in \mathcal{S}_k$ (see (51)), there exists an $\mathbf{f}'_{k,n} \in \mathcal{S}_{k,\varepsilon}$ such that $|\Delta(\mathbf{x}, \mathbf{f}_{k,n}) - \Delta(\mathbf{x}, \mathbf{f}'_{k,n})| \leq \varepsilon$ for all $\mathbf{x} \in K$. We introduce the compact notation $\mathcal{X}_n = (\mathbf{X}_1, \ldots, \mathbf{X}_n)$ for the training data. Thus we can write

$$\begin{aligned} E[\Delta(\mathbf{X}, \mathbf{f}_{k,n})|\mathcal{X}_n] - \Delta(\mathbf{f}_k^*) &= E[\Delta(\mathbf{X}, \mathbf{f}_{k,n})|\mathcal{X}_n] - \Delta_n(\mathbf{f}_{k,n}) + \Delta_n(\mathbf{f}_{k,n}) - \Delta(\mathbf{f}_k^*) \\ &\leq 2\varepsilon + E[\Delta(\mathbf{X}, \mathbf{f}'_{k,n})|\mathcal{X}_n] - \Delta_n(\mathbf{f}'_{k,n}) + \Delta_n(\mathbf{f}_{k,n}) - \Delta(\mathbf{f}_k^*) && (58) \\ &\leq 2\varepsilon + E[\Delta(\mathbf{X}, \mathbf{f}'_{k,n})|\mathcal{X}_n] - \Delta_n(\mathbf{f}'_{k,n}) + \Delta_n(\mathbf{f}_k^*) - \Delta(\mathbf{f}_k^*) && (59) \\ &\leq 2\varepsilon + 2 \cdot \max_{\mathbf{f} \in \mathcal{S}_{k,\varepsilon} \cup \{\mathbf{f}_k^*\}} |\Delta(\mathbf{f}) - \Delta_n(\mathbf{f})| && (60) \end{aligned}$$

where (58) follows from the approximating property of $\mathbf{f}'_{k,n}$ and the fact that the distribution of $\mathbf{X}$ is concentrated on K. (59) holds because $\mathbf{f}_{k,n}$ minimizes $\Delta_n(\mathbf{f})$ over all $\mathbf{f} \in \mathcal{S}_k$, and (60) follows because given $\mathcal{X}_n = (\mathbf{X}_1, \ldots, \mathbf{X}_n)$, $E[\Delta(\mathbf{X}, \mathbf{f}'_{k,n})|\mathcal{X}_n]$ is an ordinary expectation of the type $E[\Delta(\mathbf{X}, \mathbf{f})]$, $\mathbf{f} \in \mathcal{S}_{k,\varepsilon}$. Thus, for any $t > 2\varepsilon$ the union bound implies

$$\begin{aligned} &P\{E[\Delta(\mathbf{X}, \mathbf{f}_{k,n})|\mathcal{X}_n] - \Delta(\mathbf{f}_k^*) > t\} \\ &\quad \leq P\left\{\max_{\mathbf{f} \in \mathcal{S}_{k,\varepsilon} \cup \{\mathbf{f}_k^*\}} |\Delta(\mathbf{f}) - \Delta_n(\mathbf{f})| > \frac{t}{2} - \varepsilon\right\} \\ &\quad \leq (|\mathcal{S}_{k,\varepsilon}| + 1) \max_{\mathbf{f} \in \mathcal{S}_{k,\varepsilon} \cup \{\mathbf{f}_k^*\}} P\left\{|\Delta(\mathbf{f}) - \Delta_n(\mathbf{f})| > \frac{t}{2} - \varepsilon\right\} && (61) \end{aligned}$$

where $|\mathcal{S}_{k,\varepsilon}|$ denotes the cardinality of $\mathcal{S}_{k,\varepsilon}$.

Recall now Hoeffding's inequality [Hoe63] which states that if $Y_1, Y_2, \ldots, Y_n$ are independent and identically distributed real random variables such that $0 \leq Y_i \leq A$ with probability one, then for all $u > 0$,

$$P\left\{\left|\frac{1}{n}\sum_{i=1}^{n} Y_i - E[Y_1]\right| > u\right\} \leq 2e^{-2nu^2/A^2}.$$

Since the diameter of K is D, we have $\|\mathbf{x}-\mathbf{f}(t)\|^2 \leq D^2$ for all $\mathbf{x}\in K$ and $\mathbf{f}$ such that $G_{\mathbf{f}}\in K$. Thus $0\leq \Delta(\mathbf{X},\mathbf{f})\leq D^2$ with probability one and by Hoeffding's inequality, for all $\mathbf{f}\in S_{k,\varepsilon}\cup\{\mathbf{f}^*\}$ we have

$$P\left\{|\Delta(\mathbf{f})-\Delta_n(\mathbf{f})|>\frac{t}{2}-\varepsilon\right\}\leq 2e^{-2n((t/2)-\varepsilon)^2/D^4}$$

which implies by (61) that

$$P\{E[\Delta(\mathbf{X},\mathbf{f}_{k,n})|X_n]-\Delta(\mathbf{f}_k^*)>t\}\leq 2\,(|S_{k,\varepsilon}|+1)\,e^{-2n((t/2)-\varepsilon)^2/D^4} \tag{62}$$

for any $t>2\varepsilon$. Using the fact that $E[Y]=\int_0^\infty P\{Y>t\}\,dt$ for any nonnegative random variable Y, we can write for any $u>0$,

$$\begin{aligned}
\Delta(\mathbf{f}_{k,n})-\Delta(\mathbf{f}_k^*) &\leq \int_0^\infty P\{E[\Delta(\mathbf{X},\mathbf{f}_{k,n})|X_n]-\Delta(\mathbf{f}_k^*)>t\}\,dt\\
&\leq u+2\varepsilon+2\,(|S_{k,\varepsilon}|+1)\int_{u+2\varepsilon}^\infty e^{-2n((t/2)-\varepsilon)^2/D^4}\,dt\\
&\leq u+2\varepsilon+2\,(|S_{k,\varepsilon}|+1)\,D^4\cdot\frac{e^{-nu^2/(2D^4)}}{nu} &(63)\\
&\leq \sqrt{\frac{2D^4\log(|S_{k,\varepsilon}|+1)}{n}}+2\varepsilon+O(n^{-1/2}) &(64)
\end{aligned}$$

where (63) follows from the inequality $\int_x^\infty e^{-t^2/2}\,dt<(1/x)e^{-x^2/2}$, for $x>0$, and (64) follows by setting $u=\sqrt{\frac{2D^4\log(|S_{k,\varepsilon}|+1)}{n}}$ where log denotes natural logarithm. The following lemma, which is proven below, demonstrates the existence of a suitable covering set $S_{k,\varepsilon}$.

Lemma 2 *For any* $\varepsilon>0$ *there exists a finite collection of curves* $S_{k,\varepsilon}$ *in* K *such that*

$$\sup_{\mathbf{x}\in K}|\Delta(\mathbf{x},\mathbf{f})-\Delta(\mathbf{x},\mathbf{f}')|\leq\varepsilon$$

and

$$|S_{k,\varepsilon}|\leq 2^{\frac{LD}{\varepsilon}+3k+1}V_d^{k+1}\left(\frac{D^2\sqrt{d}}{\varepsilon}+\sqrt{d}\right)^d\left(\frac{LD\sqrt{d}}{k\varepsilon}+3\sqrt{d}\right)^{kd}$$

where V_d *is the volume of the d-dimensional unit sphere and D is the diameter of K.*

It is not hard to see that setting $\varepsilon=1/k$ in Lemma 2 gives the upper bound

$$2D^4\log(|S_{k,\varepsilon}|+1)\leq kC(L,D,d)$$

where $C(L,D,d)$ does not depend on k. Combining this with (64) and the approximation bound given by (56) results in

$$\Delta(\mathbf{f}_{k,n})-\Delta(\mathbf{f}^*)\leq\sqrt{\frac{kC(L,D,d)}{n}}+\frac{DL+2}{k}+O(n^{-1/2}).$$

The rate at which $\Delta(\mathbf{f}_{k,n})$ approaches $\Delta(\mathbf{f}^*)$ is optimized by setting the number of segments k to be proportional to $n^{1/3}$. With this choice $J(\mathbf{f}_{k,n}) = \Delta(\mathbf{f}_{k,n}) - \Delta(\mathbf{f}^*)$ has the asymptotic convergence rate

$$J(\mathbf{f}_{k,n}) = O(n^{-1/3}),$$

and the proof of Theorem 2 is complete.

To show the bound (53), let $\delta \in (0,1)$ and observe that by (62) we have

$$P\{E[\Delta(\mathbf{X},\mathbf{f}_{k,n})|X_n] - \Delta(\mathbf{f}_k^*) \leq t\} > 1-\delta$$

whenever $t > 2\varepsilon$ and

$$\delta = 2\left(|S_{k,\varepsilon}|+1\right)e^{-2n((t/2)-\varepsilon)^2/D^4}.$$

Solving this equation for t and letting $\varepsilon = 1/k$ as before, we obtain

$$\begin{aligned} t &= \sqrt{\frac{2D^4\log\left(|S_{k,1/k}|+1\right) - 2D^4\log(\delta/2)}{n}} + \frac{2}{k} \\ &\leq \sqrt{\frac{kC(L,D,d) - 2D^4\log(\delta/2)}{n}} + \frac{2}{k}. \end{aligned}$$

Therefore, with probability at least $1-\delta$, we have

$$E[\Delta(\mathbf{X},\mathbf{f}_{k,n})|X_n] - \Delta(\mathbf{f}_k^*) \leq \sqrt{\frac{kC(L,D,d) - 2D^4\log(\delta/2)}{n}} + \frac{2}{k}.$$

Combining this bound with the approximation bound $\Delta(\mathbf{f}_k^*) - \Delta(\mathbf{f}^*) \leq (DL)/k$ gives (53). □

Proof of Lemma 2 Consider a rectangular grid with side length $\delta > 0$ in $\mathbb{R}^d$. With each point $\mathbf{y}$ of this grid associate its Voronoi region (a hypercube of side length δ) defined as the set of points which are closer to $\mathbf{y}$ than to any other points of the grid. Let $K_\delta \subset K$ denote the collection of points of this grid which fall in K plus the projections of those points of the grid to K whose Voronoi regions have nonempty intersections with K. Then we clearly have

$$\max_{\mathbf{x}\in K}\min_{\mathbf{y}\in K_\delta}\|\mathbf{x}-\mathbf{y}\| \leq \frac{\sqrt{d}\delta}{2}. \tag{65}$$

Let $\delta = \varepsilon/(D\sqrt{d})$ and define $S_{k,\varepsilon}$ to be the family of all polygonal curves $\hat{\mathbf{f}}$ having $k+1$ vertices $\hat{\mathbf{y}}_0,\ldots,\hat{\mathbf{y}}_k \in K_\delta$ and satisfying the length constraint

$$l(\hat{\mathbf{f}}) \leq L + k\sqrt{d}\delta. \tag{66}$$

To see that $S_{k,\varepsilon}$ has the desired covering property, let $\mathbf{f} \in S_k$ be arbitrary with vertices $\mathbf{y}_0,\ldots,\mathbf{y}_k$, choose $\hat{\mathbf{y}}_i \in K_\delta$ such that $\|\mathbf{y}_i - \hat{\mathbf{y}}_i\| \leq \sqrt{d}\delta/2$, $i = 0,\ldots,k$, and let $\hat{\mathbf{f}}$ be the polygonal curve with vertices $\hat{\mathbf{y}}_0,\ldots,\hat{\mathbf{y}}_k$. Since $\sum_i\|\mathbf{y}_i - \mathbf{y}_{i-1}\| \leq L$ by the definition of S_k, the triangle inequality implies that $\hat{\mathbf{f}}$ satisfies (66) and thus $\hat{\mathbf{f}} \in \mathcal{S}_{k,\varepsilon}$. On the other hand, without loss of generality, assume that the line segment connecting $\mathbf{y}_{i-1}$ and $\mathbf{y}_i$ and the line segment connecting $\hat{\mathbf{y}}_{i-1}$ and $\hat{\mathbf{y}}_i$ are both linearly parameterized over $[0,1]$. Then

$$\begin{aligned}\max_{0\le t\le 1}\|\mathbf{f}(t)-\hat{\mathbf{f}}(t)\| &= \max_{0\le t\le 1}\|t\mathbf{y}_i+(1-t)\mathbf{y}_{i-1}-t\hat{\mathbf{y}}_i-(1-t)\hat{\mathbf{y}}_{i-1}\| \\ &\le \max_{0\le t\le 1}(t\|\mathbf{y}_i-\hat{\mathbf{y}}_i\|+(1-t)\|\mathbf{y}_{i-1}-\hat{\mathbf{y}}_{i-1}\|) \\ &\le \frac{\sqrt{d}\delta}{2}.\end{aligned}$$

This shows that $\max\{\rho(\mathbf{f},\hat{\mathbf{f}}),\rho(\hat{\mathbf{f}},\mathbf{f})\} \le \sqrt{d}\delta/2$. Then it follows from (54) that $\mathcal{S}_{k,\varepsilon}$ is an ε-cover for $\mathcal{S}_k$ since for all $\mathbf{x}\in K$,

$$\begin{aligned}|\Delta(\mathbf{x},\mathbf{f})-\Delta(\mathbf{x},\hat{\mathbf{f}})| &\le 2D\max\{\rho(\mathbf{f},\hat{\mathbf{f}}),\rho(\hat{\mathbf{f}},\mathbf{f})\} \\ &\le 2D\sqrt{d}\delta/2=\varepsilon.\end{aligned}$$

Let L_i, $i=1,\dots,k$ denote the length of the ith segment of $\hat{\mathbf{f}}$ and let

$$\hat{L}_i = \left\lceil \frac{L_i}{\sqrt{d}\delta} \right\rceil \sqrt{d}\delta$$

where $\lceil x \rceil$ denotes the least integer not less than x. Fix the sequence $\hat{L}_1^k = \hat{L}_1,\dots,\hat{L}_k$ and define $\mathcal{S}_{k,\varepsilon}(\hat{L}_1^k) \subset \mathcal{S}_{k,\varepsilon}$ as the set of all $\hat{\mathbf{f}} \in \mathcal{S}_{k,\varepsilon}$ whose segment lengths generate this particular sequence. To bound $|\mathcal{S}_{k,\varepsilon}(\hat{L}_1^k)|$ note that the first vertex $\hat{\mathbf{y}}_0$ of an $\hat{\mathbf{f}} \in \mathcal{S}_{k,\varepsilon}(\hat{L}_1^k)$ can be any of the points in K_δ which contains as many points as there are Voronoi cells intersecting K. Since the diameter of K is D, there exists a sphere of radius $D+\sqrt{d}\delta$ which contains these Voronoi cells. Thus the cardinality of K_δ can be upper bounded as

$$|K_\delta| \le V_d\left(\frac{D+\sqrt{d}\delta}{\delta}\right)^d$$

where V_d is the volume of the unit sphere in $\mathbb{R}^d$. Assume $\hat{\mathbf{y}}_0,\dots,\hat{\mathbf{y}}_{i-1}$, $1\le i\le k$ has been chosen. Since $\|\hat{\mathbf{y}}_i-\hat{\mathbf{y}}_{i-1}\| = L_i \le \hat{L}_i$, there are no more than

$$V_d\left(\frac{L_i+\sqrt{d}\delta}{\delta}\right)^d \le V_d\left(\frac{\hat{L}_i+\sqrt{d}\delta}{\delta}\right)^d$$

possibilities for choosing $\hat{\mathbf{y}}_i$. Therefore,

$$|\mathcal{S}_{k,\varepsilon}(\hat{L}_1^k)| \le V_d^{k+1}\left(\frac{D+\sqrt{d}\delta}{\delta}\right)^d \prod_{i=1}^k\left(\frac{\hat{L}_i+\sqrt{d}\delta}{\delta}\right)^d.$$

By (66) and the definition of $\hat{L}_i$, we have

$$\frac{1}{k}\sum_{i=1}^k(\hat{L}_i+\sqrt{d}\delta) \le \frac{1}{k}\sum_{i=1}^k(L_i+2\sqrt{d}\delta) \le \frac{L}{k}+3\sqrt{d}\delta. \qquad (67)$$

Therefore, the arithmetic-geometric mean inequality implies that

$$\prod_{i=1}^{k}(\hat{L}_i + \sqrt{d}\delta) \le \left(L/k + 3\sqrt{d}\delta\right)^k,$$

and thus

$$|S_{k,\varepsilon}(\hat{L}_1^k)| \le V_d^{k+1}\left(\frac{D+\sqrt{d}\delta}{\delta}\right)^d \left(\frac{L}{k\delta} + 3\sqrt{d}\right)^{kd}.$$

On the other hand, by (67) we have $\sum_i \frac{\hat{L}_i}{\sqrt{d}\delta} \le \frac{L}{\sqrt{d}\delta} + 2k$ and therefore the number of distinct sequences $\hat{L}_1^k$ is upper bounded by

$$\binom{\lceil \frac{L}{\sqrt{d}\delta} + 2k\rceil + k}{k} = \binom{\lceil \frac{L}{\sqrt{d}\delta}\rceil + 3k}{k} \le 2^{\lceil \frac{L}{\sqrt{d}\delta}\rceil + 3k}.$$

Substituting $\delta = \varepsilon/(D\sqrt{d})$ we obtain

$$\begin{aligned} |S_{k,\varepsilon}| &= \sum_{\hat{L}_1^k} |S_{k,\varepsilon}(\hat{L}_1^k)| \\ &\le 2^{\lceil \frac{LD}{\varepsilon}\rceil + 3k} V_d^{k+1}\left(\frac{D^2\sqrt{d}}{\varepsilon} + \sqrt{d}\right)^d \left(\frac{LD\sqrt{d}}{k\varepsilon} + 3\sqrt{d}\right)^{kd}. \end{aligned}$$

□

Chapter 5

The Polygonal Line Algorithm

Given a set of data points $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$, the task of finding a polygonal curve with k segments and length L which minimizes $\frac{1}{n}\sum_{i=1}^{n} \Delta(\mathbf{x}_i, \mathbf{f})$ is computationally difficult. In this chapter we propose a suboptimal method with reasonable complexity which also picks the length L and the number of segments k of the principal curve automatically. We describe and analyze the algorithm in Section 5.1. Test results on simulated data and comparison with the HS and BR algorithms are presented in Section 5.2.

5.1 The Polygonal Line Algorithm

The basic idea is to start with a straight line segment $\mathbf{f}_{0,n}$, the shortest segment of the first principal component line which contains all of the projected data points, and in each iteration of the algorithm to increase the number of segments by one by adding a new vertex to the polygonal curve produced in the previous iteration. After adding a new vertex, we update the positions of all vertices in an inner loop by minimizing a penalized distance function to produce $\mathbf{f}_{k,n}$. The algorithm stops when k exceeds a threshold. This stopping criterion (described in Section 5.1.1) is based on a heuristic complexity measure, determined by the number of segments k, the number of data points n, and the average squared distance $\Delta_n(\mathbf{f}_{k,n})$. The flow chart of the algorithm is given in Figure 9. The evolution of the curve produced by the algorithm is illustrated in Figure 10.

In the inner loop, we attempt to minimize a penalized distance function defined as

$$G_n(\mathbf{f}) = \Delta_n(\mathbf{f}) + \lambda P(\mathbf{f}) \tag{68}$$

The first component $\Delta_n(\mathbf{f})$ is the average squared distance of points in $\mathcal{X}_n$ from the curve $\mathbf{f}$ defined by (19) on page 22. The second component $P(\mathbf{f})$ is a penalty on the average curvature of the curve

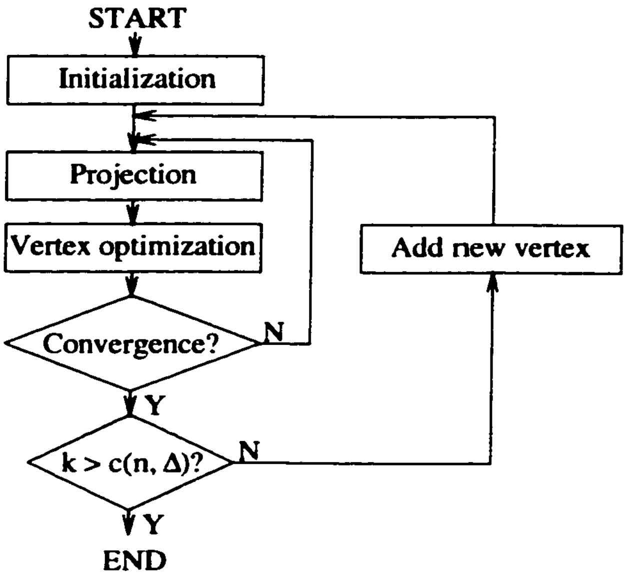


Figure 9: The flow chart of the polygonal line algorithm.

defined by

$$P(\mathbf{f}) = \frac{1}{k+1}\sum_{i=1}^{k+1} P_{\mathbf{v}}(\mathbf{v}_i) \tag{69}$$

where k is the number of segments of $\mathbf{f}$ and $P_{\mathbf{v}}(\mathbf{v}_i)$ is the curvature penalty imposed at vertex $\mathbf{v}_i$. In general, $P_{\mathbf{v}}(\mathbf{v}_i)$ is small if line segments incident to $\mathbf{v}_i$ join smoothly at $\mathbf{v}_i$. An important general property of $P_{\mathbf{v}}(\mathbf{v}_i)$ that it is local in the sense that it can change only if $\mathbf{v}_i$ or immediate neighbors of $\mathbf{v}_i$ are relocated. The exact form of $P_{\mathbf{v}}(\mathbf{v}_i)$ is presented in Section 5.1.2.

Achieving a low average distance means that the curve closely fits the data. Keeping $P(\mathbf{f})$ low ensures the smoothness of the curve. The penalty coefficient λ plays the balancing role between these two competing criteria. To achieve robustness, we propose a heuristic data-dependent penalty coefficient in Section 5.1.3.

$G_n(\mathbf{f})$ is a complicated nonlinear function of $\mathbf{f}$ so finding its minimum analytically is impossible. Furthermore, simple gradient-based optimization methods also fail since $G_n(\mathbf{f})$ is not differentiable at certain points. To minimize $G_n(\mathbf{f})$, we iterate between a projection step and a vertex optimization step until convergence (Figure 9). In the projection step, the data points are partitioned into "nearest neighbor regions" according to which segment or vertex they project. The resulting partition is formally defined in Section 5.1.4 and illustrated in Figure 11. In the vertex optimization step (Section 5.1.5), we use a gradient-based method to minimize $G_n(\mathbf{f})$ assuming that the partition computed in the previous projection step does not change. Under this condition the objective function becomes differentiable everywhere so a gradient-based method can be used for finding a local minimum. The drawback is that if the assumption fails to hold, that is, some data points leave

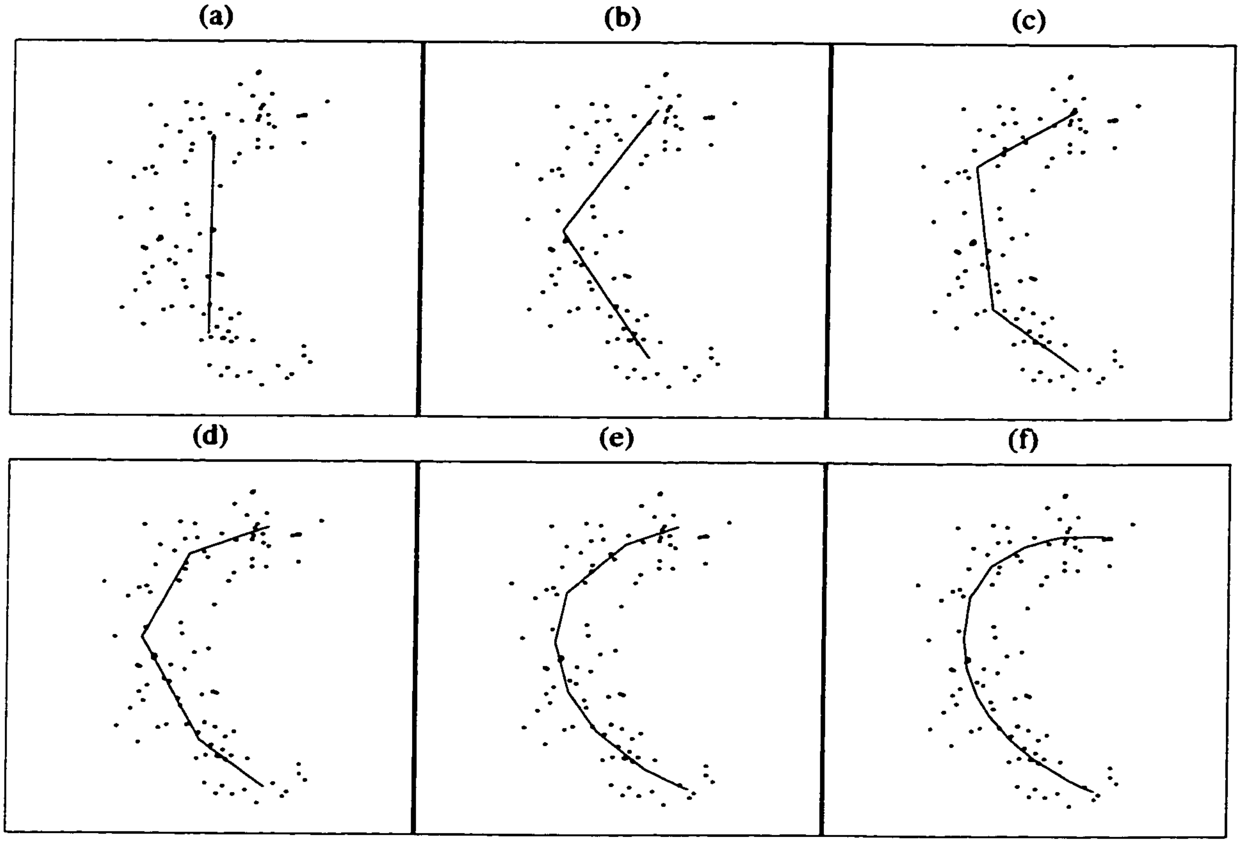


Figure 10: The curves $\mathbf{f}_{k,n}$ produced by the polygonal line algorithm for $n = 100$ data points. The data was generated by adding independent Gaussian errors to both coordinates of a point chosen randomly on a half circle. (a) $\mathbf{f}_{1,n}$, (b) $\mathbf{f}_{2,n}$, (c) $\mathbf{f}_{3,n}$, (d) $\mathbf{f}_{4,n}$, (e) $\mathbf{f}_{8,n}$, (f) $\mathbf{f}_{15,n}$ (the output of the algorithm).

their nearest neighbor regions while vertices of the curve are moved, the objective function $G_n(\mathbf{f})$ might increase in this step. As a consequence, the convergence of the optimizing iteration cannot be guaranteed in theory. In practice, during extensive test runs, however, the algorithm was observed to always converge.

5.1.1 Stopping Condition

According to the theoretical results of Section 4.2, the number of segments k is an important factor that controls the balance between the estimation and approximation errors, and it should be proportional to $n^{1/3}$ to achieve the $O(n^{1/3})$ convergence rate for the expected squared distance. Although the theoretical bounds are not tight enough to determine the optimal number of segments for a given data size, we have found that $k \sim n^{1/3}$ works in practice. We have also found that the final value of k should also depend on the average squared distance to achieve robustness. If the variance of the noise is relatively small, we can keep the approximation error low by allowing a relatively large number of segments. On the other hand, when the variance of the noise is large (implying a high estimation error), a low approximation error does not improve the overall performance significantly, so in this case a smaller number of segments can be chosen. The stopping condition blends these two considerations. The algorithm stops when k exceeds

$$c\big(n,\Delta_n(\mathbf{f}_{k,n})\big) = \beta n^{1/3}\frac{r}{\sqrt{\Delta_n(\mathbf{f}_{k,n})}} \tag{70}$$

where r is the "radius" of the data defined by

$$r = \max_{\mathbf{x}\in\mathcal{X}_n}\left\|\mathbf{x} - \frac{1}{n}\sum_{\mathbf{y}\in\mathcal{X}_n}\mathbf{y}\right\| \tag{71}$$

(included to achieve scale-independence), and β is a parameter of the algorithm which was determined by experiments and was set to the constant value 0.3.

Note that in a practical sense, the number of segments plays a more important role in determining the computational complexity of the algorithm than in measuring the quality of the approximation. Experiments showed that, due to the data-dependent curvature penalty, the number of segments can increase even beyond the number of data points without any indication of overfitting. While increasing the number of segments beyond a certain limit offers only marginal improvement in the approximation, it causes the algorithm to slow down considerably. Therefore, in on-line applications, where speed has priority over precision, it is reasonable to use a smaller number of segments than indicated by (70), and if "aesthetic" smoothness is an issue, to fit a spline through the vertices of the curve (see Section 6.2.2 for an example).

5.1.2 The Curvature Penalty

The most important heuristic component of the algorithm is the curvature penalty $P(\mathbf{v}_i)$ imposed at a vertex $\mathbf{v}_i$. In the theoretical algorithm the average squared distance $\Delta_n(\mathbf{x},\mathbf{f})$ is minimized subject to the constraint that $\mathbf{f}$ is a polygonal line with k segments and length not exceeding L. One could use a Lagrangian formulation and attempt to optimize $\mathbf{f}$ by minimizing a penalized squared error of the form $\Delta_n(\mathbf{f}) + \lambda l(\mathbf{f})^2$. Although this direct length penalty can work well in certain applications, it yields poor results in terms of recovering a smooth generating curve. In particular, this approach is very sensitive to the choice of λ and tends to produce curves which, similarly to the HS algorithm, exhibit a "flattening" estimation bias towards the center of the curvature.

Instead of an explicit length penalty, to ensure smoothness of the curve, we penalize sharp angles between line segments. At inner vertices $\mathbf{v}_i$, $2 \le i \le k$, we penalize the cosine of the angle of the two incident line segment of $\mathbf{v}_i$. The cosine function is convex in the interval $[\pi/2,\pi]$ and its derivative is zero at π which makes it especially suitable for the steepest descent algorithm. To make the algorithm invariant under scaling, we multiply the cosines by the squared radius (71) of the data. At the endpoints ($\mathbf{v}_i$, $i = 1, k+1$), we keep the direct penalty on the squared length of the first (or last) segment as suggested by the theoretical model. Formally, let γ_i denote the angle at vertex $\mathbf{v}_i$, let $\pi(\mathbf{v}_i) = r^2(1+\cos\gamma_i)$, let $\mu_+(\mathbf{v}_i) = \|\mathbf{v}_i - \mathbf{v}_{i+1}\|^2$, and let $\mu_-(\mathbf{v}_i) = \|\mathbf{v}_i - \mathbf{v}_{i-1}\|^2$. Then the penalty imposed at $\mathbf{v}_i$ is defined by

$$P_{\mathbf{v}}(\mathbf{v}_i) = \begin{cases} \mu_+(\mathbf{v}_i) & \text{if } i = 1, \\ \pi(\mathbf{v}_i) & \text{if } 1 < i < k+1, \\ \mu_-(\mathbf{v}_i) & \text{if } i = k+1. \end{cases} \tag{72}$$

Although we do not have a formal proof, we offer the following informal argument to support our observation that the principal curve exhibits a substantially smaller estimation bias if the proposed curvature penalty is used instead of a direct length penalty. When calculating the gradient of the penalty with respect to an inner vertex $\mathbf{v}_i$, it is assumed that all vertices of the curve are fixed except $\mathbf{v}_i$. If a direct penalty on the squared length of the curve is used, the gradient of the penalty can be calculated as the gradient of the *local length penalty* at $\mathbf{v}_i$ ($1 < i < k+1$) defined as

$$P_l(\mathbf{v}_i) = l(\mathbf{s}_{i-1})^2 + l(\mathbf{s}_i)^2 = \|\mathbf{v}_i - \mathbf{v}_{i-1}\|^2 + \|\mathbf{v}_i - \mathbf{v}_{i+1}\|^2.$$

This local length penalty is minimized if the angle at $\mathbf{v}_i$ is π, which means that the gradient vector induced by the penalty always points towards the center of the curvature. If the data is spread equally to the two sides of the generating curve, the distance term cannot balance the inward-pulling effect of the penalty, so the estimated principal curve will always produce a bias towards the center of the curvature. On the other hand, if we penalize sharp angles at $\mathbf{v}_i$ and at the two immediate neighbors of $\mathbf{v}_i$ (the three angles that can change if $\mathbf{v}_i$ is moved while all other vertices are fixed), the minimum is no longer achieved at π but at a smaller angle.

Note that the chosen penalty formulation is related to the original principle of penalizing the length of the curve. At inner vertices, penalizing sharp angles indirectly penalizes long segments. At the endpoints ($\mathbf{v}_i$, $i = 1, k+1$), where penalizing sharp angles does not translate to penalizing long line segments, the penalty on a nonexistent angle is replaced by a direct penalty on the squared length of the first (or last) segment. Also note that although the direct length penalty yields poor results in terms of recovering a smooth generating curve, it may be used effectively under different assumptions.

5.1.3 The Penalty Factor

One important issue is the amount of smoothing required for a given data set. In the HS algorithm one needs to determine the penalty coefficient of the spline smoother, or the span of the scatterplot smoother. In our algorithm, the corresponding parameter is the curvature penalty factor λ. If some a priori knowledge about the distribution is available, one can use it to determine the smoothing parameter. However, in the absence of such knowledge, the coefficient should be data-dependent. Based on heuristic considerations explained below, and after carrying out practical experiments, we set

$$\lambda = \lambda' \cdot \frac{k}{n^{1/3}} \cdot \frac{\sqrt{\Delta_n(\mathbf{f}_{k,n})}}{r} \tag{73}$$

where λ' is a parameter of the algorithm which was determined by experiments and was set to the constant value 0.13.

By setting the penalty to be proportional to the average distance of the data points from the curve, we avoid the zig-zagging behavior of the curve resulting from overfitting when the noise is relatively large. At the same time, this penalty factor allows the principal curve to closely follow the generating curve when the generating curve itself is a polygonal line with sharp angles and the data is concentrated on this curve (the noise is very small).

In our experiments we have found that the algorithm is more likely to avoid local minima if a small penalty is imposed initially and the penalty is gradually increased as the number of segments grows. The number of segments is normalized by $n^{1/3}$ since the final number of segments, according to the stopping condition (Section 5.1.1), is proportional to $n^{1/3}$.

5.1.4 The Projection Step

Let $\mathbf{f}$ denote a polygonal line with vertices $\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}$ and line segments $\mathbf{s}_1, \ldots, \mathbf{s}_k$, such that $\mathbf{s}_i$ connects vertices $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$. In this step the data set $\mathcal{X}_n$ is partitioned into (at most) $2k+1$ disjoint sets $V_1, \ldots, V_{k+1}$ and $S_1, \ldots, S_k$, the nearest neighbor regions of the vertices and segments of $\mathbf{f}$, respectively, in the following manner. For any $\mathbf{x} \in \mathbb{R}^d$ let $\Delta(\mathbf{x}, \mathbf{s}_i)$ be the squared distance from $\mathbf{x}$ to $\mathbf{s}_i$ (see definition (21) on page 23), let $\Delta(\mathbf{x}, \mathbf{v}_i) = \|\mathbf{x} - \mathbf{v}_i\|^2$, and let

$$V_i = \left\{\mathbf{x} \in \mathcal{X}_n : \Delta(\mathbf{x}, \mathbf{v}_i) = \Delta(\mathbf{x}, \mathbf{f}),\ \Delta(\mathbf{x}, \mathbf{v}_i) < \Delta(\mathbf{x}, \mathbf{v}_m),\ m = 1, \ldots, i-1\right\}.$$

Upon setting $V = \bigcup_{i=1}^{k+1} V_i$, the S_i sets are defined by

$$S_i = \left\{\mathbf{x} \in \mathcal{X}_n : \mathbf{x} \notin V, \Delta(\mathbf{x}, \mathbf{s}_i) = \Delta(\mathbf{x}, \mathbf{f}), \Delta(\mathbf{x}, \mathbf{s}_i) < \Delta(\mathbf{x}, \mathbf{s}_m), m = 1, \ldots, i-1\right\}.$$

The resulting partition is illustrated in Figure 11.

5.1.5 The Vertex Optimization Step

In this step we attempt to minimize the penalized distance function (68) assuming that none of the data points leave the nearest neighbor cell of a line segment or a vertex. This is clearly an incorrect assumption but without it we could not use any gradient-based minimization method since the distance of a point $\mathbf{x}$ and the curve is not differentiable (with respect to the vertices of the

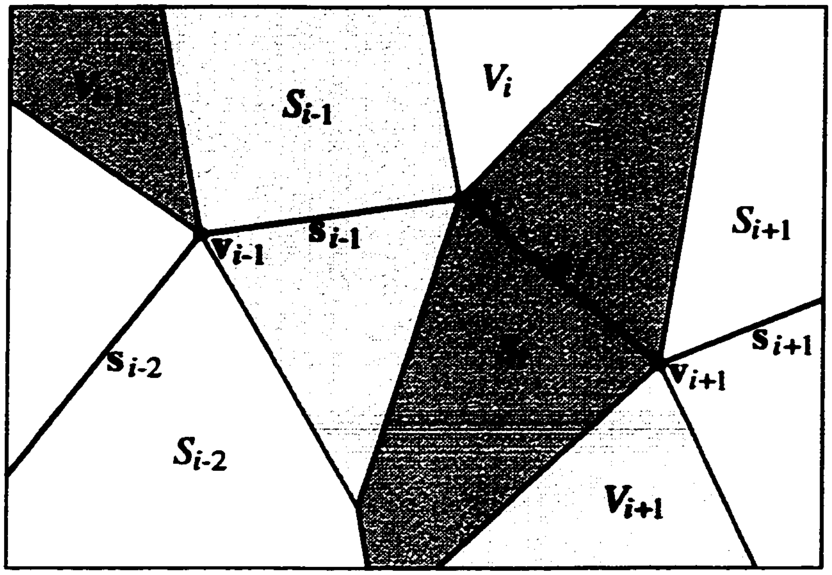


Figure 11: A nearest neighbor partition of $\mathbb{R}^2$ induced by the vertices and segments of $\mathbf{f}$. The nearest point of $\mathbf{f}$ to any point in the set V_i is the vertex $\mathbf{v}_i$. The nearest point of $\mathbf{f}$ to any point in the set S_i is a point of the line segment $\mathbf{s}_i$.

curve) if $\mathbf{x}$ falls on the boundary of two nearest neighbor regions. Also, to check whether a data point has left the nearest neighbor cell of a line segment or a vertex, we would have to execute a projection step each time when a vertex is moved, which is computationally infeasible. Technically, this assumption means that the distance of a data point $\mathbf{x}$ and a line segment $\mathbf{s}_i$ is measured as if $\mathbf{s}_i$ were an infinite line. Accordingly, let $\mathbf{s}'_i$ be the line obtained by the infinite extension of the line segment $\mathbf{s}_i$, let

$$\sigma_+(\mathbf{v}_i) = \sum_{\mathbf{x}\in S_i} \Delta(\mathbf{x},\mathbf{s}'_i),$$

$$\sigma_-(\mathbf{v}_i) = \sum_{\mathbf{x}\in S_{i-1}} \Delta(\mathbf{x},\mathbf{s}'_{i-1}),$$

and

$$\nu(\mathbf{v}_i) = \sum_{\mathbf{x}\in V_i} \Delta(\mathbf{x},\mathbf{v}_i)$$

where $\Delta(\mathbf{x},\mathbf{s}'_i)$ is the Euclidean squared distance of $\mathbf{x}$ and the infinite line $\mathbf{s}'_i$ as defined by (20) on page 22, and define

$$\Delta'_n(\mathbf{f}) = \frac{1}{n}\left(\sum_{i=1}^{k+1} \nu(\mathbf{v}) + \sum_{i=1}^{k} \sigma_+(\mathbf{v}_i)\right).$$

In the vertex optimization step we minimize a "distorted" objective function $G'_n(\mathbf{f}) = \Delta'_n(\mathbf{f}) + \lambda P(\mathbf{f})$. Note that after every projection step, until any data point crosses the boundary of a nearest neighbor cell, the "real" distance function $\Delta_n(\mathbf{f})$ is equal to $\Delta'_n(\mathbf{f})$, so $G_n(\mathbf{f}) = G'_n(\mathbf{f})$.

The gradient of the objective function $G'_n(\mathbf{f})$ with respect to a vertex $\mathbf{v}_i$ can be computed locally in the following sense. On the one hand, only distances of data points that project to $\mathbf{v}_i$ or to the two incident line segments to $\mathbf{v}_i$ can change when $\mathbf{v}_i$ is moved. On the other hand, when the vertex $\mathbf{v}_i$ is moved, only angles at $\mathbf{v}_i$ and at neighbors of $\mathbf{v}_i$ can change. Therefore, the gradient of $G'_n(\mathbf{f})$ with respect to $\mathbf{v}_i$ can be computed as

$$\nabla_{\mathbf{v}_i} G'_n(\mathbf{f}) = \nabla_{\mathbf{v}_i}\big(\Delta'_n(\mathbf{f}) + \lambda P(\mathbf{f})\big) = \nabla_{\mathbf{v}_i}\big(\Delta_n(\mathbf{v}_i) + \lambda P(\mathbf{v}_i)\big)$$

where

$$\Delta_n(\mathbf{v}_i) = \begin{cases} \frac{1}{n}\big(\nu(\mathbf{v}_i) + \sigma_+(\mathbf{v}_i)\big) & \text{if } i = 1 \\ \frac{1}{n}\big(\sigma_-(\mathbf{v}_i) + \nu(\mathbf{v}_i) + \sigma_+(\mathbf{v}_i)\big) & \text{if } 1 < i < k+1 \\ \frac{1}{n}\big(\sigma_-(\mathbf{v}_i) + \nu(\mathbf{v}_i)\big) & \text{if } i = k+1 \end{cases} \tag{74}$$

and

$$P(\mathbf{v}_i) = \begin{cases} \frac{1}{k+1}\big(P_v(\mathbf{v}_i) + P_v(\mathbf{v}_{i+1})\big) & \text{if } i = 1 \\ \frac{1}{k+1}\big(P_v(\mathbf{v}_{i-1}) + P_v(\mathbf{v}_i) + P_v(\mathbf{v}_{i+1})\big) & \text{if } 1 < i < k+1 \\ \frac{1}{k+1}\big(P_v(\mathbf{v}_{i-1}) + P_v(\mathbf{v}_i)\big) & \text{if } i = k+1. \end{cases} \tag{75}$$

Once the gradients $\nabla_{\mathbf{v}_i} G'_n(\mathbf{f})$, $i = 1, \ldots, m$, are computed, a local minimum of $G'_n(\mathbf{f})$ can be obtained by any gradient-based optimization method. We found that the following iterative minimization scheme works particularly well. To find a new position for a vertex $\mathbf{v}_i$, we fix all other vertices and execute a line search in the direction of $-\nabla_{\mathbf{v}_i} G'_n(\mathbf{f})$. This step is repeated for all vertices and the iteration over all vertices is repeated until convergence. The flow chart of the optimization step is given in Figure 12.

5.1.6 Convergence of the Inner Loop

In the vertex optimization step $G'_n(\mathbf{f})$ clearly cannot increase. Unfortunately, $G'_n(\mathbf{f})$ does not always decrease in the projection step. Since the curve is kept unchanged in this step, $P(\mathbf{f})$ is constant but it is possible that $\Delta'_n(\mathbf{f})$ increases. *After* the projection step it is always true that $\Delta'_n(\mathbf{f}) = \Delta_n(\mathbf{f})$ since every data point belongs to the nearest neighbor cell of its nearest vertex or line segment. *Before* the projection step, however, it is possible that $\Delta'_n(\mathbf{f}) < \Delta_n(\mathbf{f})$. The reason is that, contrary to our assumption, there can be data points that leave the nearest neighbor cell of a line segment in the optimization step. For such a data point $\mathbf{x}$, it is possible that the real distance of $\mathbf{x}$ and the curve is larger than it is measured by $\Delta_n(\mathbf{v}_i)$ as indicated by Figure 13.

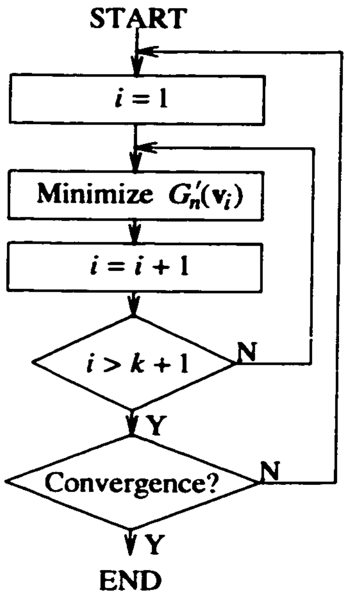


Figure 12: The flow chart of the optimization step.

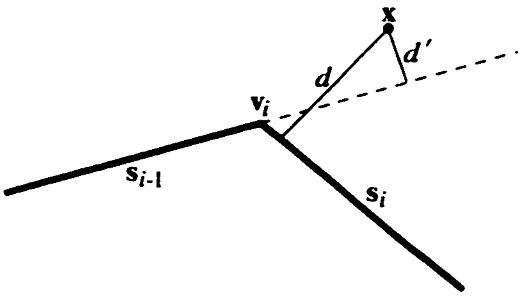


Figure 13: Assume that $\mathbf{x}$ belongs to S_{i-1}. The distance of $\mathbf{x}$ and the curve is d, while $\Delta_n(\mathbf{v}_i)$ measures the distance as d'.

As a consequence, the convergence of the inner loop cannot be guaranteed. In practice, during extensive test runs, however, the algorithm was observed to always converge. We found that if there is any increase in $\Delta'_n(\mathbf{f})$ in the projection step, it is almost always compensated by the decrease of $G_n(\mathbf{f})$ in the optimization step.

5.1.7 Adding a New Vertex

We start with the optimized $\mathbf{f}_{k,n}$ and choose the segment that has the largest number of data points projecting to it. If more than one such segment exist, we choose the longest one. The midpoint of this segment is selected as the new vertex. Formally, let $I = \{i : |S_i| \geq |S_j|,\ j = 1, \ldots, k\}$, and $\ell = \arg\max_{i \in I} \|\mathbf{v}_i - \mathbf{v}_{i+1}\|$. Then the new vertex is $\mathbf{v}_{new} = (\mathbf{v}_\ell + \mathbf{v}_{\ell+1})/2$.

5.1.8 Computational Complexity

The complexity of the inner loop is dominated by the complexity of the projection step, which is $O(nk)$. Increasing the number of segments one at a time (as described in Section 5.1.7), the complexity of the algorithm to obtain $\mathbf{f}_{k,n}$ is $O(nk^2)$. Using the stopping condition of Section 5.1.1, the computational complexity of the algorithm becomes $O(n^{5/3})$. This is slightly better than the $O(n^2)$ complexity of the HS algorithm.

The complexity can be dramatically decreased in certain situations. One possibility is to add more than one vertex at a time. For example, if instead of adding only one vertex, a new vertex is placed at the midpoint of every segment, then we can reduce the computational complexity for producing $\mathbf{f}_{k,n}$ to $O(nk\log k)$. One can also set k to be a constant if the data size is large, since increasing k beyond a certain threshold brings only diminishing returns. Also, k can be naturally set to a constant in certain applications, giving $O(nk)$ computational complexity. These simplifications work well in certain situations, but the original algorithm is more robust.

Note that the optimization of $G_n(\mathbf{v}_i)$ can be done in $O(1)$ time if the sample mean of the data points in V_i, and the sample means and the sample covariance matrices of the data points in S_{i-1} and S_i are stored. The maintenance of these statistics can be done in the projection step when the data points are sorted into the nearest neighbor sets. The statistics must be updated only for data points that are moved from a nearest neighbor set into another in the projection step. The number of such data points tends to be very small as the algorithm progresses so the computational requirements of this operation is negligible compared to other steps of the algorithm.

The projection step can be accelerated by using special data structures. The construction we present here is based on the following two observations. Firstly, when the noise is relatively low and the line segments are relatively long, most of the data points are very far from the second nearest line segment compared to their distance from the curve. Secondly, as the algorithm progresses, the vertices move less and less in the optimization step so most of the data points stay in their original nearest neighbor sets. If we can guarantee that a given data point $\mathbf{x}$ stays in its nearest neighbor set, we can save the time of measuring the distance between $\mathbf{x}$ and each line segment of the curve.

Formally, let $\delta\mathbf{v}^{(j)}$ be the maximum shift of a vertex in the jth optimization step defined by

$$\delta\mathbf{v}^{(j)} = \begin{cases} \infty & \text{if } j = 0 \\ \max\limits_{i=1,\dots,k+1} \left\| \mathbf{v}_i^{(j)} - \mathbf{v}_i^{(j+1)} \right\| & \text{otherwise.} \end{cases}$$

Let the distance between a data point $\mathbf{x}$ and a line segment $\mathbf{s}$ be

$$d(\mathbf{x},\mathbf{s}) = \sqrt{\Delta(\mathbf{x},\mathbf{s})} = \|\mathbf{x} - \mathbf{s}(t_{\mathbf{s}}(\mathbf{x}))\|.$$

First we show that the distance between any data point and any line segment can change at most

$\delta \mathbf{v}^{(j)}$ in the jth optimization step. Let $t_1 = t_{\mathbf{s}^{(j)}}(\mathbf{x})$ and $t_2 = t_{\mathbf{s}^{(j+1)}}(\mathbf{x})$, assume that both $\mathbf{s}^{(j)}$ and $\mathbf{s}^{(j+1)}$ are parameterized over $[0,1]$, and assume that $d\left(\mathbf{x},\mathbf{s}^{(j)}\right) \geq d\left(\mathbf{x},\mathbf{s}^{(j+1)}\right)$. Then we have

$$\begin{aligned}
&\left|d\left(\mathbf{x},\mathbf{s}^{(j)}\right) - d\left(\mathbf{x},\mathbf{s}^{(j+1)}\right)\right| \\
&= d\left(\mathbf{x},\mathbf{s}^{(j)}\right) - d\left(\mathbf{x},\mathbf{s}^{(j+1)}\right) \\
&= \left\|\mathbf{x}-\mathbf{s}^{(j)}(t_1)\right\| - \left\|\mathbf{x}-\mathbf{s}^{(j+1)}(t_2)\right\| \\
&\leq \left\|\mathbf{x}-\mathbf{s}^{(j)}(t_2)\right\| - \left\|\mathbf{x}-\mathbf{s}^{(j+1)}(t_2)\right\| &(76)\\
&\leq \left\|\mathbf{s}^{(j)}(t_2)-\mathbf{s}^{(j+1)}(t_2)\right\| &(77)\\
&= \left\|t_2\mathbf{s}^{(j)}(0)-(1-t_2)\mathbf{s}^{(j)}(1)-t_2\mathbf{s}^{(j+1)}(0)+(1-t_2)\mathbf{s}^{(j+1)}(1)\right\| \\
&\leq t_2\left\|\mathbf{s}^{(j)}(0)-\mathbf{s}^{(j+1)}(0)\right\| + (1-t_2)\left\|\mathbf{s}^{(j+1)}(1)-\mathbf{s}^{(j)}(1)\right\| &(78)\\
&\leq \delta\mathbf{v}^{(j)} &(79)
\end{aligned}$$

where (76) holds because $\mathbf{s}^{(j)}(t_1)$ is the closest point of $\mathbf{s}^{(j)}$ to $\mathbf{x}$, (77) and (78) follows from the triangle inequality, and (79) follows from the assumption that none of the endpoints of $\mathbf{s}^{(j)}$ are shifted by more than $\delta\mathbf{v}^{(j)}$. By symmetry, a similar inequality holds if $d\left(\mathbf{x},\mathbf{s}^{(j)}\right) < d\left(\mathbf{x},\mathbf{s}^{(j+1)}\right)$.

Now consider a data point $\mathbf{x}$, and let $\mathbf{s}_{i_1}^{(j)}$ and $\mathbf{s}_{i_2}^{(j)}$ be the nearest and second nearest line segments to $\mathbf{x}$, respectively. Then if

$$d\left(\mathbf{x},\mathbf{s}_{i_1}^{(j)}\right) \leq d\left(\mathbf{x},\mathbf{s}_{i_2}^{(j)}\right) - 2\delta\mathbf{v}^{(j)}, \tag{80}$$

then for any $i \neq i_1$, we have

$$\begin{aligned}
d\left(\mathbf{x},\mathbf{s}_{i_1}^{(j+1)}\right) &\leq d\left(\mathbf{x},\mathbf{s}_{i_1}^{(j)}\right) + \delta\mathbf{v}^{(j)} &(81)\\
&\leq d\left(\mathbf{x},\mathbf{s}_{i_2}^{(j)}\right) - \delta\mathbf{v}^{(j)} &(82)\\
&\leq d\left(\mathbf{x},\mathbf{s}_{i}^{(j)}\right) - \delta\mathbf{v}^{(j)} &(83)\\
&\leq d\left(\mathbf{x},\mathbf{s}_{i}^{(j+1)}\right) &(84)
\end{aligned}$$

where (81) and (84) follows from (79), (82) follows from (80), and (83) holds since $\mathbf{s}_{i_2}^{(j)}$ is the second nearest line segment to $\mathbf{x}$. (84) means that if (80) holds, $\mathbf{s}_{i_1}$ remains the nearest line segment to $\mathbf{x}$ after the jth optimization step. Furthermore, it is easy to see that after the $(j+j_1)$th optimization step, $\mathbf{s}_{i_1}$ is still the nearest line segment to $\mathbf{x}$ if

$$d\left(\mathbf{x},\mathbf{s}_{i_1}^{(j)}\right) \leq d\left(\mathbf{x},\mathbf{s}_{i_2}^{(j)}\right) - 2\sum_{\ell=j}^{j+j_1}\delta\mathbf{v}^{(\ell)}.$$

Practically, this means that in the subsequent projection steps we only have to decide whether $\mathbf{x}$ belongs to S_{i_1}, V_{i_1}, or V_{i_1+1}. So, by storing the index of the first and second nearest segment for each data point $\mathbf{x}$, and computing the maximum shift $\delta\mathbf{v}^{(j)}$ after each optimization step, we can save a lot of computation in the projection steps especially towards the end of the optimization when $\delta\mathbf{v}^{(j)}$ is relatively small.

5.1.9 Remarks

Heuristic Versus Core Components

It should be noted that the two core components of the algorithm, the projection and the vertex optimization steps, are combined with more heuristic elements such as the stopping condition (70) the form of the penalty term (75) of the optimization step. The heuristic parts of the algorithm have been tailored to the task of recovering an underlying generating curve for a distribution based on a finite data set of randomly drawn points (see the experimental results in Section 5.2). When the algorithm is intended for an application with a different objective, the core components can be kept unchanged but the heuristic elements may be replaced according to the new objectives.

Relationship with the SOM algorithm

As a result of introducing the nearest neighbor regions S_i and V_i, the polygonal line algorithm substantially differs from methods based on the self-organizing map (Section 3.2.2). On the one hand, although we optimize the positions of the *vertices* of the curve, the distances of the data points are measured from the *line segments and vertices* of the curve onto which they project, which means that the manifold fitted to the data set is indeed a polygonal curve. On the other hand, the self-organizing map measures distances *exclusively from the vertices*, and the connections serve only as a tool to visualize the topology of the map. The line segments are not, by any means, part of the manifold fitted to the data set. Therefore, even if the resulted map looks like a polygonal curve (as it does if the topology is one-dimensional), the optimized manifold remains the set of codepoints, not the depicted polygonal curve.

One important practical implication of our principle is that we can use a relatively small number of vertices and still obtain good approximation to an underlying generating curve.

Relationship of Four Unsupervised Learning a Algorithms

There is an interesting informal relationship between the HS algorithm with spline smoothing, the polygonal line algorithm, Tibshirani's semi-parametric model (Section 3.2.1, [Tib92]), and the Generative Topographic Mapping (Bishop et al.'s [BSW98] principled alternative to SOM described briefly in Section 3.2.2). On the one hand, the HS algorithm and the polygonal line algorithm assume a nonparametric model of the source distribution whereas Tibshirani's algorithm and the

GTM algorithm assume that the data was generated by adding an independent Gaussian noise to a vector generated on a nonlinear manifold according to an underlining distribution. On the other hand, the polygonal line algorithm and the GTM algorithm "discretize" the underlining manifold, that is, the number of parameters representing the manifold is substantially less than the number of data points, whereas the HS algorithm and Tibshirani's algorithm represents the manifold by the projection points of *all* data points. Table 1 summarizes the relationship between the four methods.

	"Analogue"	"Discrete"
	number of nodes = number of points	number of nodes < number of points
Semi-parametric	Tibshirani's method	GTM
Nonparametric	HS algorithm with spline smoothing	Polygonal line algorithm

Table 1: The relationship between four unsupervised learning algorithms.

Implementation

The polygonal line algorithm has been implemented in Java, and it is available at the WWW site

`http://www.cs.concordia.ca/~grad/kegl/pcurvedemo.html`

5.2 Experimental Results

We have extensively tested the proposed algorithm on two-dimensional data sets. In most experiments the data was generated by a commonly used (see, e.g., [HS89, Tib92, MC95]) additive model

$$\mathbf{X} = \mathbf{Y} + \mathbf{e} \tag{85}$$

where $\mathbf{Y}$ is uniformly distributed on a smooth planar curve (hereafter called the *generating curve*) and $\mathbf{e}$ is bivariate additive noise which is independent of $\mathbf{Y}$.

In Section 5.2.1 we compare the polygonal line algorithm, the HS algorithm, and, for closed generating curves, the BR algorithm [BR92]. The various methods are compared subjectively based mainly on how closely the resulting curve follows the shape of the generating curve. We use varying generating shapes, noise parameters, and data sizes to demonstrate the robustness of the polygonal line algorithm.

In Section 5.2.2 we analyze the performance of the polygonal line algorithm in a quantitative fashion. These experiments demonstrate that although the generating curve in model (85) is in general not a principal curve either in the HS sense or in our definition, the polygonal line algorithm well approximates the generating curve as the data size grows and as the noise variance decreases.

In Section 5.2.3 we show two scenarios in which the polygonal line algorithm (along with the HS algorithm) fails to produce meaningful results. In the first, the high number of abrupt changes in the direction of the generating curve causes the algorithm to oversmooth the principal curve, even when the data is concentrated on the generating curve. This is a typical situation when the penalty parameter λ' should be decreased. In the second scenario, the generating curve is too complex (e.g., it contains loops, or it has the shape of a spiral), so the algorithm fails to find the global structure of the data if the process is started from the first principal component. To recover the generating curve, one must replace the initialization step by a more sophisticated routine that approximately captures the global structure of the data.

5.2.1 Comparative Experiments

In general, in simulation examples considered by HS, the performance of the new algorithm is comparable with the HS algorithm. Due to the data dependence of the curvature penalty factor and the stopping condition, our algorithm turns out to be more robust to alterations in the data generating model, as well as to changes in the parameters of the particular model.

We use model (85) with varying generating shapes, noise parameters, and data sizes to demonstrate the robustness of the polygonal line algorithm. All plots show the generating curve, the curve produced by our polygonal line algorithm (Polygonal principal curve), and the curve produced by the HS algorithm with spline smoothing (HS principal curve), which we have found to perform better than the HS algorithm using scatterplot smoothing. For closed generating curves we also include the curve produced by the BR algorithm [BR92] (BR principal curve), which extends the HS algorithm to closed curves. The two coefficients of the polygonal line algorithm are set in all experiments to the constant values $\beta = 0.3$ and $\lambda' = 0.13$.

In Figure 14 the generating curve is a circle of radius $r = 1$, the sample size is $n = 100$, and $\mathbf{e} = (e_1, e_2)$ is a zero mean bivariate uncorrelated Gaussian with variance $E(e_i^2) = 0.04$, for $i = 1, 2$. The performance of the three algorithms (HS, BR, and the polygonal line algorithm) is comparable, although the HS algorithm exhibits more bias than the other two. Note that the BR algorithm [BR92] has been tailored to fit closed curves and to reduce the estimation bias. In Figure 15, only half of the circle is used as a generating curve and the other parameters remain the same. Here, too, both the HS and our algorithm behave similarly.

When we depart from these usual settings, the polygonal line algorithm exhibits better behavior than the HS algorithm. In Figure 16(a) the data was generated similarly to the data set of Figure 15, and then it was linearly transformed using the matrix $\begin{pmatrix} 0.7 & 0.4 \\ -0.8 & 1.0 \end{pmatrix}$. In Figure 16(b) the transformation $\begin{pmatrix} -1.0 & -1.2 \\ 1.0 & -0.2 \end{pmatrix}$ was used. The original data set was generated by an S-shaped generating curve, consisting of two half circles of unit radii, to which the same Gaussian noise was added as in Figure 15. In both cases the polygonal line algorithm produces curves that fit the generator curve more closely. This is especially noticeable in Figure 16(a) where the HS principal curve fails to follow the shape

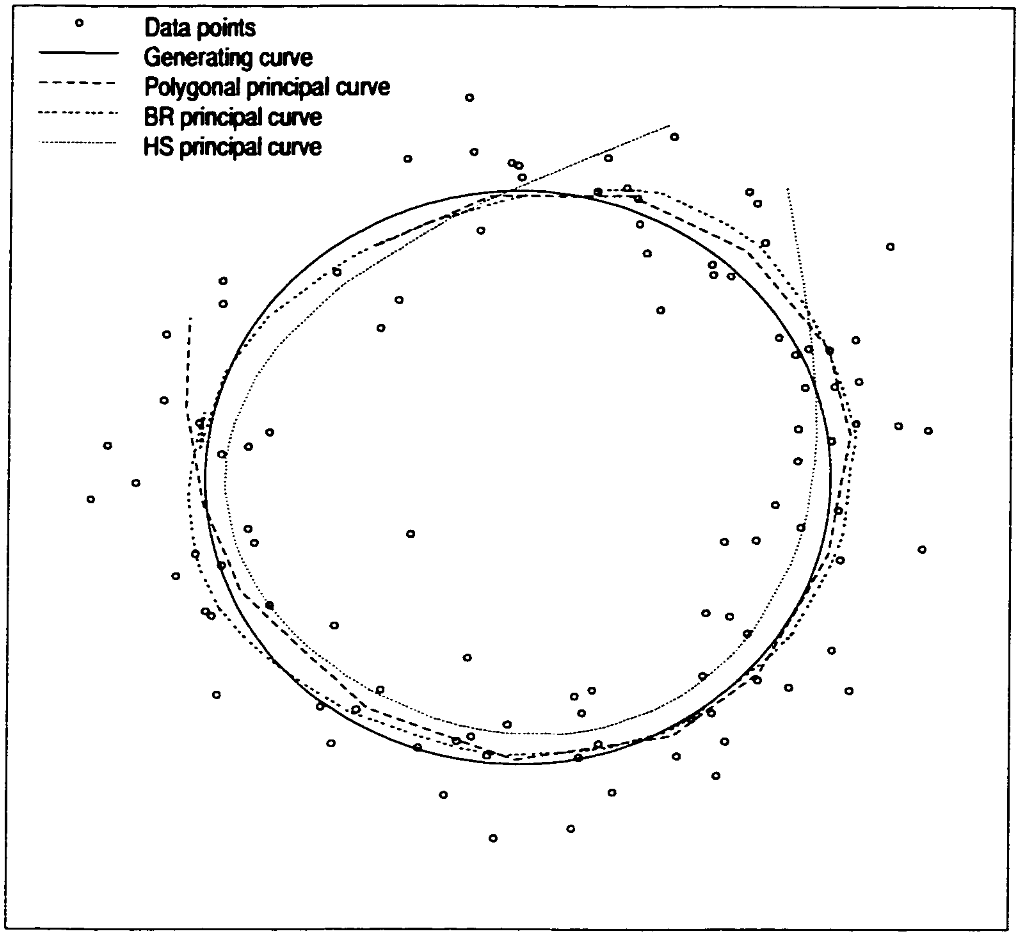


Figure 14: The circle example. The BR and the polygonal line algorithms show less bias than the HS algorithm.

of the distorted half circle.

There are two situations when we expect our algorithm to perform particularly well. If the distribution is concentrated on a curve, then according to both the HS and our definitions the principal curve is the generating curve itself. Thus, if the noise variance is small, we expect both algorithms to very closely approximate the generating curve. The data in Figure 17 was generated using the same additive Gaussian model as in Figure 14, but the noise variance was reduced to $E(e_i^2) = 0.0001$ for $i = 1,2$. In this case we found that the polygonal line algorithm outperformed both the HS and the BR algorithms.

The second case is when the sample size is large. Although the generating curve is not necessarily the principal curve of the distribution, it is natural to expect the algorithm to well approximate the generating curve as the sample size grows. Such a case is shown in Figure 18, where $n = 10000$ data points were generated (but only 2000 of these were actually plotted). Here the polygonal line algorithm approximates the generating curve with much better accuracy than the HS algorithm.

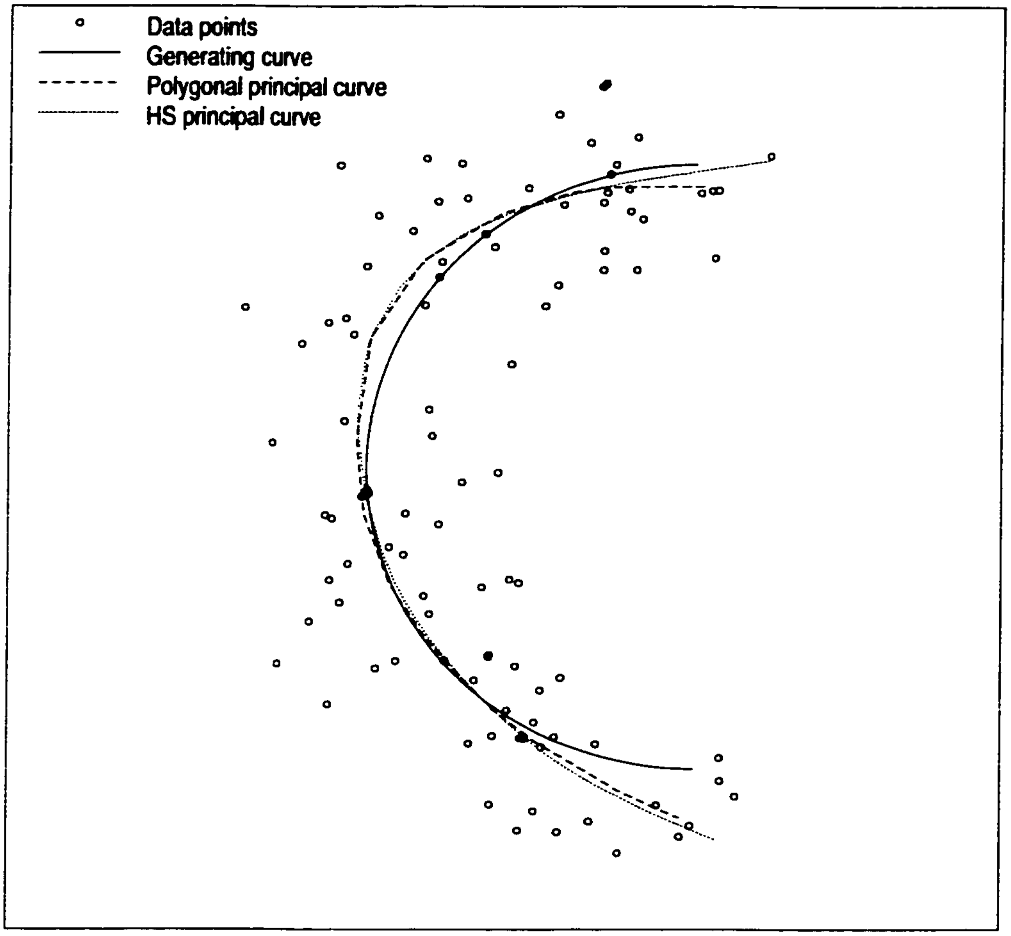
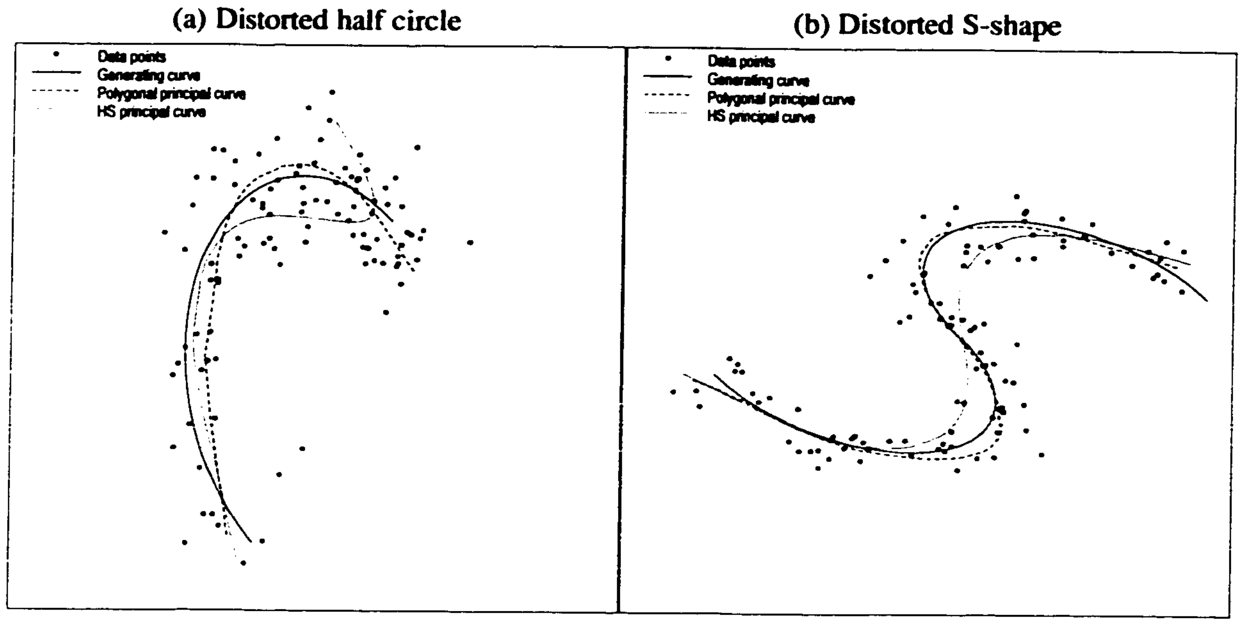


Figure 15: The half circle example. The HS and the polygonal line algorithms produce similar curves.

5.2.2 Quantitative Analysis

Although in the model (85) the generating curve is in general not the principal curve in our definition (or in the HS definition), it is of interest to numerically evaluate how well the polygonal line algorithm approximates the generating curve. In the light of the last two experiments of the previous section, we are especially interested in how the approximation improves as the sample size grows and as the noise variance decreases.

In these experiments the generating curve $\mathbf{g}(t)$ is a circle of radius $r = 1$ centered at the origin and the noise is zero mean bivariate uncorrelated Gaussian. We chose 21 different data sizes ranging from 10 to 10000, and 6 different noise standard deviations ranging from $\sigma = 0.05$ to $\sigma = 0.4$. For each particular data size and noise variance value, 100 to 1000 random data sets were generated. We run the polygonal line algorithm on each data set, and recorded several measurements in each experiment (Figure 19 shows the resulted curve in three sample runs). The measurements were then averaged over the experiments. To eliminate the distortion occurring at the endpoints, we initialized



(a) Distorted half circle

(b) Distorted S-shape

Figure 16: Transformed data sets. The polygonal line algorithm still follows fairly closely the "distorted" shapes.

the polygonal line algorithm by a closed curve, in particular, by an equilateral triangle inscribed in the generating circle.

For the measure of approximation, in each experiment we numerically evaluated the average distance defined by

$$\delta = \frac{1}{l(\mathbf{f})} \int \min_s \|\mathbf{f}(t) - \mathbf{g}(s)\| dt$$

where the polygonal line $\mathbf{f}$ is parameterized by its arc length. The measurements were then averaged over the experiments to obtain $\overline{\delta}_{n,\sigma}$ for each data size n and noise standard deviation σ. The dependence of the average distance $\overline{\delta}_{n,\sigma}$ on the data size and the noise variance is plotted on a logarithmic scale in Figure 20. The resulting curves justify our informal observation made earlier that the approximation substantially improves as the data size grows, and as the variance of the noise decreases.

To evaluate how well the distance function of the polygonal principal curve estimates the variance of the noise, we also measured the relative difference between the standard deviation of the noise σ and the measured $RMSE(\mathbf{f}) = \sqrt{\Delta_n(\mathbf{f})}$ defined as

$$\varepsilon = \frac{|\sigma - RMSE(\mathbf{f})|}{\sigma}.$$

The measurements were then averaged over the experiments to obtain $\overline{\varepsilon}_{n,\sigma}$ for each data size n and noise standard deviation σ. The dependence of the relative error $\overline{\varepsilon}_{n,\sigma}$ on the data size and the noise variance is plotted on a logarithmic scale in Figure 21. The graph indicates that, especially if the

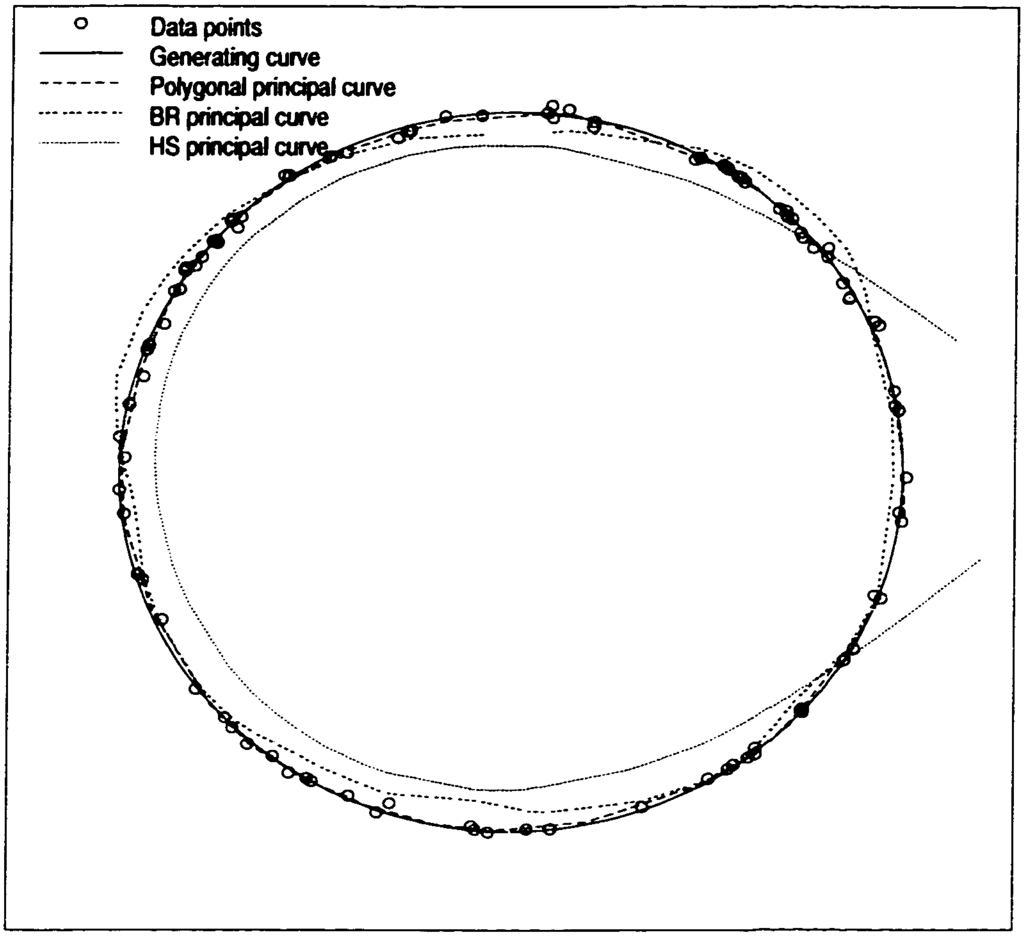


Figure 17: Small noise variance. The polygonal line algorithm follows the generating curve more closely than the HS and the BR algorithms.

standard deviation of the noise is relatively large ($\sigma \geq 0.2$), the relative error does not decrease under a certain limit as the data size grows. This suggest that the estimation exhibits an inherent bias built in the generating model (85). To support this claim, we measured the average radius of the polygonal principal curve defined by

$$\bar{r} = \frac{1}{l(\mathbf{f})} \int \|\mathbf{f}(t)\| dt,$$

where $\mathbf{f}$ is parameterized by its arc length. The measurements were then averaged over the experiments to obtain $\bar{r}_{n,\sigma}$ for each data size n and noise standard deviation σ. We also averaged the $RMSE$ values to obtain $\overline{RMSE}_{n,\sigma}$ for each data size n and noise standard deviation σ. Then we compared $\bar{r}_{n,\sigma}$ and $\overline{RMSE}_{n,\sigma}$ to the theoretical values obtained by HS,

$$r^* \approx r + \frac{\sigma^2}{2r} = 1 + \frac{\sigma^2}{2}$$

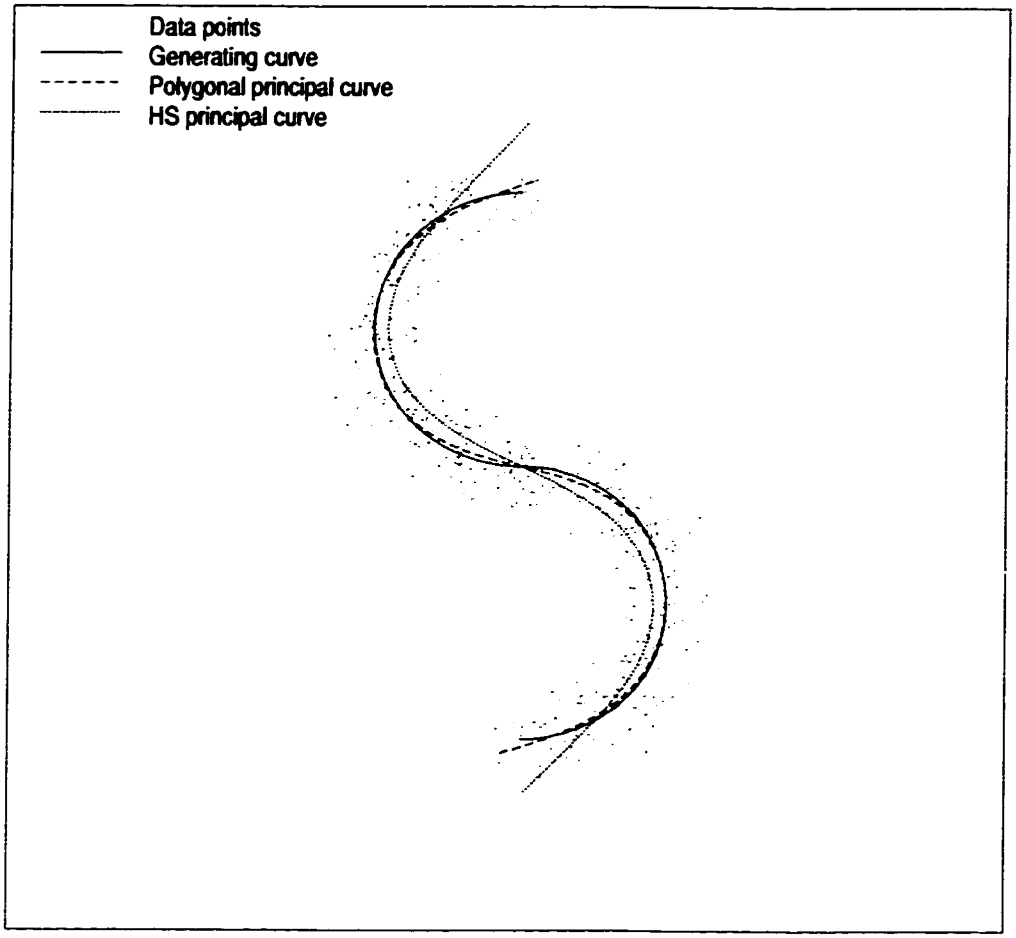


Figure 18: Large sample size. The curve produced by the polygonal line algorithm is nearly indistinguishable from the generating curve.

and

$$RMSE^* = \sqrt{\Delta(\mathbf{f}^*)} \approx \sigma\sqrt{1-\frac{\sigma^2}{4r^2}} = \sigma\sqrt{1-\frac{\sigma^2}{4}},$$

respectively. (For the definitions of r^* and $\Delta(\mathbf{f}^*)$ see (42) and (43) in Section 3.1.3). Table 2 shows the numerical results for $n = 1000$ and $n = 10000$. The measurements indicate that the average radius and $RMSE$ values measured on the polygonal principal curve are in general closer to the biased values calculated on the theoretical (HS) principal curve than to the original values of the generating curve. The model bias can also be visually detected for large sample sizes and large noise variance. In Figure 19(c), the polygonal principal curve is outside the generating curve almost everywhere.

HS and BR pointed out that in practice, the estimation bias tends to be much larger than the model bias. The fact that we could numerically detect the relatively small model bias supports our claim that the polygonal line algorithm substantially reduces the estimation bias.

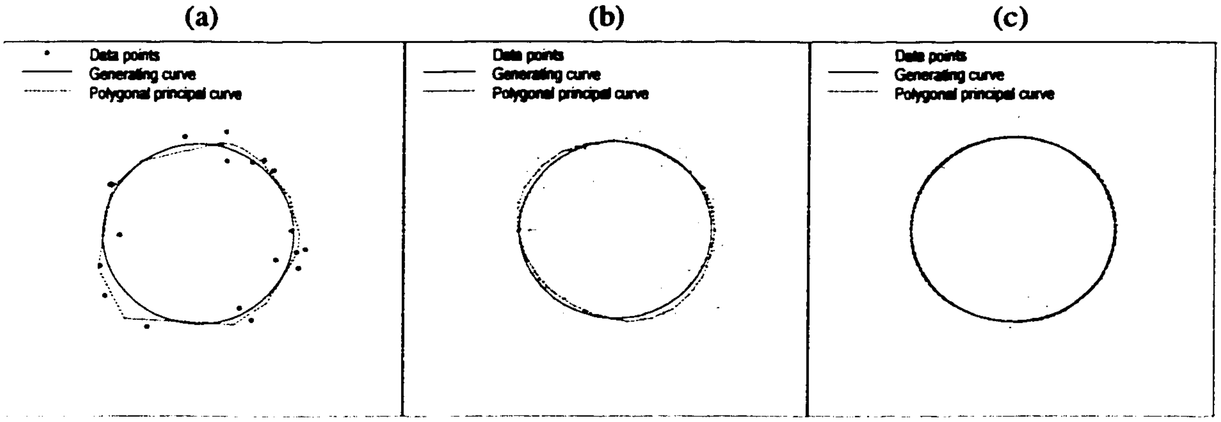


Figure 19: Sample runs for the quantitative analysis. (a) $n = 20$, $\sigma = 0.1$. (b) $n = 1000$, $\sigma = 0.3$. (c) $n = 10000$, $\sigma = 0.2$.

σ	0.05	0.1	0.15	0.2	0.3	0.4
$RMSE^*$	0.04998	0.09987	0.14958	0.199	0.29661	0.39192
$\overline{RMSE}_{1000,\sigma}$	0.04963	0.09957	0.148	0.19641	0.28966	0.37439
$\overline{RMSE}_{10000,\sigma}$	0.05003	0.0998	0.14916	0.19797	0.2922	0.378
r	1.0	1.0	1.0	1.0	1.0	1.0
r^*	1.00125	1.005	1.01125	1.02	1.045	1.08
$\bar{r}_{1000,\sigma}$	1.00135	1.00718	1.01876	1.01867	1.0411	1.08381
$\bar{r}_{10000,\sigma}$	0.99978	1.01038	1.00924	1.01386	1.03105	1.08336

Table 2: The average radius and *RMSE* values measured on the polygonal principal curve are in general closer to the biased values calculated on the theoretical (HS) principal curve than to the original values of the generating curve.

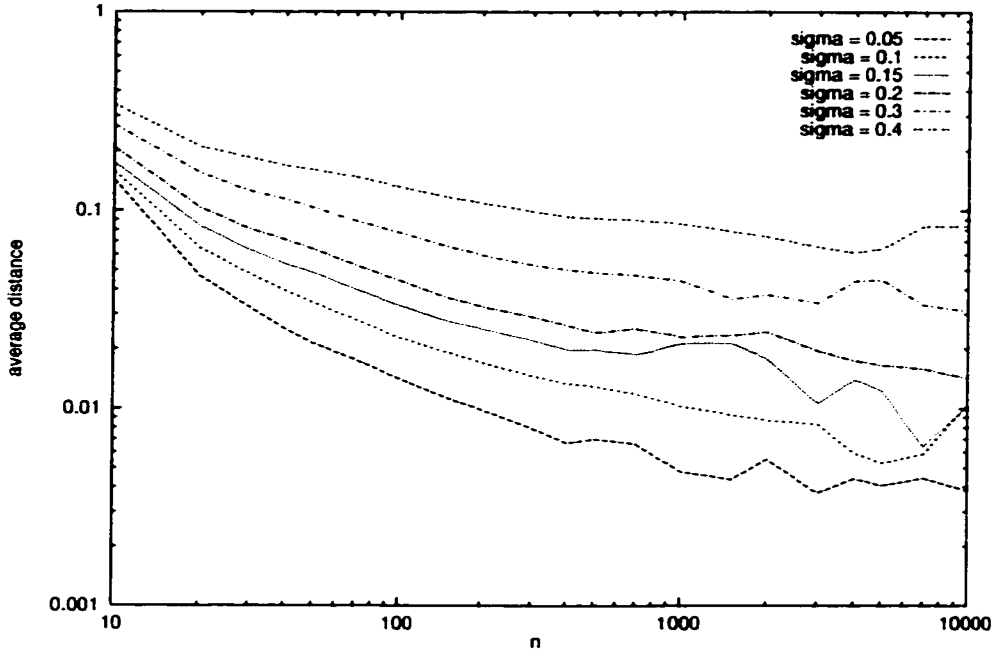


Figure 20: The average distance $\overline{\delta}_{n,\sigma}$ of the generating curve and the polygonal principal curve in terms of σ and n. The approximation improves as the data size grows, and as the variance of the noise decreases.

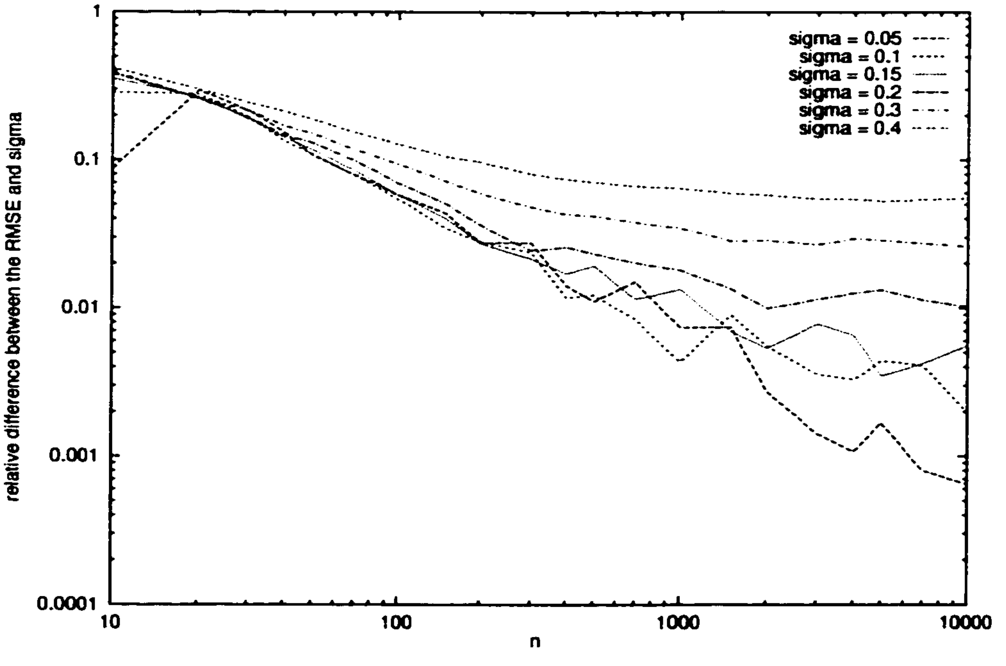


Figure 21: The relative difference $\overline{\varepsilon}_{n,\sigma}$ between the standard deviation of the noise σ and the measured RMSE.

5.2.3 Failure Modes

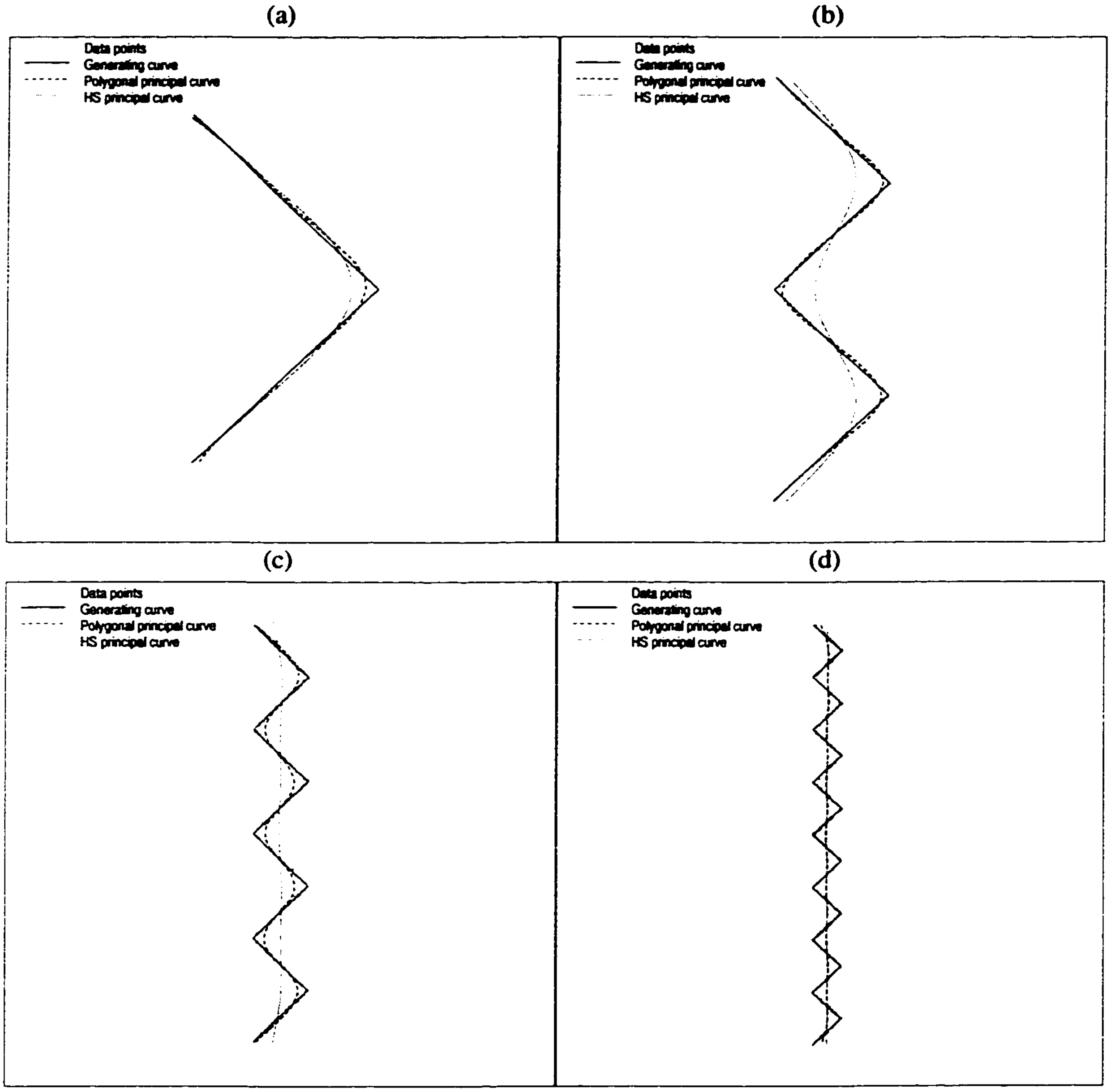


Figure 22: Abrupt changes in the direction of the generating curve. The polygonal line algorithm oversmoothes the principal curve as the number of direction changes increases.

We describe two specific situations when the polygonal line algorithm fails to recover the generating curve. In the first scenario, we use zig-zagging generating curves $\mathbf{g}_i$ for $i = 1, 2, 3, 4$ consisting of 2^i line segments of equal length, such that two consecutive segments join at a right angle (Figure 22). In these experiments, the number of the data points generated on a line segment is constant (it is set to 100), and the variance of the bivariate Gaussian noise is $l^2 \cdot 0.0005$, where l is the length of a line segment. Figure 22 shows the principal curves produced by the HS and the polygonal line algorithms in the four experiments. Although the polygonal principal curve follows the generating curve more closely than the HS principal curve in the first three experiments (Figures 22(a), (b), and (c)), the two algorithms produce equally poor results if the number of line segments exceeds a certain limit (Figure 22(d)).

Analysis of the data-dependent penalty term explains this behavior of the polygonal line algorithm. Since the relative penalty (in terms of the distance function) is proportional to the number of line segments of the *principal curve*, it relatively increases as the number of the line segments of the *generating curve* grows. To achieve the same local smoothness in the four experiments, the penalty factor should be gradually decreased as the number of line segments of the generating curve grows. Indeed, if the constant of the penalty term is reset to $\lambda' = 0.02$ in the fourth experiment, the polygonal principal curve recovers the generating curve with high accuracy (Figure 23).

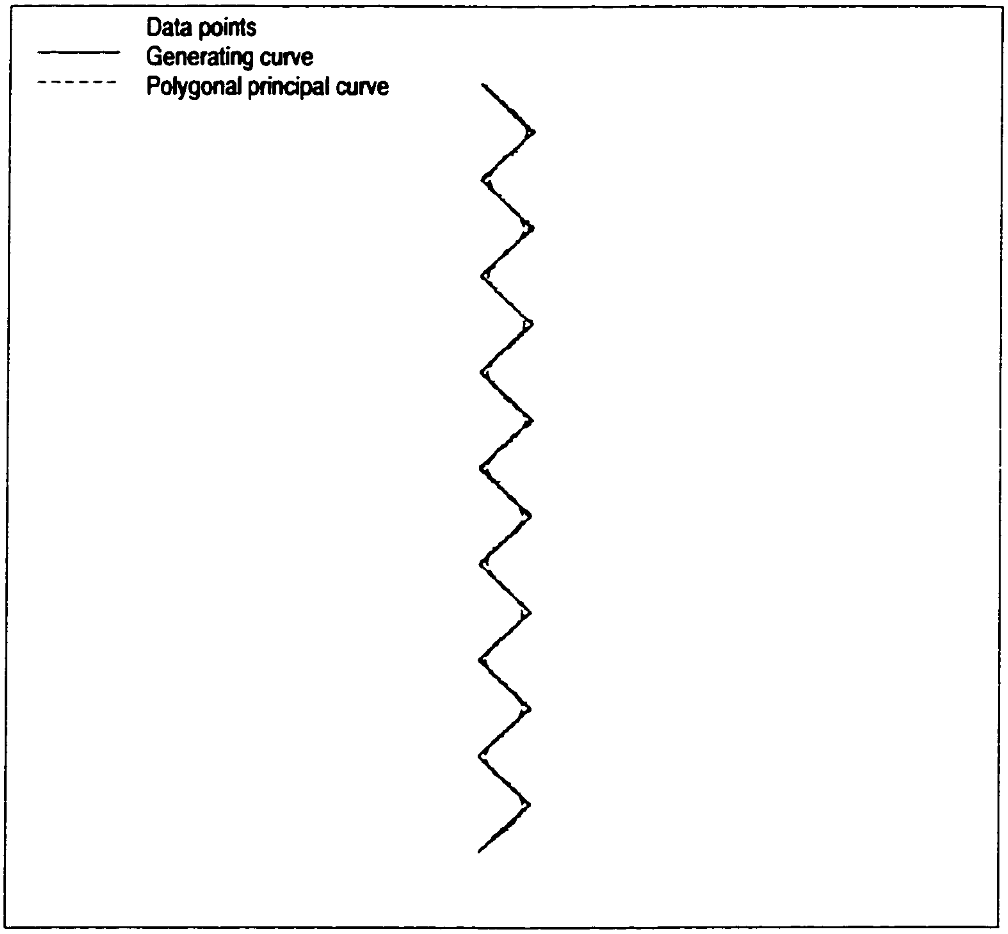


Figure 23: The polygonal principal curve follows the zig-zagging generating curve closely if the penalty coefficient is decreased.

The second scenario when the polygonal line algorithm fails to produce a meaningful result is when the generating curve is too complex so the algorithm does not find the global structure of the data. To test the gradual degradation of the algorithm, we used spiral-shaped generating curves of increasing length, i.e., we set $\mathbf{g}_i(t) = (t\sin(i\pi t), t\cos(i\pi t))$ for $t \in [0,1]$ and $i = 1,\ldots,6$. The variance of the noise was set to 0.0001, and we generated 1000 data points in each experiment. Figure 24 shows the principal curves produced by the HS and the polygonal line algorithms in four experiments ($i = 2,3,4,6$). In the first three experiments (Figures 24(a), (b), and (c)), the polygonal principal curve is almost indistinguishable from the generating curve, while the HS algorithm either oversmoothes the principal curve (Figure 24(a) and (b)), or fails to recover the shape of the generating curve (Figure 24(c)). In the fourth experiment both algorithms fail to find the shape of the generating curve (Figure 24(d)). The failure here is due to the fact that the algorithm is stuck in a local minima between the initial curve (the first principal component line) and the desired solution (the generating curve). If this is likely to occur in an application, the initialization step must be replaced by a more sophisticated routine that approximately captures the global structure of the data. Figure 25 indicates that this indeed works. Here we manually initialize both algorithms by a polygonal line with eight vertices. Using this "hint", the polygonal line algorithm produces an almost perfect solution, while the HS algorithm still cannot recover the shape of the generating curve.

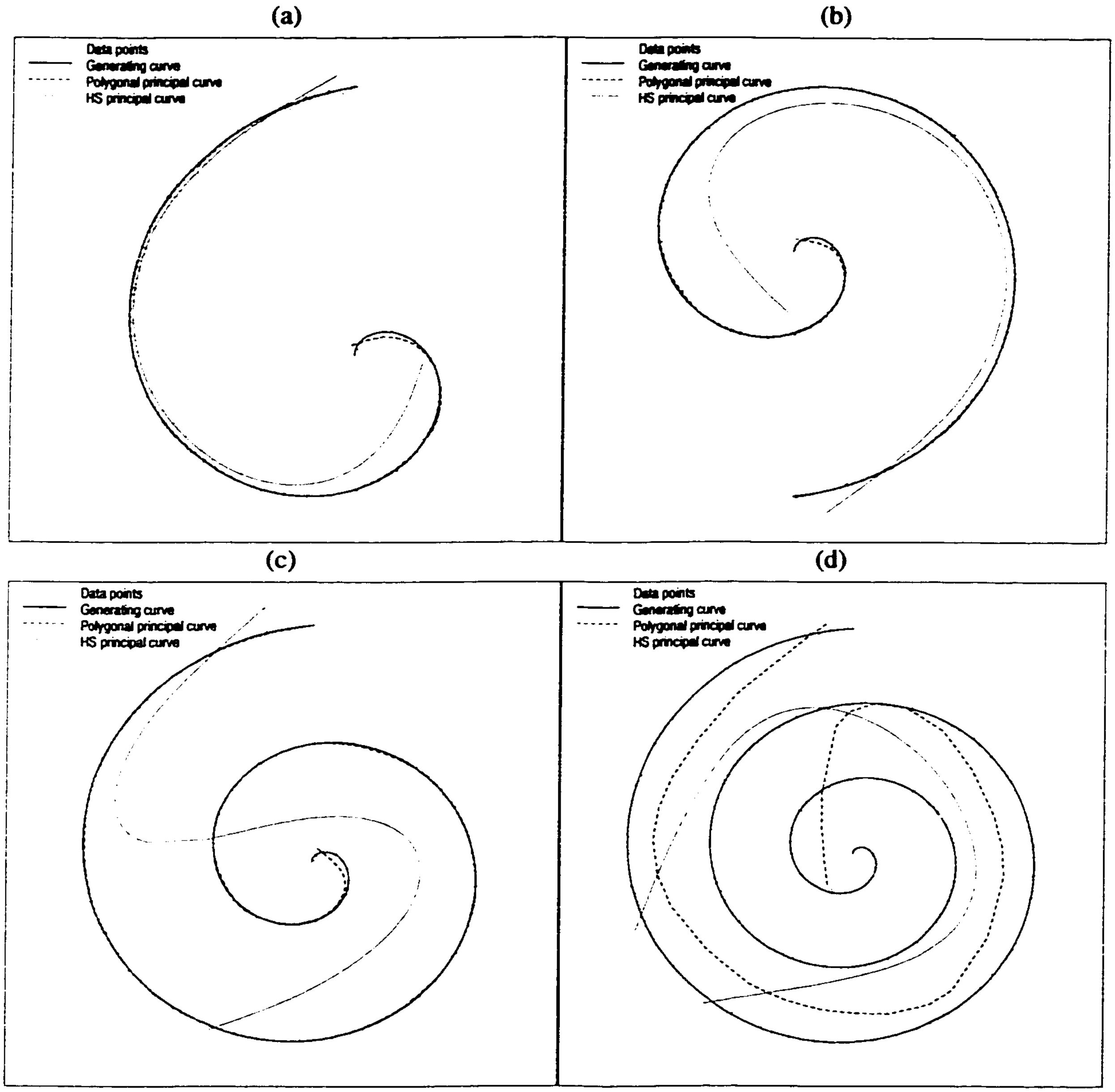


Figure 24: Spiral-shaped generating curves. The polygonal line algorithm fails to find the generating curve as the length of the spiral is increased.

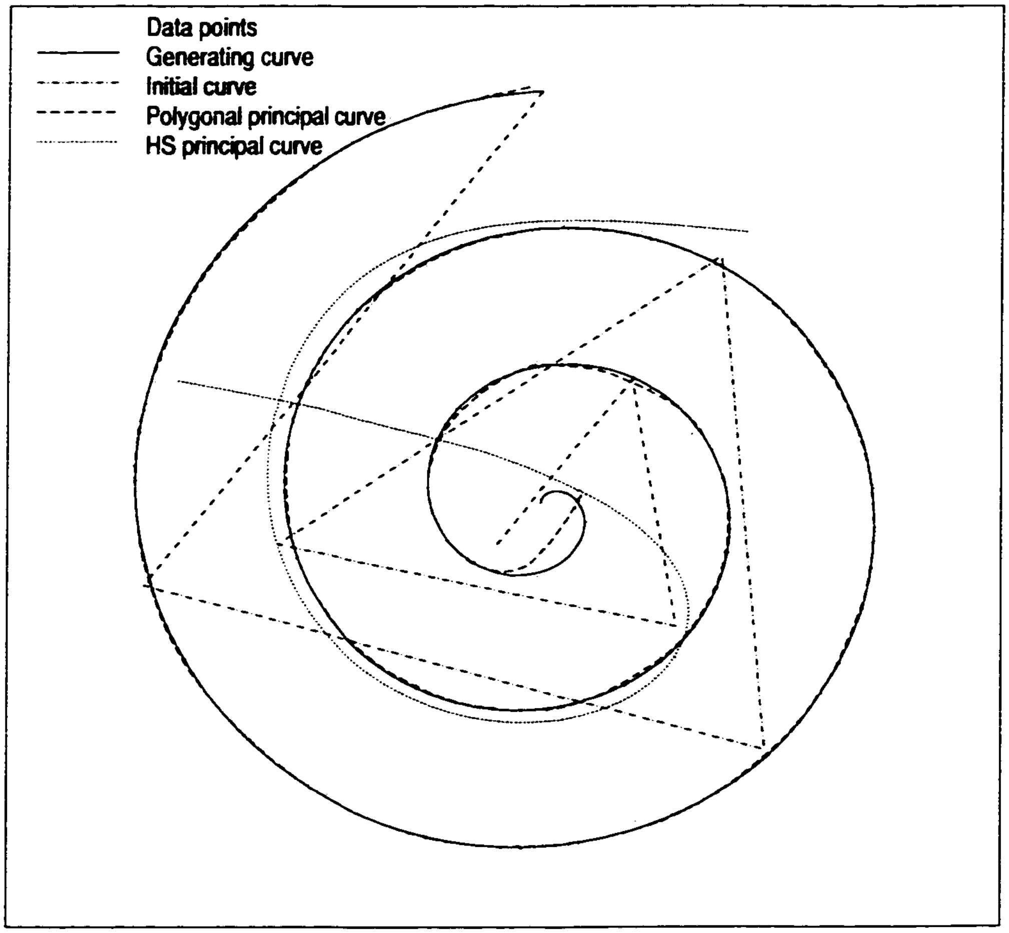


Figure 25: The performance of the polygonal line algorithm improves significantly if the global structure of the generating curve is captured in the initialization step.

Chapter 6

Application of Principal Curves to Hand-Written Character Skeletonization

The main subject of this chapter is an application of an extended version of the polygonal line algorithm to hand-written character skeletonization. Skeletonization is one of the important areas in image processing. It is most often, although not exclusively, used for images of hand-written or printed characters so we describe it here in this context. When we look at the image of a letter, we see it as a collection of curves rather than a raster of pixels. Since the earliest days of computers, it has been one of the challenges for researchers working in the area of pattern recognition to imitate this ability of the human mind [Din55, KCRU57]. Approaching skeletonization from a practical point of view, representing a character by a set of thin curves rather than by a raster of pixels is useful for reducing the storage space and processing time of the character image. It was found that this representation is particularly effective in finding relevant features of the character for optical character recognition [Deu68, AH69].

The objective of skeletonization is to find the medial axis of a character. Ideally, the medial axis is defined as a smooth curve (or set of curves) that follows the shape of a character equidistantly from its contours. In case of hand-written characters, one can also define the medial axis as the trajectory of the penstroke that created the letter. Most skeletonization algorithms approximate the medial axis by a unit-width binary image obtained from the original character by iteratively peeling its contour pixels until there remains no more removable pixel [Pav80, NS84, SA86]. The process is called the ***thinning*** of the character template, and the result is the ***skeleton*** of the character. The different thinning methods are characterized by the rules that govern the deletion of black pixels.

In this chapter we propose another approach to skeletonization. The development of the method was inspired by the apparent similarity between the definition of principal curves and the medial axis. A principal curve is a smooth curve that goes through the "middle" of a data set, whereas the medial axis is a set of smooth curves that go equidistantly from the contours of a character. Therefore, by representing the black pixels of a character by a two-dimensional data set, one can use the principal curve of the data set to approximate the medial axis of the character. Other methods using this "analogue" approach for skeletonization are described in Section 6.1. In this section we also summarize existing applications of the HS principal curve algorithm.

Since the medial axis can be a *set* of connected curves rather then only *one* curve, in Section 6.2 we extend the polygonal line algorithm to find a ***principal graph*** of a data set. The extended algorithm also contains two elements specific to the task of skeletonization, an initialization method to capture the approximate topology of the character, and a collection of restructuring operations to improve the structural quality of the skeleton produced by the initialization method. To avoid confusion, in what follows we use the term skeleton for the unit-width binary image approximating the medial axis, and we refer to the set of connected curves produced by the polygonal line algorithm as the ***skeleton graph*** of the character template.

In Section 6.3 test results of the extended polygonal line algorithm are presented. In Section 6.3.1 we apply the algorithm to isolated hand-written digits from the NIST Special Database 19 [Gro95]. The results indicate that the proposed algorithm finds a smooth medial axis of the great majority of a wide variety of character templates, and substantially improves the pixelwise skeleton obtained by traditional thinning methods. In Section 6.3.2 we present results of experiments with images of continuous handwriting. These experiments demonstrate that the skeleton graph produced by the algorithm can be used for representing hand-written text efficiently.

6.1 Related Work

6.1.1 Applications and Extensions of the HS Algorithm

Hastie [Has84] presented several experiments on real data. In the first example, computer chip waste is sampled and analyzed by two laboratories to estimate the gold content of the lot. It is in the interest of the owner of the lot to know which laboratory produces on average lower gold content estimates for a given sample. The principal curve method was used to point out that at higher levels of gold content one of the laboratories produced higher assays than the other. The difference was reversed at lower levels. [Has84] argues that these results could not have been obtained by using standard regression techniques. In another example, a principal curve was used for non-linear factor analysis on a data set of three-dimensional points representing measurements of mineral content of core samples.

The first real application of principal curves was part of the Stanford Linear Collider project [HS89]. The collider consists of a linear accelerator used to accelerate two particle beams, and two arcs that bend these beams to bring them to collision. The particle beams are guided by roughly 475 magnets that lie on a smooth curve with a circumference of about three kilometers. Measurement errors in the range of ±1 millimeters in placing the magnets resulted that the beam could not be adequately focused. Engineers realized that it was not necessary to move the magnets to the ideal curve, but rather to a curve through the existing positions that was smooth enough to allow focused bending of the beam. The HS principal curve procedure was used to find this curve.

Banfield and Raftery [BR92] described an almost fully automatic method for identifying ice floes and their outlines in satellite images. The core procedure of the method uses a closed principal curve to estimate the floe outlines. Besides eliminating the estimation bias of the HS algorithm (see Section 3.1.3), [BR92] also replaced the initialization step of the HS algorithm by a more sophisticated routine that produced a rough estimate of the floe outlines. Furthermore, [BR92] extended existing clustering methods by allowing groups of data points to be centered about principal curves rather than points or lines.

Principal curve clustering was further extended and analyzed by Stanford and Raftery [SR97]. Here, fitting a principal curve is combined with the Classification EM algorithm [CG92] to iteratively refine clusters of data centered about principal curves. The number of clusters and the smoothness parameters of the principal curves are chosen automatically by comparing approximate Bayes factors [KR95] of different models. Combining the clustering algorithm with a denoising procedure and an initialization procedure, [SR97] proposed an automatic method for extracting curvilinear features of simulated and real data.

Chang and Ghosh [CG98b, CG98a] used principal curves for nonlinear feature extraction and pattern classification. [CG98b] pointed out experimentally that a combination of the HS and BR algorithms (the BR algorithm is run after the HS algorithm) reduces the estimation bias of the HS algorithm and also decreases the variance of the BR algorithm that was demonstrated in Section 5.2. [CG98b] and [CG98a] demonstrated on several examples that the improved algorithm can be used effectively for feature extraction and classification.

Reinhard and Niranjan [RN98] applied principal curves to model the short time spectrum of speech signals. First, high-dimensional data points representing diphones (pairs of consecutive phones) are projected to a two-dimensional subspace. Each diphone is than modeled by a principal curve. In the recognition phase, test data is compared to the principal curves representing the different diphones, and classified as the diphone represented by the nearest principal curve. [RN98] demonstrated in experiments that the diphone recognition accuracy of can can be comparable to the accuracy of the state-of-the-art hidden Markov models.

6.1.2 Piecewise Linear Approach to Skeletonization

[SWP98] used the HS principal curve algorithm for character skeletonization. The initial curve is produced by a variant of the SOM algorithm where the neighborhood relationships are defined by a minimum spanning tree of the pixels of the character template. The HS algorithm is then used to fit the curve to the character template. In the expectation step a weighted kernel smoother is used which, in this case, is equivalent to the update rule of the SOM algorithm. [SWP98] demonstrated that principal curves can be successfully used for skeletonizing characters in fading or noisy texts where traditional skeletonization techniques are either inapplicable or perform poorly.

Similar skeletonization methods were proposed by Mahmoud et al. [MAG91] and Datta and Parui [DP97]. Similarly to [SWP98], [DP97] uses the SOM algorithm to optimize the positions of vertices of a piecewise linear skeleton. The algorithm follows a "bottom-up" strategy in building the skeletal structure: the approximation starts from a linear topology and later adds forks and loops to the skeleton based on local geometric patterns formed during the SOM optimization. [MAG91] proposed an algorithm to obtain piecewise linear skeletons of Arabic characters. The method is based on fuzzy clustering and the fuzzy ISODATA algorithm [BD75] that uses a similar optimization to the batch version of the SOM algorithm.

Although, similarly to the principal graph algorithm, [MAG91, DP97, SWP98] also use a piecewise linear approximation of the skeleton of the character, their approaches substantially differ from our approach in that smoothness of the skeleton is not a primary issue in these works. Although the SOM algorithm implicitly ensures smoothness of the skeleton to a certain extent, it lacks a clear and intuitive formulation of the two competing criteria, smoothness of the skeleton and closeness of the fit, which is explicitly present in our method. In this sense our algorithm complements these methods rather then competes with them. For example, the method of [SWP98] could be used as an alternative initialization step for the principal graph algorithm if the input is too noisy for our thinning-based initialization step. One could also use the restructuring operations described in [DP97] combined with the fitting-and-smoothing optimization step of the principal graph algorithm in a "bottom-up" approach of building the skeleton graph.

6.2 The Principal Graph Algorithm

In this section we describe the principal graph algorithm, an extension of the polygonal line algorithm for finding smooth skeletons of hand-written character templates. To transform binary (black-and-white) character templates into two-dimensional data sets, we place the midpoint of the bottom-most left-most pixel of the template to the center of a coordinate system. The unit length of the coordinate system is set to the width (and height) of a pixei, so the midpoint of each pixel has integer coordinates. Then we add the midpoint of each black pixel to the data set. Figure 26 illustrates the data representation model.

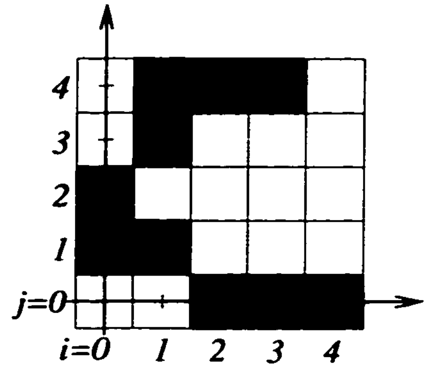


Figure 26: Representing a binary image by the integer coordinates of its black pixels. The 5 × 5 image is transformed into the set $\mathcal{X} = \left\{ \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right], \left[\begin{smallmatrix}0\\2\end{smallmatrix}\right], \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right], \left[\begin{smallmatrix}1\\3\end{smallmatrix}\right], \left[\begin{smallmatrix}1\\4\end{smallmatrix}\right], \left[\begin{smallmatrix}2\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}2\\4\end{smallmatrix}\right], \left[\begin{smallmatrix}3\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}3\\4\end{smallmatrix}\right], \left[\begin{smallmatrix}4\\0\end{smallmatrix}\right] \right\}$.

The polygonal line algorithm was tested on images of isolated handwritten digits from the NIST Special Database 19 [Gro95]. We found that the polygonal line algorithm can be used effectively to find smooth medial axes of simple digits which contain no loops or crossings of strokes. Figure 27 shows some of these results.

To find smooth skeletons of more complex characters we extend and modify the polygonal line algorithm. In Section 6.2.1 we extend the optimization and the projection steps so that in the inner loop of the polygonal line algorithm we can optimize Euclidean graphs rather than only polygonal curves. To capture the approximate topology of the character, we replace the initialization step by a more sophisticated routine based on a traditional thinning method. The new initialization procedure is described in Section 6.2.2. Since the initial graph contains enough vertices for a smooth approximation, we no longer need to use the outer loop of the polygonal line algorithm to add vertices to the graph one by one. Instead, we use the inner loop of the algorithm only twice. Between the two fitting-and-smoothing steps, we "clean" the skeleton graph from spurious branches and loops that are created by the initial thinning procedure. Section 6.2.3 describes the restructuring operations used in this step. The flow chart of the extended polygonal line algorithm is given in Figure 28. Figure 29 illustrates the evolution of the skeleton graph on an example.

6.2.1 Principal Graphs

In this section we introduce the notion of a *Euclidean graph* as a natural extension of polygonal curves. The principal curve algorithm is then extended to optimize a Euclidean graph rather than a single curve. We introduce new vertex types to accommodate junction points of a graph. The new vertex types are tailored to the task of finding a smooth skeleton of a character template. In a

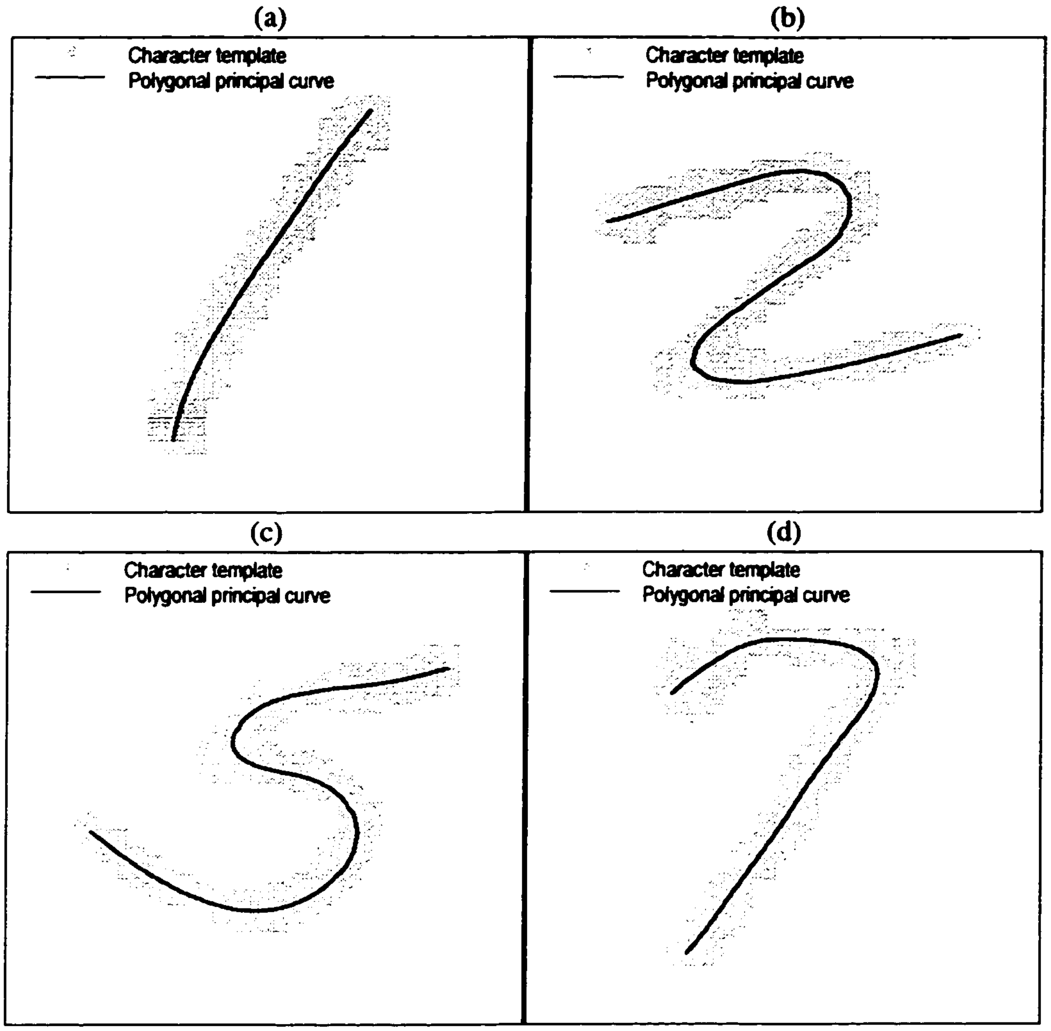


Figure 27: The polygonal line algorithm can be used effectively to find smooth medial axes of simple digits which contain no loops or crossings of strokes.

different application, other vertex types can be introduced along the same lines.

Once the local distance function and the local penalty term are formulated for the new vertex types, the vertex optimization step (Section 5.1.5) is completely defined for Euclidean graphs. The projection step (Section 5.1.4) can be used without modification. As another indication of the robustness of the polygonal line algorithm, the penalty factor λ, which was developed using the data generating model (85), remains as defined in (73).

Euclidean Graphs

A Euclidean graph $G_{\mathcal{V}\mathcal{S}}$ in the d-dimensional Euclidean space is defined by two sets, $\mathcal{V}$ and $\mathcal{S}$, where $\mathcal{V} = \{\mathbf{v}_1,\ldots,\mathbf{v}_m\} \subset \mathbb{R}^d$ is a set of *vertices*, and $\mathcal{S} = \{(\mathbf{v}_{i_1},\mathbf{v}_{j_1}),\ldots,(\mathbf{v}_{i_k},\mathbf{v}_{j_k})\} = \{\mathbf{s}_{i_1j_1},\ldots,\mathbf{s}_{i_k,j_k}\}$,

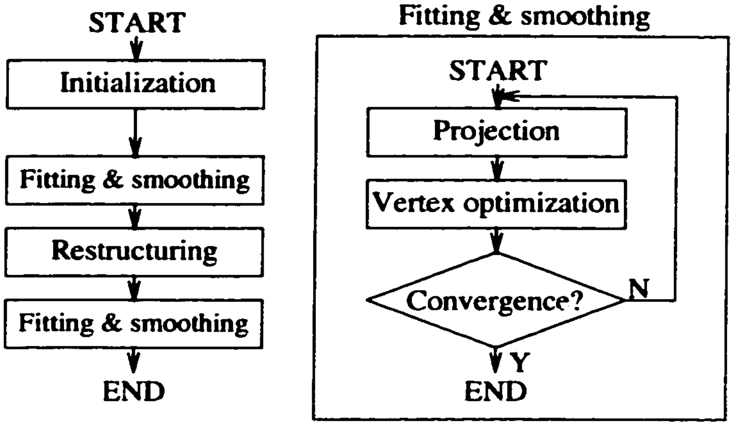


Figure 28: The flow chart of the extended polygonal line algorithm.

$1 \leq i_1, j_1, \ldots, i_k, j_k \leq m$ is a set of *edges*, such that $\mathbf{s}_{ij}$ is a line segment that connects $\mathbf{v}_i$ and $\mathbf{v}_j$. We say that two vertices are *adjacent* or *neighbors* if there is an edge connecting them. The edge $\mathbf{s}_{ij} = (\mathbf{v}_i, \mathbf{v}_j)$ is said to be *incident* with the vertices $\mathbf{v}_i$ and $\mathbf{v}_j$. The vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ are also called the *endpoints* of $\mathbf{s}_{ij}$. The *degree* of a vertex is the number of edges incident with it.

Let $\mathbf{x} \in \mathbb{R}^d$ be an arbitrary data point. The squared Euclidean distance between $\mathbf{x}$ and a graph $G_{\mathcal{V},\mathcal{S}}$ is the squared distance between $\mathbf{x}$ and the nearest edge of $G_{\mathcal{V},\mathcal{S}}$ to $\mathbf{x}$, i.e.,

$$\Delta(\mathbf{x}, G_{\mathcal{V},\mathcal{S}}) = \min_{\mathbf{s}\in\mathcal{S}} \Delta(\mathbf{x}, \mathbf{s}).$$

Then, given a data set $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^d$, the empirical distance function of $G_{\mathcal{V},\mathcal{S}}$ is defined as usual,

$$\Delta_n(G_{\mathcal{V},\mathcal{S}}) = \frac{1}{n}\sum_{i=1}^{n} \Delta(\mathbf{x}_i, G_{\mathcal{V},\mathcal{S}}).$$

Note that a polygonal curve $\mathbf{f}$ is a special Euclidean graph with the property that the vertices of $\mathbf{f}$, $\mathbf{v}_1, \ldots, \mathbf{v}_m$, can be indexed so that $\mathbf{s}_{ij} = (\mathbf{v}_i, \mathbf{v}_j)$ is an edge if and only if $j = i + 1$.

In what follows we will use the term *graph* as an abbreviation for Euclidean graph. We will also omit the indices of $G_{\mathcal{V},\mathcal{S}}$ if it does not cause confusion.

New Vertex Types

The definition of the local distance function (74) in Section 5.1.5 differentiates between vertices at the end and in the middle of the polygonal curve. We call these vertices *end-vertices* and *line-vertices*, respectively. In this section we introduce new vertex types to accommodate intersecting curves that occur in handwritten characters. Vertices of different types are characterized by their degrees and the types of the local curvature penalty imposed at them (see Table 3).

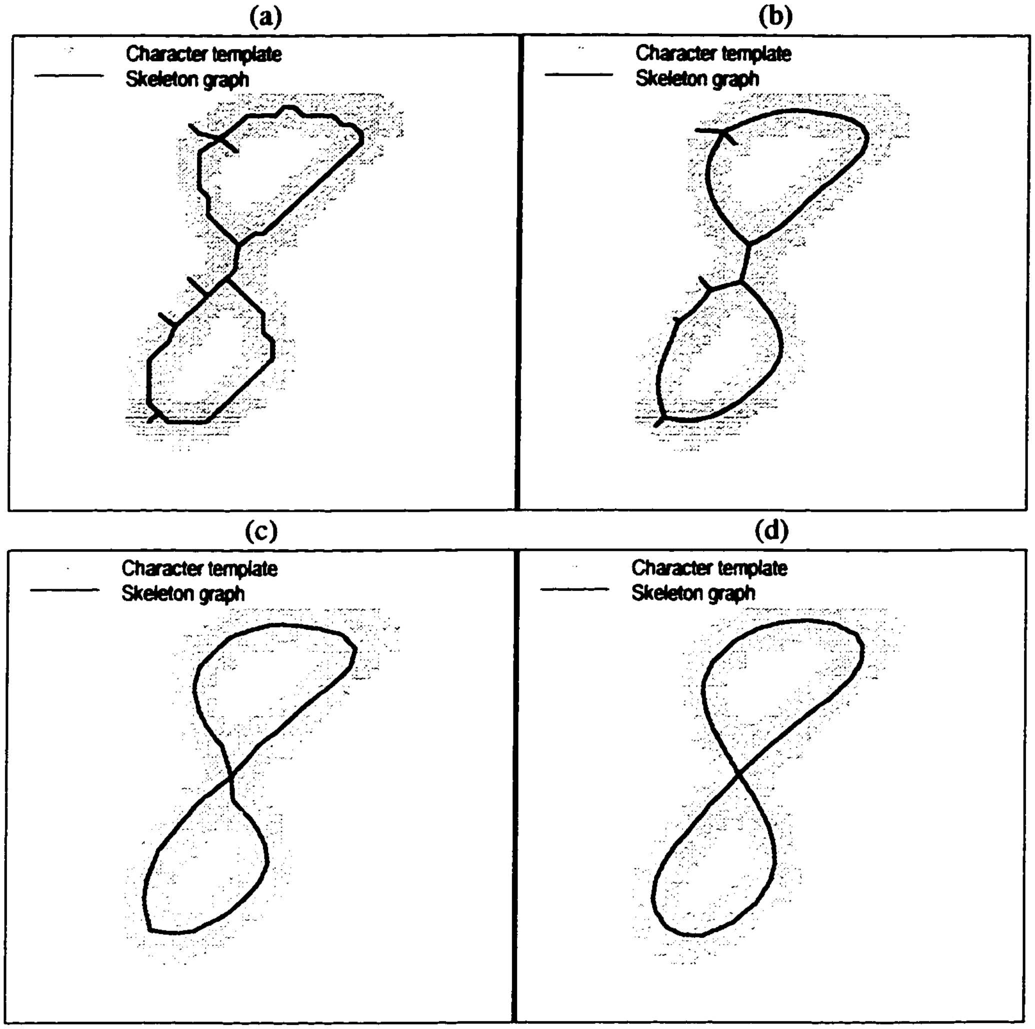


Figure 29: Evolution of the skeleton graph. The skeleton graph produced by the extended polygonal line algorithm (a) after the initialization step, (b) after the first fitting-and-smoothing step, (c) after the restructuring step, and (d) after the second fitting-and-smoothing step (the output of the algorithm).

The only vertex type of degree one is the end-vertex. Here we penalize the squared length of the incident edge as defined in (75). If two edges are joined by a vertex, the vertex is either a line-vertex or a *corner-vertex*. The angle at a line-vertex is penalized as in (75), while at a corner vertex we penalize the angle for its deviation from right angle. We introduce three different vertex types of degree three. At a *star3-vertex*, no penalty is imposed. At a *T-vertex*, we penalize one of the three angles for its deviation from straight angle. The remaining two angles are penalized for their deviations from right angle. At a *Y-vertex*, two of the possible angles are penalized for their deviations from straight angle. We use only two of the several possible configurations at a vertex of degree four. At a *star4-vertex* no penalty is imposed, while at an *X-vertex* we penalize sharp angles on the two crossing curves. Vertices of degree three or more are called *junction-vertices*.

In principle, several other types of vertices can be considered. However, in practice we found that these types are sufficient to represent hand-written characters from the Latin alphabet and of the ten digits. Vertices at the end and in the middle of a curve are represented by end-vertices and line-vertices, respectively. Two curves can be joined at their endpoints by a corner-vertex (Figure 30(a)). The role of a Y-vertex is to "merge" two smooth curves into one (Figure 30(b)). A T-vertex is used to join the end of a curve to the middle of another curve (Figure 30(c)). An X-vertex represents the crossing point of two smooth curves (Figure 30(d)). Star3 and star4-vertices are used in the first fitting-and-smoothing step, before we make the decision on the penalty configuration at a particular junction-vertex.

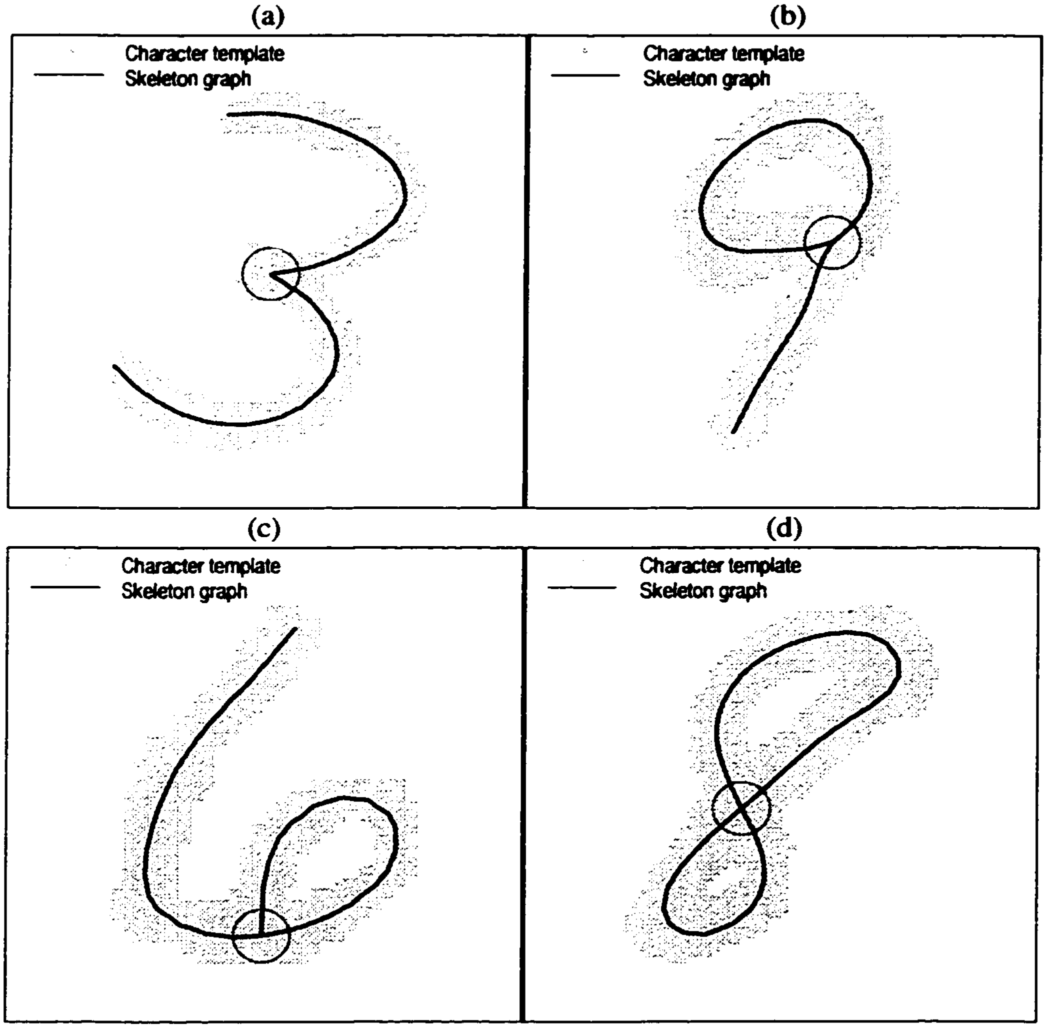


Figure 30: Roles of vertices of different types. (a) A corner-vertex joins two curves at their endpoints. (b) A Y-vertex merges two smooth curves into one. (c) A T-vertex joins the end of a curve to the middle of another curve. (d) An X-vertex represents the crossing point of two smooth curves.

The Local Distance Function

Since the edges can no longer be naturally ordered as in the case of curves, we revise our notation used in Chapter 5 in the formal definition of the formulas of the local distance function and the local penalty. Let $\mathbf{v}_1, \ldots \mathbf{v}_m$ denote the vertices of the graph and let $\mathbf{s}_{ij}$ denote the edge that connects vertices $\mathbf{v}_i$ and $\mathbf{v}_j$. Let V_i and S_{ij} be the nearest neighbor sets of vertex $\mathbf{v}_i$ and edge $\mathbf{s}_{ij}$, respectively, as defined in Section 5.1.4, let $\mathbf{s}'_{ij}$ be the line obtained by the infinite extension of the line segment $\mathbf{s}_{ij}$, and let

$$\begin{aligned}\sigma(\mathbf{s}_{ij}) &= \sum_{\mathbf{x}\in S_{ij}} \Delta(\mathbf{x}, \mathbf{s}'_{ij}),\\ \nu(\mathbf{v}_i) &= \sum_{\mathbf{x}\in V_i} \Delta(\mathbf{x}, \mathbf{v}_i).\end{aligned}$$

(Note that the new notation is a simple generalization of the notation used in Section 5.1.5 as $\nu(\mathbf{v}_i)$ is the same as before, $\sigma_+(\mathbf{v}_i) = \sigma(\mathbf{s}_{i,i+1})$, and $\sigma_-(\mathbf{v}_i) = \sigma(\mathbf{s}_{i,i-1})$.) Then the local distance function of $\mathbf{v}_i$ is defined by

$$\Delta_n(\mathbf{v}_i) = \frac{1}{n}\left(\nu(\mathbf{v}_i) + \sum_{j=1}^{\phi_i} \sigma(\mathbf{s}_{i,i_j})\right) \tag{86}$$

where ϕ_i is the degree of $\mathbf{v}_i$ $(1 \le \phi_i \le 4)$, and $i_1, \ldots, i_{\phi_i}$ are the indices of the adjacent vertices to $\mathbf{v}_i$. Note that (86) is an extension of (74) where $\Delta_n(\mathbf{v}_i)$ is defined for $\phi_i = 1, 2$.

The Local Penalty

For the definition of the local penalty, let $\pi_{ij\ell} = r^2(1 + \cos\gamma_{ij\ell})$ be the angle penalty at $\mathbf{v}_j$ where $\gamma_{ij\ell}$ denotes the angle of line segments $\mathbf{s}_{ji}$ and $\mathbf{s}_{j\ell}$, and let $\mu_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|^2$ be the length penalty at edge $\mathbf{s}_{ij}$. At corner and T-vertices we introduce $\omega_{ij\ell} = 2r^2\cos^2\gamma_{ij\ell}$ to penalize the deviation of $\gamma_{ij\ell}$ from right angle. The penalty $P_{\mathbf{v}}(\mathbf{v}_i)$ at vertex $\mathbf{v}_i$ is defined in Table 3 for the different vertex types. (Note that for end and line-vertices, $P_{\mathbf{v}}(\mathbf{v}_i)$ remains as defined in (72).) When the vertex $\mathbf{v}_i$ is moved, only angles at $\mathbf{v}_i$ and at neighbors of $\mathbf{v}_i$ can change. Therefore, the total penalty at $\mathbf{v}_i$ is defined as

$$P(\mathbf{v}_i) = P_{\mathbf{v}}(\mathbf{v}_i) + \sum_{j=1}^{\phi_i} P_{\mathbf{v}}(\mathbf{v}_{i_j}) \tag{87}$$

where ϕ_i is the degree of $\mathbf{v}_i$ $(1 \le \phi_i \le 4)$, and $i_1, \ldots, i_{\phi_i}$ are the indices of the adjacent vertices to $\mathbf{v}_i$. Note that (87) is an extension of (75) where $P(\mathbf{v}_i)$ is defined for line and end-vertices. Also note that the definition of the global penalized distance function (68), and hence the discussion in Section 5.1.6, remains valid if $\Delta'_n(\mathbf{f})$ is redefined for a graph $G_{\mathcal{V},\mathcal{S}}$ as

$$\Delta'_n(G_{\mathcal{V},\mathcal{S}}) = \sum_{\mathbf{v}\in\mathcal{V}} \nu(\mathbf{v}) + \sum_{\mathbf{s}\in\mathcal{S}} \sigma(\mathbf{s}).$$

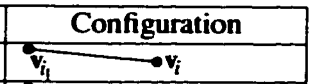
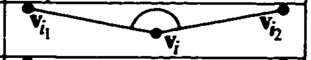
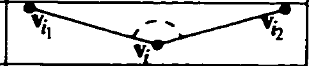
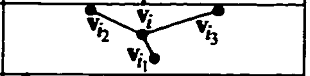
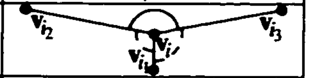
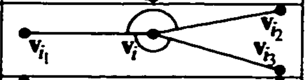
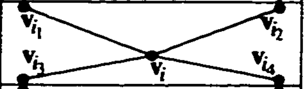
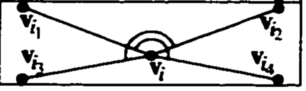
Type of $\mathbf{v}_i$	$\phi(\mathbf{v}_i)$	Penalty at $\mathbf{v}_i$	Configuration
end	1	$P_\mathbf{v}(\mathbf{v}_i) = \mu_{i,i_1}$	
line	2	$P_\mathbf{v}(\mathbf{v}_i) = \pi_{i_1,i,i_2}$	
corner	2	$P_\mathbf{v}(\mathbf{v}_i) = \omega_{i_1,i,i_2}$	
star3	3	$P_\mathbf{v}(\mathbf{v}_i) = 0$	 $\mathbf{v}_{i_2}$ $\mathbf{v}_i$ $\mathbf{v}_{i_3}$ $\mathbf{v}_{i_1}$
T	3	$P_\mathbf{v}(\mathbf{v}_i) = \pi_{i_2,i,i_3} + \omega_{i_1,i,i_2} + \omega_{i_1,i,i_3}$	 $\mathbf{v}_{i_2}$ $\mathbf{v}_i$ $\mathbf{v}_{i_1}$ $\mathbf{v}_{i_3}$
Y	3	$P_\mathbf{v}(\mathbf{v}_i) = \pi_{i_1,i,i_2} + \pi_{i_1,i,i_3}$	 $\mathbf{v}_{i_1}$ $\mathbf{v}_i$ $\mathbf{v}_{i_2}$ $\mathbf{v}_{i_3}$
star4	4	$P_\mathbf{v}(\mathbf{v}_i) = 0$	 $\mathbf{v}_{i_1}$ $\mathbf{v}_{i_2}$ $\mathbf{v}_{i_3}$ $\mathbf{v}_i$ $\mathbf{v}_{i_4}$
X	4	$P_\mathbf{v}(\mathbf{v}_i) = \pi_{i_1,i,i_4} + \pi_{i_2,i,i_3}$	 $\mathbf{v}_{i_1}$ $\mathbf{v}_{i_2}$ $\mathbf{v}_{i_3}$ $\mathbf{v}_i$ $\mathbf{v}_{i_4}$

Table 3: Vertex types and their attributes. The third column defines the penalties applied at each vertex type. The arcs in the fourth column indicate the penalized angles. The dashed arc indicates that the angle is penalized for its deviation from right angle (rather than for its deviation from from straight angle).

Degrading Vertices

Most of the reconstructing operations described in Section 6.2.3 proceed by deleting noisy edges and vertices from the skeleton graph. When an edge is deleted from the graph, the degrees of the two incident vertices decrease by one. Since there exist more than one vertex types for a given degree, the new types of the degraded vertices must be explicitly specified by degradation rules. When an edge incident to an end-vertex is deleted, we delete the vertex to avoid singular points in the skeleton graph. Line and corner-vertices are degraded to end-vertices, while star4-vertices are degraded to star3-vertices. Any vertex of degree three is degraded to a line-vertex if the remaining angle was penalized for its deviation from straight angle before the degradation, or if the angle is larger than 100 degrees. Otherwise, it is degraded to a corner-vertex. An X-vertex is degraded to a T-vertex if both of the two unpenalized angles are between 80 and 100 degrees, otherwise it is degraded to a Y-vertex. The explicit degradation rules are given in Table 4.

6.2.2 The Initialization Step

The most important requirement for the initial graph is that it approximately capture the topology of the original character template. We use a traditional connectedness-preserving thinning technique that works well for moderately noisy images. If the task is to recover characters from noisy or faded images, this initialization procedure can be replaced by a more sophisticated routine (e.g., the method based on the minimum spanning tree algorithm presented in [SWP98]) without modifying

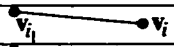
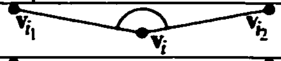
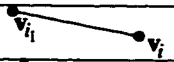
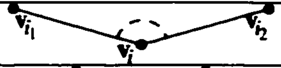
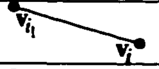
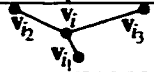
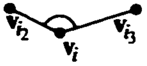
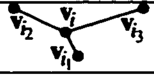
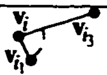
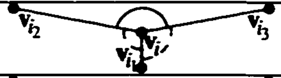
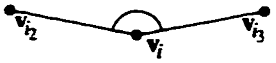
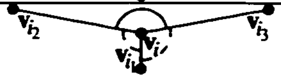
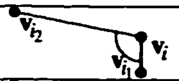
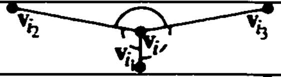
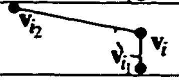
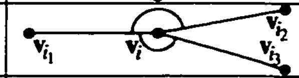
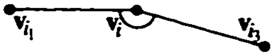
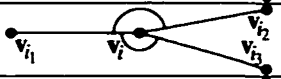
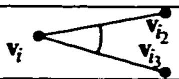
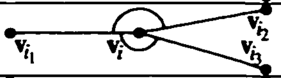
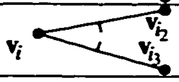
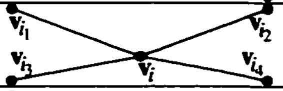
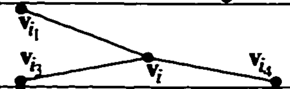
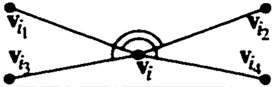
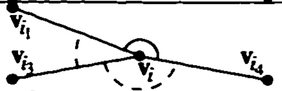
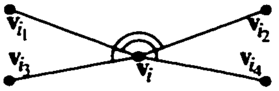
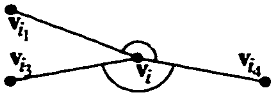
Type (before)	Configuration (before)	Deleted edge	Type (after)	Configuration (after)	Conditions
end		s_{i,i_1}	*deleted*	–	–
line		s_{i,i_2}	end		–
corner		s_{i,i_2}	end		–
star3		s_{i,i_1}	line		$\gamma_{i_2,i,i_3} > 100^\circ$
star3		s_{i,i_2}	corner		$\gamma_{i_1,i,i_3} \leq 100^\circ$
T		s_{i,i_1}	line		–
T		s_{i,i_3}	line		$\gamma_{i_1,i,i_2} > 100^\circ$
T		s_{i,i_3}	corner		$\gamma_{i_1,i,i_2} \leq 100^\circ$
Y		s_{i,i_2}	line		–
Y		s_{i,i_1}	line		$\gamma_{i_2,i,i_3} > 100^\circ$
Y		s_{i,i_1}	corner		$\gamma_{i_2,i,i_3} \leq 100^\circ$
star4		s_{i,i_2}	star3		–
X		s_{i,i_2}	T		$80^\circ \leq \gamma_{i_1,i,i_3} \leq 100^\circ$, $80^\circ \leq \gamma_{i_3,i,i_4} \leq 100^\circ$
X		s_{i,i_2}	Y		not as above, $\gamma_{i_1,i,i_4} > \gamma_{i_1,i,i_3}$, $\gamma_{i_3,i,i_4} > \gamma_{i_1,i,i_3}$

Table 4: Vertex degradation rules.

other modules of the algorithm.

We selected the particular thinning algorithm based on a survey [LLS93] which used several criteria to systematically compare twenty skeletonization algorithms. From among the algorithms that preserve connectedness, we chose the Suzuki-Abe algorithm [SA86] due to its high speed an simplicity. Other properties, such as reconstructability, quality of the skeleton (spurious branches, elongation or shrinkage at the end points), or similarity to a reference skeleton, were less important at this initial phase. Some of the imperfections are corrected by the fitting-and-smoothing operation, while others are treated in the restructuring step. The Suzuki-Abe algorithm starts by computing and storing the distance of each black pixel from the nearest white pixel (distance transformation). In the second step, layers of border pixels are iteratively deleted until pixels with locally maximal distance values are reached. Finally, some of the remaining pixels are deleted so that connectedness is preserved and the skeleton is of width one.

After thinning the template, an initial skeleton graph is computed (Figure 31). In general, midpoints of pixels of the skeleton are used as vertices of the graph, and two vertices are connected by an edge if the corresponding pixels are eight-neighbors, i.e., if they have at least one common corner. To avoid short circles and crossing edges near junctions of the skeleton, neighboring junction pixels (pixels having more than two eight-neighbors) are recursively placed into pixel-sets. For such a set, only one vertex is created in the center of gravity of the pixels' midpoints. This junction-vertex is then connected to vertices representing pixels neighboring to any of the junction pixels in the set. In this initial phase, only end, line, star3, and star4-vertices are used depending on whether the degree of the vertex is one, two, three, or four, respectively. In the rare case when a vertex representing a set of junction pixels has more than four neighbors, the neighbors are split into two or more sets of two or three vertices. Each neighbor in a set is then connected to a mutual junction-vertex, and the created junction-vertices are connected to each other. The circled vertices in Figures 31(c) and (d) demonstrate this case.

6.2.3 The Restructuring Step

The restructuring step complements the two fitting-and-smoothing steps. In the fitting-and-smoothing step we relocate vertices and edges of the skeleton graph based on their positions relative to the template, but we do not modify the skeleton graph in a graph theoretical sense. In the restructuring step we use geometric properties of the skeleton graph to modify the configuration of vertices and edges. We do not explicitly use the template, and we do not move vertices and edges of the skeleton graph in this step.

The double purpose of the restructuring step is to eliminate or rectify imperfections of the initial skeleton graph, and to simplify the skeletal description of the template. Below we define operations that can be used to modify the configuration of the skeleton graph. Since the types of the imperfections depend on properties of both the input data and the initialization method, one should carefully select the particular operations and set their parameters according to the specific application. At the description of the operations, we give approximate values for each parameter based on our experiments with a wide variety of real data. Specific settings will be given in Section 6.3 where we present the results of two particular experiments.

For the formal description of the restructuring operations, we define some simple concepts. We call a list of vertices $p_{i_1,\dots,i_\ell} = (\mathbf{v}_{i_1},\dots,\mathbf{v}_{i_\ell}), \ell > 1$ a *path* if each pair of consecutive vertices $(\mathbf{v}_{i_j},\mathbf{v}_{i_{j+1}}), j = 1,\dots,\ell-1$ is connected by an edge. A *loop* is a path $p_{i_1,\dots,i_\ell}$ such that $i_1 = i_\ell$ and none of the inner vertices $\mathbf{v}_{i_2},\dots,\mathbf{v}_{i_{\ell-1}}$ are equal to each other or to $\mathbf{v}_{i_1}$. The *length* of a path is

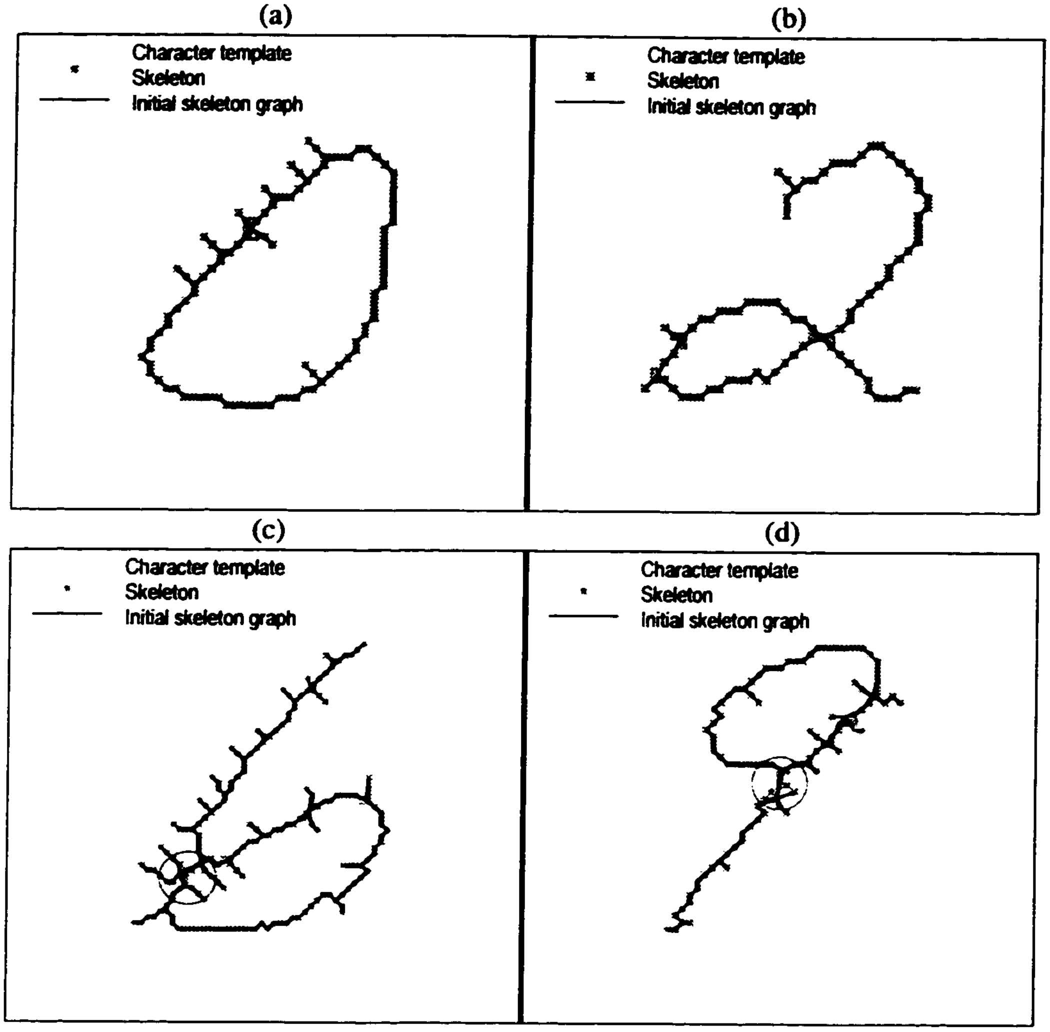


Figure 31: Examples of transforming the skeleton into an initial skeleton graph.

defined by

$$l(p_{i_1,\ldots,i_\ell}) = \sum_{j=1}^{\ell-1} l(\mathbf{s}_{i_j,i_{j+1}}) = \sum_{j=1}^{\ell-1} \|\mathbf{v}_{i_{j+1}} - \mathbf{v}_{i_j}\|.$$

A path $p_{i_1,\ldots,i_\ell}$ is *simple* if its endpoints $\mathbf{v}_{i_1}$ and $\mathbf{v}_{i_\ell}$ are not line-vertices, while all its inner vertices $\mathbf{v}_{i_2},\ldots,\mathbf{v}_{i_{\ell-1}}$ are line-vertices. A simple path is called a *branch* if at least one of its endpoints is an end-vertex. When we *delete* a simple path $p_{i_1,\ldots,i_\ell}$, we remove all inner vertices $\mathbf{v}_{i_2},\ldots,\mathbf{v}_{i_{\ell-1}}$ and all edges $\mathbf{s}_{i_1,i_2},\ldots,\mathbf{s}_{i_{\ell-1},i_\ell}$. Endpoints of the path $\mathbf{v}_{i_1}$ and $\mathbf{v}_{i_\ell}$ are degraded as specified by Table 4. Figure 32 illustrates these concepts.

Most of the reconstructing operations simplify the skeleton graph by eliminating certain simple paths that are shorter then a threshold. To achieve scale and resolution independence, we use the *thickness* of the character as the yardstick in length measurements. We estimate the thickness of the

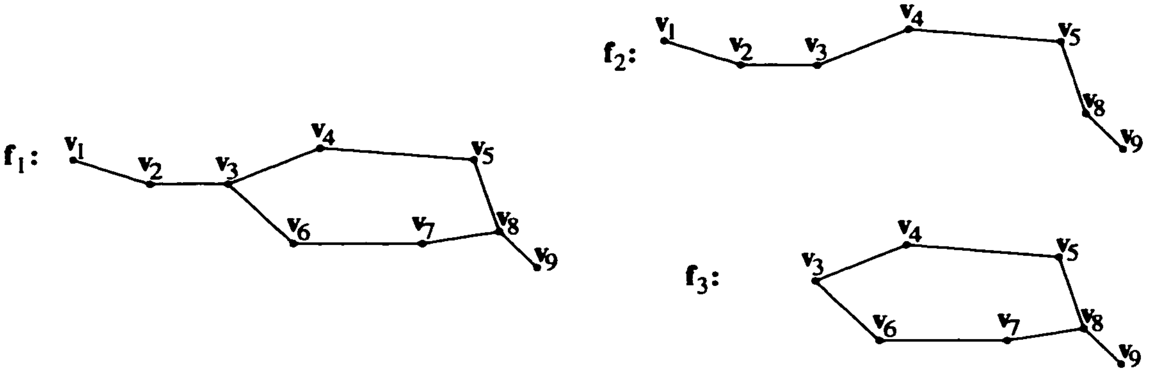


Figure 32: Paths, loops, simple paths, branches, and deletion. A loop in $\mathbf{f}_1$ is $p_{3458763}$. Simple paths of $\mathbf{f}_1$ are p_{123}, p_{3458}, p_{3678}, and p_{89}. p_{123} and p_{89} are branches of $\mathbf{f}_1$. $\mathbf{f}_2$ and $\mathbf{f}_3$ were obtained by deleting p_{3678} and p_{123}, respectively, from $\mathbf{f}_1$.

data set $\mathcal{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ by

$$\tau = 4\sum_{i=1}^{n} \sqrt{\Delta_n(\mathbf{x}_i, G)}.$$

Deleting Short Branches

Small protrusions on the contours of the character template tend to result in short branches in the initial skeleton graph. We first approached this problem by deleting any branch that is shorter than a threshold, τ_{branch}. Unfortunately, this approach proved to be too simple in practice. By setting τ_{branch} to a relatively large value, we eliminated a lot of short branches that represented "real" parts of the character, whereas by setting τ_{branch} to a relatively small value, a lot of "noisy" branches remained in the graph. We found that after the first fitting-and-smoothing step, if the size of the protrusion is comparable to the thickness of the skeleton, i.e., the protrusion is likely to be a "real" part of the skeleton, the angles of the short branch and the connecting paths tend to be close to right angle (Figure 33(a)). On the other hand, if the short branch has been created by the noisy contour of the character, the angle of the short branch and one of the connecting path tends to be very sharp (Figure 33(b)). So, in the decision of deleting the short branch $p_{i,i_3,\ldots}$ (Figure 33), we weight the length of the branch by

$$w_i = 1 - \cos^2\gamma$$

where

$$\gamma = \min(\gamma_{i_1,i,i_3}, \gamma_{i_2,i,i_3}),$$

and we delete $p_{i,i_3,\ldots}$ if $w_i l(p_{i,i_3,\ldots}) < \tau_{branch}$. Experiments showed that to delete most of the noisy branches without removing essential parts of the skeleton, τ_{branch} should be set between τ and 2τ.

Figure 34 shows three skeleton graphs before and after the deletions. To avoid recursively pruning longer branches, we found it useful to sort the short branches in increasing order by their length, and deleting them in that order. This was especially important in the case of extremely noisy skeleton graphs such as depicted by Figures 34(a) and (b).

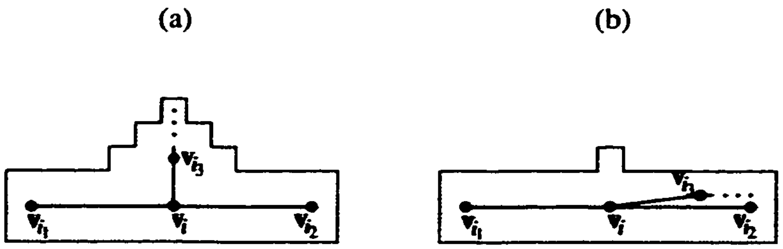


Figure 33: If the protrusion is a "real" part of the skeleton, the angles of the short branch and the connecting paths tend to be close to right angle (a), whereas if the short branch has been created by a few noisy pixels on the contour of the character, the short branch tends to slant to one of the connecting paths during the first fitting-and-smoothing step (b).

Removing Short Loops

Short loops created by thinning algorithms usually indicate isolated islands of white pixels in the template. We remove any loop from the skeleton graph if its length is below a threshold τ_{loop}. A loop is removed by deleting the longest simple path it contains. Experiments showed that to remove most of the noisy loops without removing essential parts of the skeleton, τ_{loop} should be set between 2τ and 3τ. Figure 35 shows three skeleton graphs before and after the operation.

Merging Star3-Vertices

In experiments we found that if two penstrokes cross each other at a sharp angle, the thinning procedure tends to create two star3-vertices connected by a short simple path rather then a star4-vertex. The first approach to correct this imperfection was to merge any two star3-vertices that are connected by a simple path shorter than a threshold, τ_{star3}. Unfortunately, this approach proved to be too simple in practice. By setting τ_{star3} to a relatively large value, we eliminated a lot of short paths that were not created by crossing penstrokes, whereas by setting τ_{star3} to a relatively small value, a lot of paths created by crossing penstrokes remained in the graph. We found that it is more likely that the short path $p_{i,\ldots,j}$ (Figure 36) is created by two crossing penstrokes if the angles γ_{i_1,i,i_2} and γ_{j_1,j,j_2} are small. To avoid merging $\mathbf{v}_i$ and $\mathbf{v}_j$ when these two angles are large, we weight the length of the path $p_{i,\ldots,j}$ by an experimentally developed factor

$$w_{ij} = \left[\frac{(1-\cos\gamma_{i_1,i,i_2}) + (1-\cos\gamma_{j_1,j,j_2})}{4}\right]^3,$$

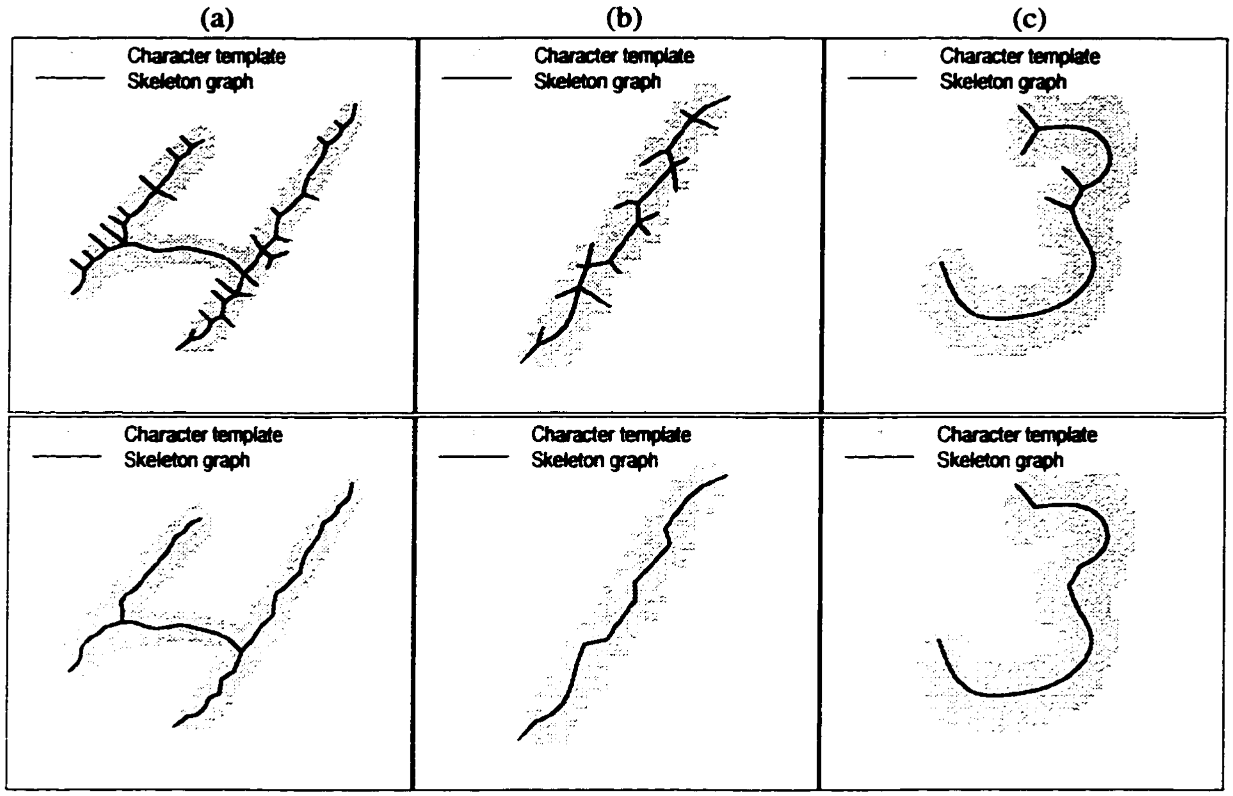


Figure 34: Deleting short branches. Skeleton graphs before (top row) and after (bottom row) the deletion.

and we merge $\mathbf{v}_i$ and $\mathbf{v}_j$ if $w_{ij}l(p_{i,\ldots,j}) < \tau_{star3}$. When merging two star3-vertices $\mathbf{v}_i$ and $\mathbf{v}_j$, we first delete the path $p_{i,\ldots,j}$, and remove $\mathbf{v}_i$ and $\mathbf{v}_j$. Then we add a new vertex $\mathbf{v}_{new}$ and connect the four remaining neighbors of $\mathbf{v}_i$ and $\mathbf{v}_j$ to $\mathbf{v}_{new}$ (Figure 36). Experiments indicated that for the best results τ_{star3} should be set between 0.5τ and τ. Figure 37 shows three skeleton graphs before and after the merge.

Updating Star3 and Star4-Vertices

Initially, all the junction-vertices of the skeleton are either star3 or star4-vertices. After the skeleton has been smoothed by the first fitting-and-smoothing step and cleaned by the restructuring operations described above, we update the junction-vertices of the skeleton to Y, T and X-vertices depending on the local geometry of the junction-vertices and their neighbors. A star4-vertex is always updated to an X-vertex. When updating a star3-vertex, we face the same situation as when degrading an X-vertex, so a star3-vertex is updated to a T-vertex if two of the angles at the vertex are between 80 and 100 degrees, otherwise it is updated to a Y-vertex. The formal rules are given in the last two rows of Table 4.

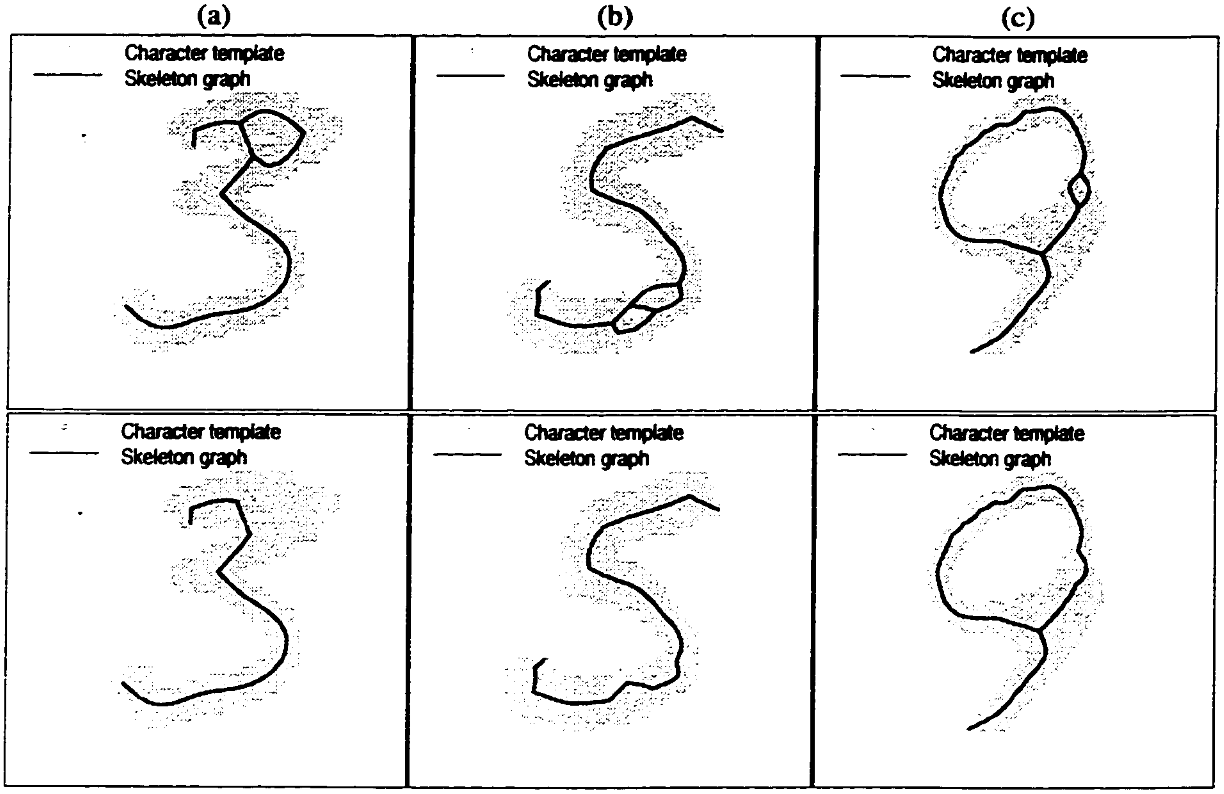


Figure 35: Removing short loops. Skeleton graphs before (top row) and after (bottom row) the removal.

Filtering Vertices

In this step, we iteratively remove every line-vertex whose two incident edges are shorter than a threshold τ_{filter}. Formally, any line-vertex $\mathbf{v}_j$ is removed from the graph if

$$\|\mathbf{v}_j - \mathbf{v}_i\| < \tau_{filter} \quad \text{and} \quad \|\mathbf{v}_j - \mathbf{v}_\ell\| < \tau_{filter}$$

where $\mathbf{v}_i$ and $\mathbf{v}_\ell$ are the neighbors of $\mathbf{v}_j$. When a line-vertex $\mathbf{v}_j$ is removed, the two edges incident to $\mathbf{v}_j$, $\mathbf{s}_{ij}$ and $\mathbf{s}_{j\ell}$, are also removed. Then the two former neighbors of $\mathbf{v}_j$ are connected by a new edge (Figure 38).

Filtering vertices is an optional operation. It can be used to speed up the optimization if the character template has a high resolution since in this case the initial skeleton graph has much more vertices than it is needed for reasonably smooth approximation. It can be also used after the optimization to improve the compression rate if the objective is to compress the image by storing the character skeleton instead of the template. In this case the filtering operation can be coupled with a smoothing operation at the other end where the character is recovered based on the skeleton graph (see Section 6.3.2). Figure 39 shows an example of a skeleton graph before and after the filtering operation.

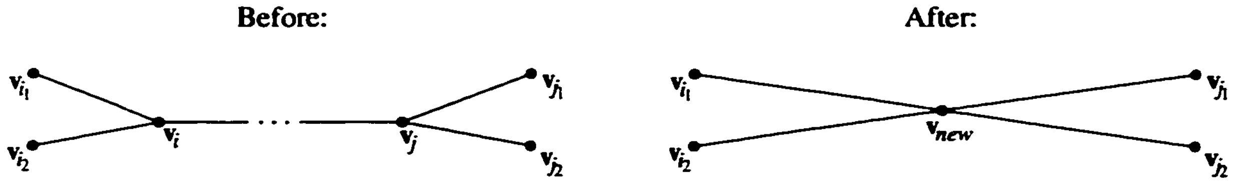


Figure 36: When merging two star3-vertices, we remove the vertices and the path connecting them. Then we add a new vertex and connect the former neighbors of the two star3-vertices to the new vertex.

6.3 Experimental Results

6.3.1 Skeletonizing Isolated Digits

In this section we report results on isolated hand-written digits from the NIST Special Database 19 [Gro95]. To set the parameters and to tune the algorithm, we chose 100 characters per digit randomly. Figures 40 through 49 display eight templates for each digit. These examples were chosen so that they roughly represent the 100 characters both in terms of the variety of the input data and in terms of the success rate of the algorithm. To illuminate the contrast between the pixelwise skeleton of the character and the skeleton graph produced by the principal graph algorithm, we show the initial graph (upper row in each figure) and the final graph (lower row in each figure) for each chosen character template. The length thresholds of the restructuring operations were set to the values indicated by Table 5.

τ_{branch}	τ_{loop}	τ_{star3}	τ_{filter}
1.2τ	3τ	τ	τ

Table 5: Length thresholds of the restructuring operations in experiments with isolated digits.

The results indicate that the principal graph algorithm finds a smooth medial axis of the great majority of the characters. In the few cases when the skeleton graph is imperfect, we could identify two sources of errors. The first cause is that, obviously, the restructuring operations do not work perfectly for all the characters. For instance, short branches can be cut (Figure 46(a)), short loops can be eliminated (Figure 42(c)), or star3-vertices can be merged mistakenly (Figure 44(h)). To correct these errors, one has to include some a-priori information in the process, such as a collection of possible configurations of skeleton graphs that can occur in hand-written digits. The other source of errors is that at this phase, we do not have restructuring operations that *add* components to the skeleton graph. For instance, skeleton graphs in Figures 42(e) and 48(d) could be improved by connecting broken lines based on the closeness of their endpoints. One could also add short paths to create branches or loops that were missing from the initial graph (Figures 42(b) and 48(f)). This operation could be based on local thickness measurements along the graph that could point out protrusions caused by overlapping lines in the character. The exact definitions and implementations

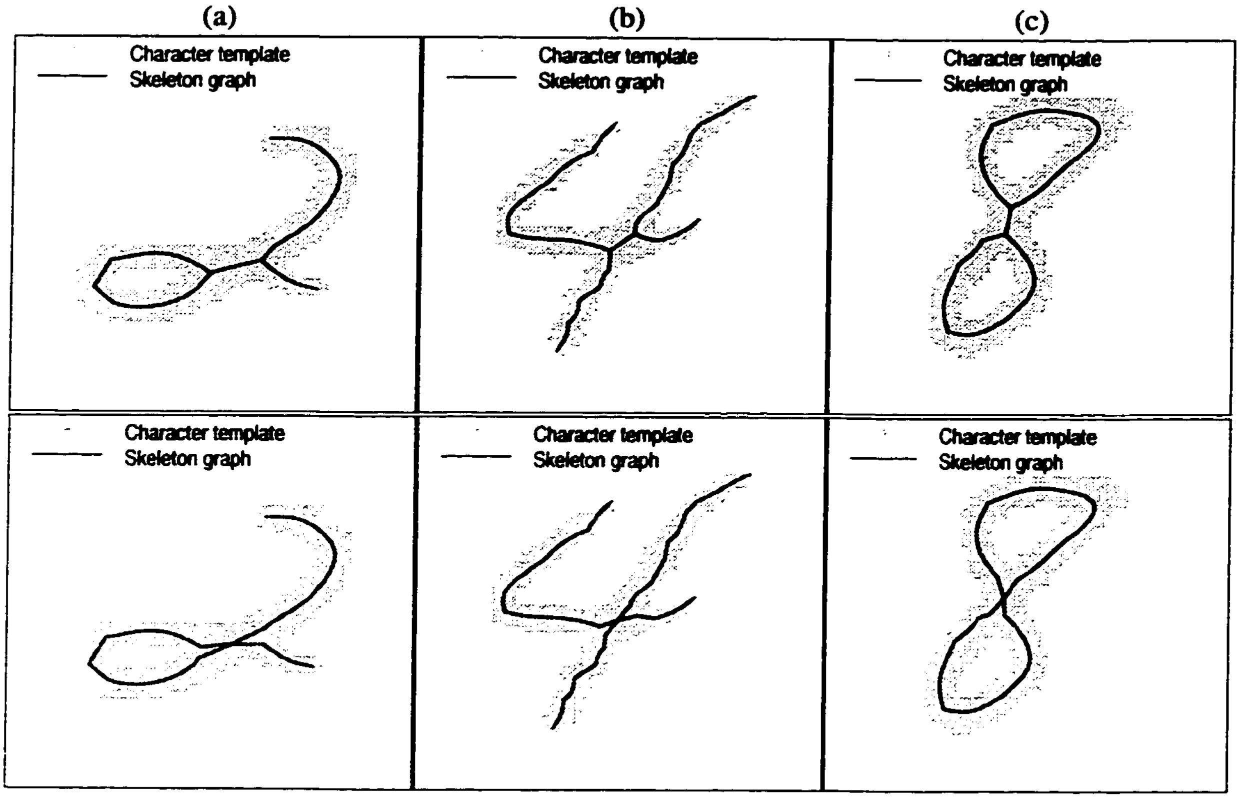


Figure 37: Merging star3-vertices. Skeleton graphs before (top row) and after (bottom row) the merge.

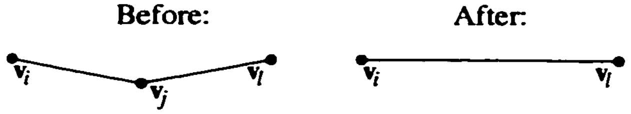


Figure 38: Removing the line-vertex $\mathbf{v}_j$ in the filtering operation.

of these operations are subjects of future research.

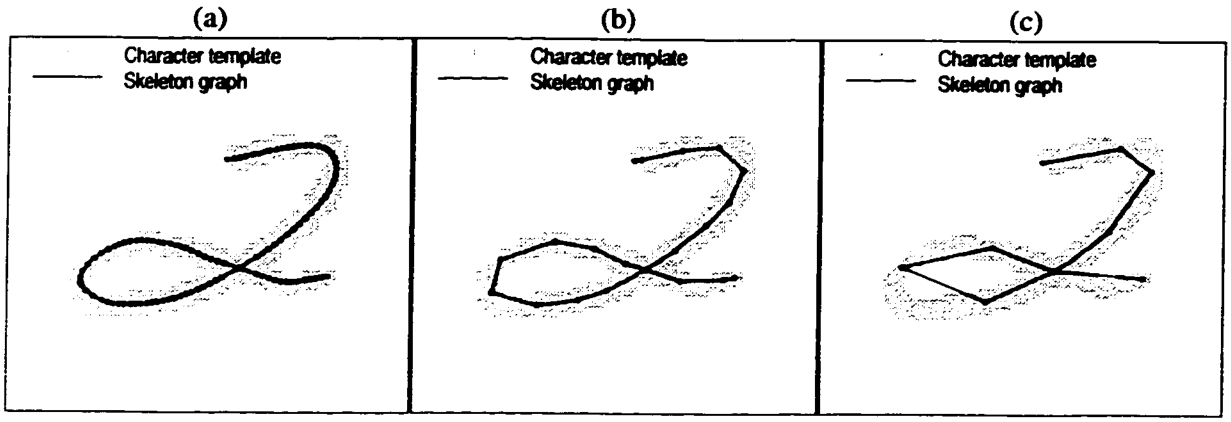


Figure 39: Filtering vertices. A skeleton graph (a) before filtering, (b) after filtering with $\tau_{filter} = 1$, and (b) after filtering with $\tau_{filter} = 2$.

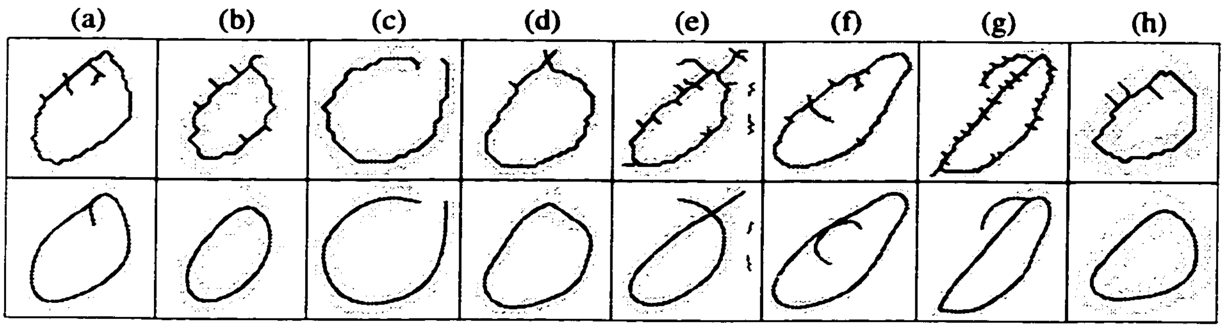


Figure 40: Skeleton graphs of isolated 0's. Initial (upper row) and final (lower row) skeletons.

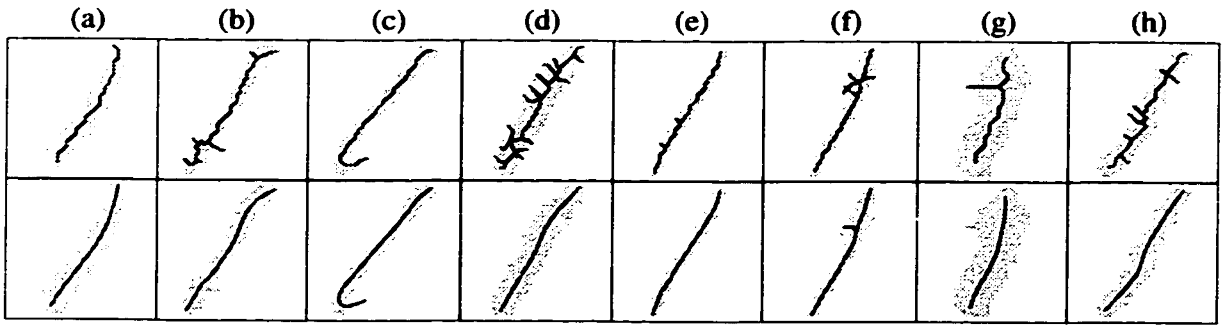


Figure 41: Skeleton graphs of isolated 1's. Initial (upper row) and final (lower row) skeletons.

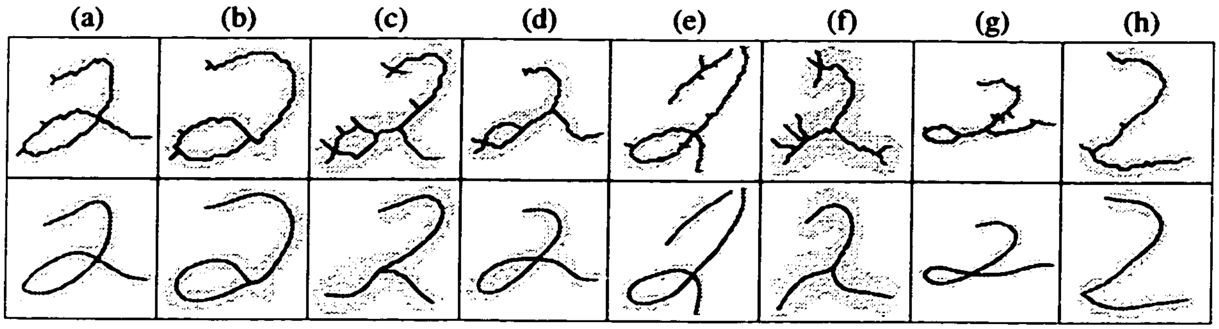


Figure 42: Skeleton graphs of isolated 2's. Initial (upper row) and final (lower row) skeletons.

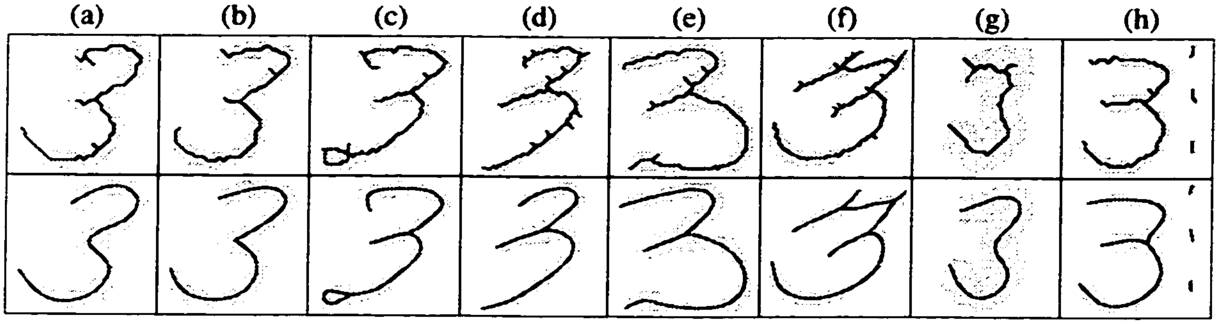


Figure 43: Skeleton graphs of isolated 3's. Initial (upper row) and final (lower row) skeletons.

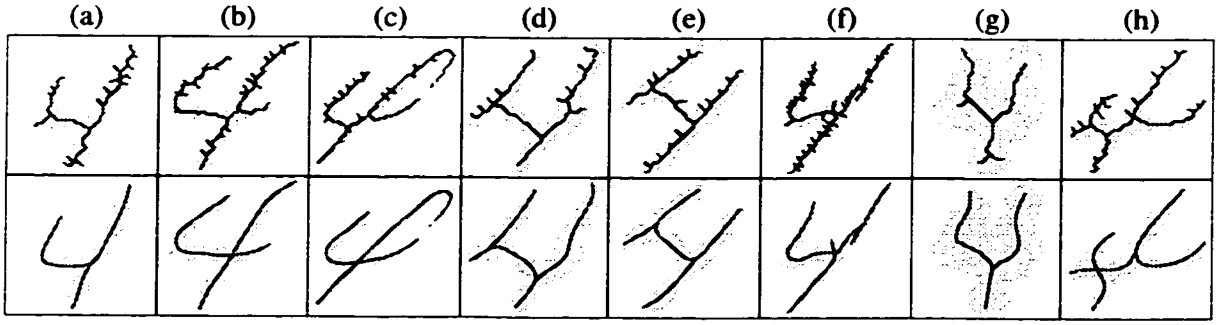


Figure 44: Skeleton graphs of isolated 4's. Initial (upper row) and final (lower row) skeletons.

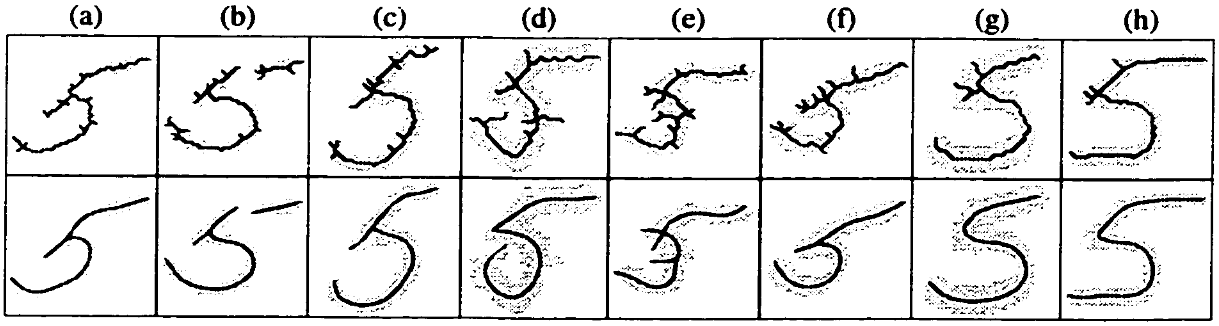


Figure 45: Skeleton graphs of isolated 5's. Initial (upper row) and final (lower row) skeletons.

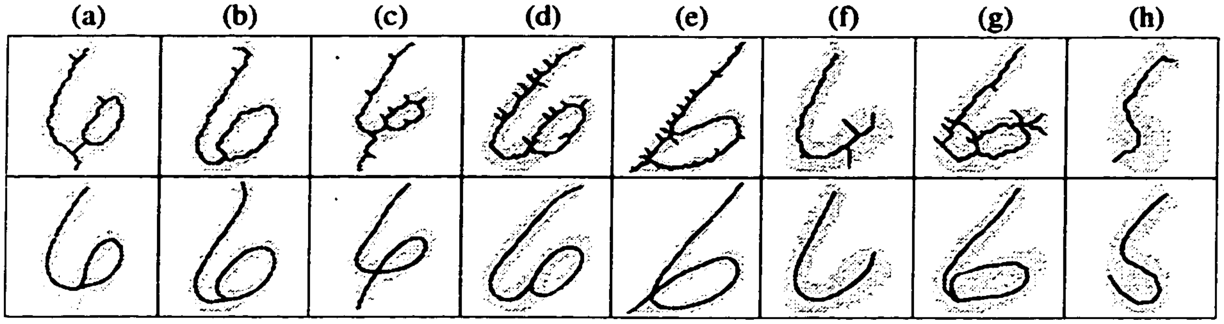


Figure 46: Skeleton graphs of isolated 6's. Initial (upper row) and final (lower row) skeletons.

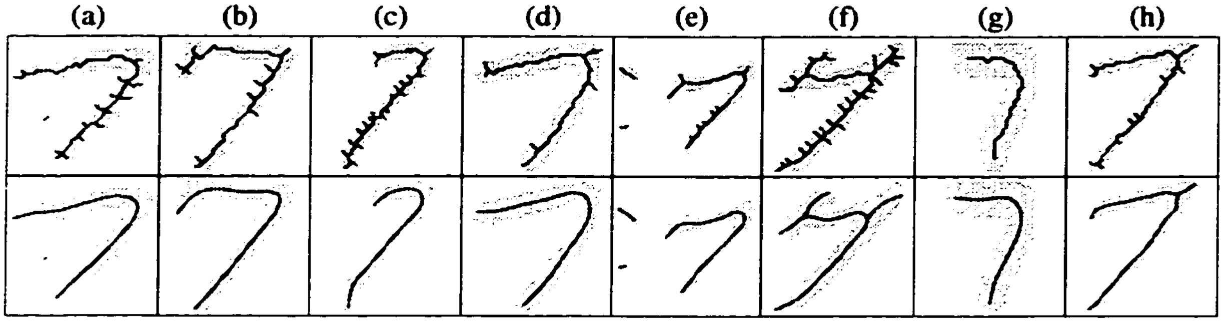


Figure 47: Skeleton graphs of isolated 7's. Initial (upper row) and final (lower row) skeletons.

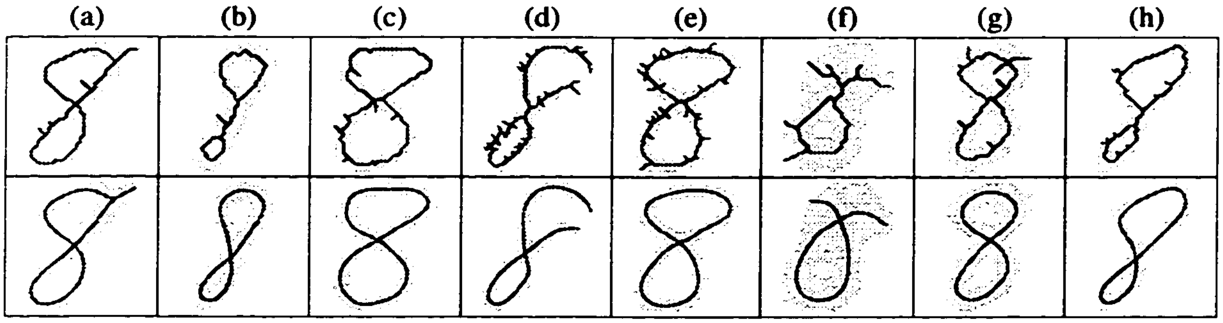


Figure 48: Skeleton graphs of isolated 8's. Initial (upper row) and final (lower row) skeletons.

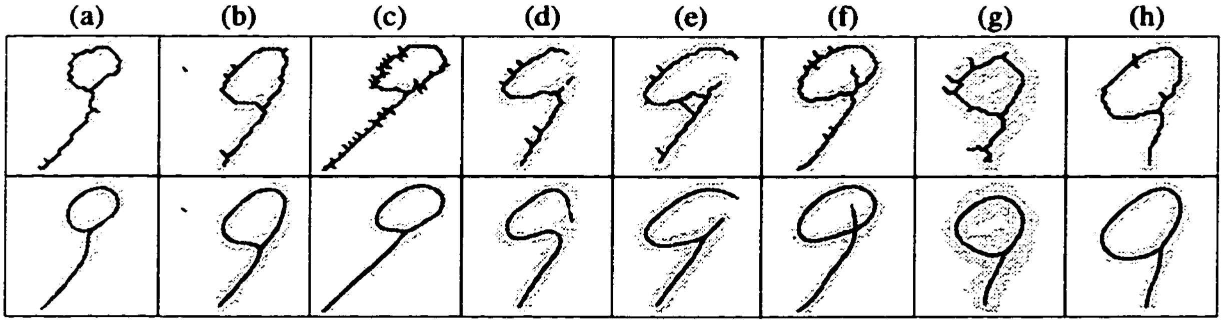


Figure 49: Skeleton graphs of isolated 9's. Initial (upper row) and final (lower row) skeletons.

6.3.2 Skeletonizing and Compressing Continuous Handwriting

In this section we present results of experiments with images of continuous handwriting. We used the principal graph algorithm to skeletonize short pangrams (sentences that contain all the letters of the alphabet) written by different individuals. The emphasis in these experiments was on using the skeleton graph for representing hand-written text efficiently.

Figure 50 shows the images of two pangrams written by two individuals. For the sake of easy referencing, hereafter we will call them Alice (Figure 50(a)) and Bob (Figure 50(b)). After scanning the images, the principal graph algorithm was used to produce the skeleton graphs depicted by Figure 51. Since the images were much cleaner than the images of isolated digits used in the previous section, τ_{branch} and τ_{loop} were set slightly lower than in the previous experiments. We also found that the incorrect merge of two star3-vertices has a much worse visual effect than not merging two star3-vertices when they should be merged, so we set τ_{star3} to half of the value that was used in the experiments with isolated digits. Finally, we did not use filtering vertices in the restructuring step. The length thresholds of the restructuring operations were set to the values indicated by Table 6. The thickness of each curve in Figure 51 was set to the estimated thickness τ of the template.

τ_{branch}	τ_{loop}	τ_{star3}	τ_{filter}
τ	2τ	0.5τ	0

Table 6: Length thresholds of the restructuring operations in experiments with continuous handwriting. $\tau_{filter} = 0$ indicates that we did not filter vertices in the reconstruction step.

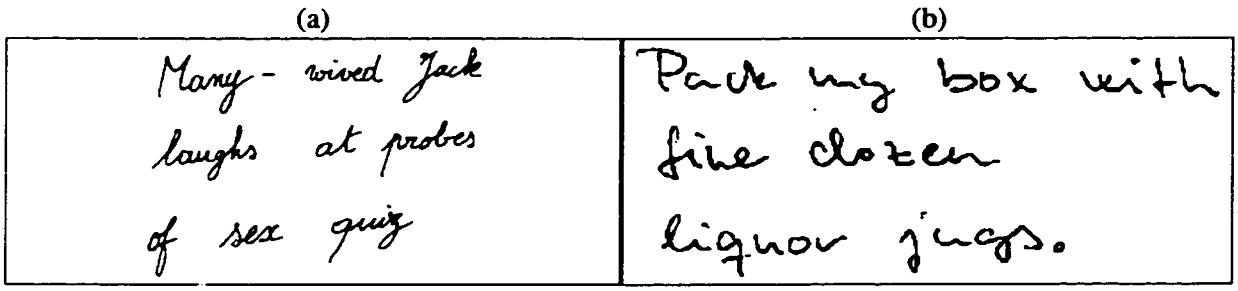


Figure 50: Original images of continuous handwritings. (a) Alice, (b) Bob.

To demonstrate the efficiency of representing the texts by their skeleton graphs, we applied the vertex filtering operation *after* the skeleton graphs were produced. For achieving high compression rate, τ_{filter} should be set to a relatively large value to remove most of the line-vertices from the skeleton graph. Since filtering with a large threshold has an unfortunate visual effect of producing sharp-angled polygonal curves (see Figure 39(c)), we fit cubic splines through the vertices of each path of the skeleton graph. Tables 7 and 8 show the results.

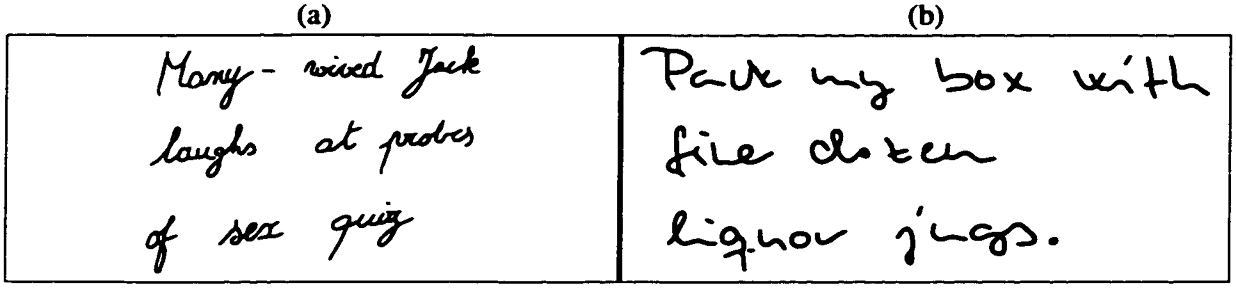


Figure 51: Skeleton graphs of continuous handwritings. (a) Alice, (b) Bob.

To be able to compute the number of bytes needed for storing the images, in the compression routine we also set the number of bits n_b used to store each coordinate of a vertex. The vertices are stored consecutively with one bit sequence of length n_b marking the end of a path. So, for example, when n_b is set to 8, the vertices of the skeleton graph are rounded to the points of a 255 × 255 rectangular grid, and the remaining byte is used to mark the end of a path. By using this scheme, the skeleton graph can be stored by using

$$N = \lceil (n_p + 2m)n_b/8 \rceil$$

bytes where n_p is the number of paths and m is the number of vertices. Tables 7 and 8 show the skeleton graphs and the number of bytes needed to store the images. The numbers of paths in Alice's and Bob's texts are 144 and 62, respectively. As a comparison, the size of the raw bitmap compressed by using the Liv-Zempel algorithm (gzip under UNIX) is 2322 bytes in Alice's case, and 1184 bytes in Bob's case. So, for instance, if the filter threshold is set to 6τ, and 8 bits are used to store the coordinates of the vertices, the algorithm produces a skeleton that approximates the original text quite well while compressing the image to less than half of the size of the gzipped raw bitmap. Note that the bit sequence representing the skeleton graph can be further compressed by using traditional compression methods.

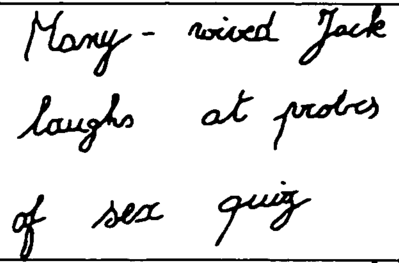
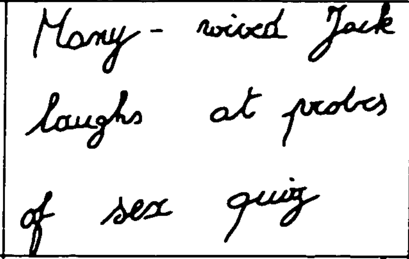
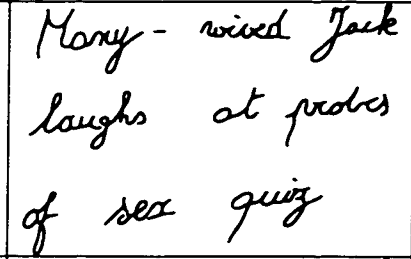
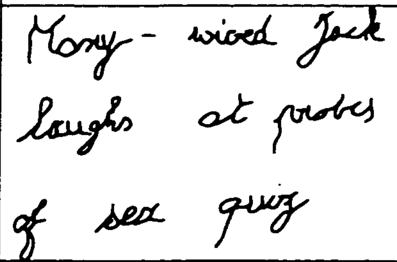
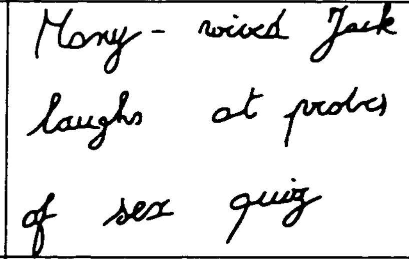
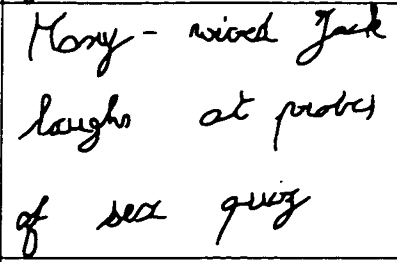
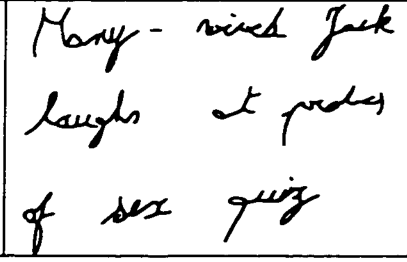
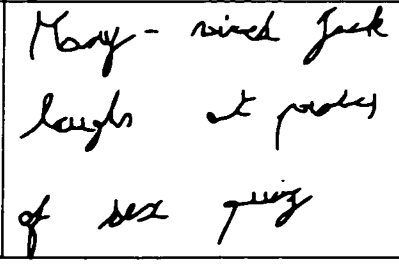
τ_{filter}	m	$n_b = 8$		$n_b = 6$	
		Skeleton graph	N	**Skeleton graph**	N
2τ	747		1638	–	–
4τ	497		1138	–	–
6τ	404		952		714
10τ	359		862		647
20τ	319		782		587

Table 7: Compression of Alice's handwriting. m is the number of vertices, n_b is the number of bits used to store each coordinate of a vertex, and N is the total number of bytes needed to store the skeleton graph of the image.

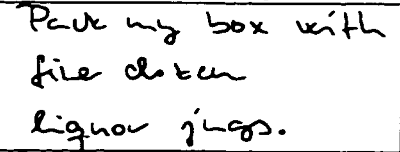
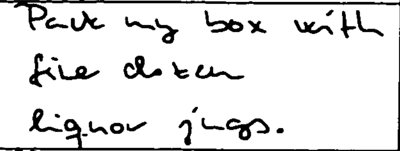
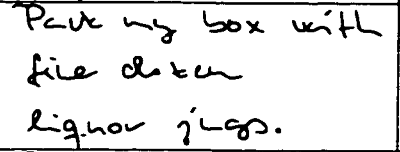
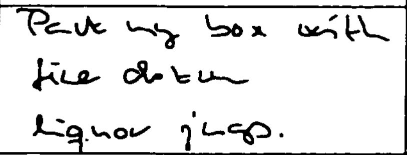
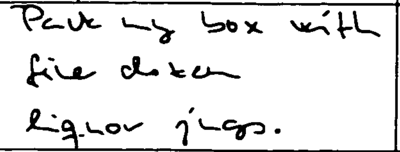
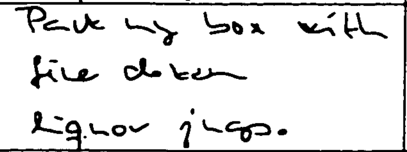
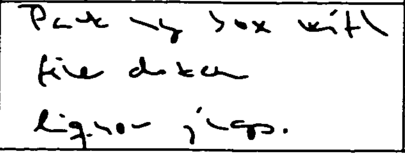
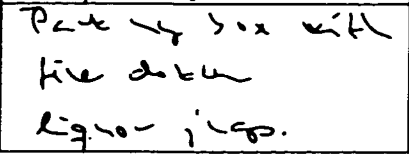
τ_{filter}	m	$n_b = 8$		$n_b = 6$	
		Skeleton graph	N	**Skeleton graph**	N
2τ	432		936	–	–
4τ	278		628	–	–
6τ	223		518		389
10τ	193		458		344
20τ	170		412		309

Table 8: Compression of Bob's handwriting. m is the number of vertices, n_b is the number of bits used to store each coordinate of a vertex, and N is the total number of bytes needed to store the skeleton graph of the image.

Chapter 7

Conclusion

The three pillars of the thesis are the theoretical results described in Chapter 4, the polygonal line algorithm proposed in Chapter 5, and the experimental results with the principal graph algorithm presented in Chapter 6. Below we summarize our main results in these three areas, and briefly discuss some of the possible areas of future research.

Theory

We proposed a new definition of principal curves with a length constraint. Based on the new definition, we proved the following two results.

- **Existence of principal curves.** Principal curves in the new sense exist for all distributions with final second moments.

- **Consistency and rate of convergence.** For distributions concentrated on a bounded and closed convex set, an estimator of the principal curve can be constructed based on a data set of n i.i.d. sample points such that the expected loss of the estimator converges to the loss of the principal curve at a rate of $n^{-1/3}$.

Two interesting open problems are the following.

- **Concrete principal curves.** It would be of both theoretical and practical interest if concrete examples of principal curves of basic multivariate densities could be found.

- **A more practical constraint.** It would be convenient to replace the length constraint with a practically more suitable restriction, such as a limit on the maximum curvature of the curve, that is more closely related to the curvature penalty applied in the practical algorithm.

Algorithm

Our main result here is a practical algorithm to estimate principal curves based on data. Experimental results on simulated data demonstrate that the polygonal line algorithm compares favorably to previous methods both in terms of performance and computational complexity.

A possible area of further research is to extend the polygonal line algorithm to find multi-dimensional manifolds. There are two fundamentally different approaches to extend principal curves to principal surfaces or to arbitrary-dimensional principal manifolds. In the first approach, the theoretical model and the algorithm are either extended to include smooth non-parametric surfaces [Has84, HS89], or they can be used, without modification, to find arbitrary-dimensional principal manifolds [SMS98, SWS98]. The second approach follows the strategy of an iterative PCA algorithm which finds the ith largest principal component by finding the first principal component in the linear subspace orthogonal to the first $i-1$ principal components. In the second approach, therefore, the one-dimensional principal curve routine is called iteratively so that in ith iteration, we compute the principal curve of the data set obtained by subtracting from the data points their projections to the principal curve computed in the $(i-1)$th iteration [Del98, CG98b, DM95].

Theoretically, it is not impossible to use the first approach, i.e., to extend the polygonal line algorithm to find arbitrary-dimensional piecewise linear manifolds. Technically, however, it is not clear at this point how this extension could be done. The second approach, on the other hand, seems feasible to be implemented with the polygonal line algorithm. The exact design of the algorithm is subject of future research.

Applications

We proposed an extended version of the polygonal line algorithm to find principal graphs of data sets obtained from binary templates of black-and-white images. Test results indicate that the principal graph algorithm can be used to find a smooth medial axis of a wide variety of character templates, and to represent hand-written text efficiently.

Here, the main objective of future research is to improve the computational complexity of the method. At this point the "general purpose" polygonal line algorithm is used as the "main engine" for the principal graph algorithm. We expect that by incorporating the special features of the highly structured data obtained from binary templates into the algorithm, the efficiency of the algorithm can be increased substantially.

Bibliography

[AH69] T. M. Alcorn and C. W. Hoggar. Preprocessing of data for character recognition. *Marconi Review*, 32:61–81, 1969.

[Ale84] K. Alexander. Probability inequalities for empirical processes and a law of the iterated logarithm. *Annals of Probability*, pages 1041–1067, 1984.

[Ash72] R. B. Ash. *Real Analysis and Probability*. Academic Press, New York, 1972.

[Bar87] D. J. Bartholomew. *Latent Variable Models and Factor Analysis*. Charles Griffin & Co. Ltd., London, 1987.

[BCJL94] P. V. Balakrishnan, M. C. Cooper, V. S. Jacob, and P. A. Lewis. A study of the classification capabilities of neural networks using unsupervised learning: a comparison with k-means clustering. *Psychometrika*, 59(4):509–525, 1994.

[BD75] J. Bezdek and J. Dunn. Optimal fuzzy partitions: A heuristic for estimating the parameters in a mixture of normal distributions. *IEEE Transactions on Computers*, 24:835–838, 1975.

[BDHW97] G. Balzuweit, R. Der, M. Herrmann, and M. Welk. An algorithm for generalized principal curves with adaptive topology in complex data sets. Technical Report 3/97, Institut für Informatik, Universität Leipzig, 1997.

[BLL98] P. Bartlett, T. Linder, and G. Lugosi. The minimax distortion redundancy in empirical quantizer design. *IEEE Transactions on Information Theory*, 44(5):1802–1813, 1998.

[BN95] J. C. Bezdek and R. P. Nikhil. An index of topological preservation for feature extraction. *Pattern Recognition*, 28(3):381–391, 1995.

[BR92] J. D. Banfield and A. E. Raftery. Ice floe identification in satellite images using mathematical morphology and clustering about principal curves. *Journal of the American Statistical Association*, 87:7–16, 1992.

[BSW96] C. M. Bishop, M. Svensén, and C. K. I. Williams. EM optimization of latent-variables density models. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 465–471. The MIT Press, 1996.

[BSW98] C. M. Bishop, M. Svensén, and C. K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–235, 1998.

[BT98] C. M. Bishop and M. E. Tipping. A hierarchical latent variable model for data visualization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):281–293, 1998.

[CG92] G. Celeux and G. Govaert. A classification EM algorithm and two stochastic versions. *Computational Statistics and Data Analysis*, 14:315–332, 1992.

[CG98a] K. Chang and J. Ghosh. Principal curve classifier – a nonlinear approach to pattern classification. In *IEEE International Joint Conference on Neural Networks*, pages 695–670, Anchorage, AL, May 5–9 1998.

[CG98b] K. Chang and J. Ghosh. Principal curves for nonlinear feature extraction and classification. In *Applications of Artificial Neural Networks in Image Processing III*, volume 3307, pages 120–129, San Jose, CA, Jan 24–30 1998. SPIE Photonics West '98 Electronic Image Conference.

[Cho94] P. A. Chou. The distortion of vector quantizers trained on n vectors decreases to the optimum as $o_p(1/n)$. In *Proceedings of IEEE International Symposium on Information Theory*, Trondheim, Norway, 1994.

[Cle79] W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74:829–835, 1979.

[DBH96] R. Der, G. Balzuweit, and M. Herrmann. Constructing principal manifolds in sparse data sets by self-organizing maps with self-regulating neighborhood width. In *Proceedings of the International Conference on Neural Networks*, pages 480–483, 1996.

[Del98] P. Delicado. Principal curves and principal oriented points. Technical Report 309, Department d'Economia i Empresa, Universitat Pompeu Fabra, 1998.

[Deu68] E. S. Deutsch. Preprocessing for character recognition. In *Proceedings of the IEE NPL Conference on Pattern Recognition*, pages 179–190, 1968.

[DGL96] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, New York, 1996.

[Din55] G. P. Dinnen. Programming pattern recognition. In *Proceedings of the Western Joint Computer Conference*, pages 94–100, New York, 1955.

[DK82] P. A. Devijver and J. Kittler. *Pattern Recognition: a Statistical Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[DLR77] A.P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B*, 39:1–38, 1977.

[DM95] D. Dong and T. J. McAvoy. Nonlinear principal component analysis – based on principal curves and neural networks. *Computers Chem. Engineering*, 20(1):65–78, 1995.

[DP97] A. Datta and S. K. Parui. Skeletons from dot patterns: A neural network approach. *Pattern Recognition Letters*, 18:335–342, 1997.

[DS96a] T. Duchamp and W. Stuetzle. Extremal properties of principal curves in the plane. *Annals of Statistics*, 24(4):1511–1520, 1996.

[DS96b] T. Duchamp and W. Stuetzle. Geometric properties of principal curves in the plane. In Helmut Rieder, editor, *Robust statistics, data analysis, and computer intensive methods: in honor of Peter Huber's 60th birthday*, volume 109 of *Lecture notes in statistics*, pages 135–152. Springer-Verlag, 1996.

[EOS92] E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67:47–55, 1992.

[Eve84] B. S. Everitt. *An Introduction to Latent Variable Models*. Chapman and Hall, London, 1984.

[Fle97] A. Flexer. Limitations of self-organizing maps for vector quantization and multidimensional scaling. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference*, volume 9, pages 445–451. MIT Press, 1997.

[Fle99] A Flexer. On the use of self-organizing maps for clustering and visualization. In J.M. Zytkow and J. Rauch, editors, *Principles of Data Mining and Knowledge Discovery, Third European Conference, PKDD'99*, Lecture Notes in Artificial Intelligence 1704, pages 80–88, Prague, Czech Republic, 1999. Springer.

[Föl89] P. Földiák. Adaptive network for optimal linear feature extraction. In IEEE Press, editor, *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, volume 1, pages 401–405, New York, 1989.

[GG92] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer, Boston, 1992.

[GKL80] R. M. Gray, J. C. Kieffer, and Y. Linde. Locally optimal block quantizer design. *Information and Control*, 45:178–198, May 1980.

[Gro95] P. Grother. *NIST Special Database 19*. National Institute of Standards and Technology, Advanced Systems Division, 1995.

[Har75] J. A. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.

[Has84] T. Hastie. *Principal curves and surfaces*. PhD thesis, Stanford University, 1984.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[Hot33] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 498–520, 1933.

[HS89] T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84:502–516, 1989.

[JD88] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentica Hall, Englewood Cliffs, New Jersey, 1988.

[JW92] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[Kar47] K. Karhunen. Über lineare methoden in der wahrscheinlichkeitsrechnung. *Amer. Acad. Sci., Fennicade, Ser. A, I*, 37:3–79, 1947. (Translation: RAND Corporation, Santa Monica, California, Rep. T-131, Aug. 1960).

[KCRU57] R. A. Kirsh, L. Cahn, C. Ray, and G. J. Urban. Experiment in processing pictorial information with a digital computer. In *Proceedings of the Eastern Joint Computer Conference*, pages 221–229, New York, 1957.

[KKLZ98] B. Kégl, A. Krzyżak, T. Linder, and K. Zeger. Principal curves: Learning and convergence. In *Proceedings of IEEE International Symposium on Information Theory*, page 387, 1998.

[KKLZ99] B. Kégl, A. Krzyżak, T. Linder, and K. Zeger. A polygonal line algorithm for constructing principal curves. In *Advances in Neural Information Processing Systems*, volume 11, pages 501–507. The MIT Press, 1999.

[KKLZ00] B. Kégl, A. Krzyżak, T. Linder, and K. Zeger. Learning and design of principal curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000. (to appear).

[Koh82] T. Kohonen. Clustering, taxonomy, and topological maps of patterns. In *Proceedings of the 6th International Conference on Pattern Recognition*, pages 114–128, Munich, 1982.

[Koh97] T. Kohonen. *The Self-Organizing Map*. Springer-Verlag, 2nd edition, 1997.

[KR95] R. E. Kass and A. E. Raftery. Bayes factors. *Journal of the American Statistical Association*, pages 773–795, 1995.

[Kra91] M.A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991.

[KT61] A. N. Kolmogorov and V. M. Tikhomirov. ε-entropy and ε-capacity of sets in function spaces. *Translations of the American Mathematical Society*, 17:277–364, 1961.

[KW78] J. B. Kruskal and M. Wish. Multidimensional scaling. In *Sage University Paper Series on Quantitative Applications in the Social Sciences*, number 07-011. Sage Publications, Beverly Hills, California, 1978.

[LBG80] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, COM-28:84–95, 1980.

[LLS93] S. W. Lee, L. Lam, and C.Y. Suen. A systematic evaluation of skeletonization algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(5):1203–1225, 1993.

[LLZ94] T. Linder, G. Lugosi, and K. Zeger. Rates of convergence in the source coding theorem, in empirical quantizer design and in universal lossy source coding. *IEEE Transactions on Information Theory*, 40:1728–1740, 1994.

[Lut90] S. P. Luttrell. Derivation of a class of training algorithms. *IEEE Transactions on Neural Networks*, pages 229–232, 1990.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, pages 281–296, 1967.

[Mac95] D. J. C. MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research*, 354(1):73–80, 1995.

[MAG91] S. Mahmoud, I. Abuhaiba, and R. Green. Skeletonization of arabic characters using clustering based skeletonization algorithm (CBSA). *Pattern Recognition*, 24(5):453–464, 1991.

[MC95] F. Mulier and V. Cherkassky. Self-organization as an iterative kernel smoothing process. *Neural Computation*, 7:1165–1177, 1995.

[MMT95] E. C. Malthouse, R. H. S. Mah, and A. C. Tamhane. Some theoretical results on nonlinear principal component analysis. In *Proceedings of the American Control Conference*, pages 744–748, June 1995.

[MZ97] N. Merhav and J. Ziv. On the amount of side information required for lossy data compression. *IEEE Transactions on Information Theory*, 43:1112–1121, 1997.

[NS84] N. J. Naccache and R. Shingal. SPTA: A proposed algorithm for thinning binary patterns. *IEEE Transactions on Systems, Man and Cybernetics*, 14(3), 1984.

[Oja92] E. Oja. Principal components, minor components, and linear neural networks. *Neural Networks*, 5:927–935, 1992.

[O'N66] B. O'Neil. *Elementary Differential Geometry*. Academic Press, Inc., Orlando, Florida, 1966.

[Pav80] T. Pavlidis. A thinning algorithm for discrete binary images. *Computer Graphics and Image Processing*, 13(2):142–157, 1980.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philos. Mag.*, 6(2):559–572, 1901.

[Pol81] D. Pollard. Strong consistency of *k*-means clustering. *Annals of Statistics*, 9:135–140, 1981.

[Pol82] D. Pollard. A central limit theorem for *k*-means clustering. *Annals of Probability*, 10:919–926, 1982.

[RMS92] H. Ritter, T. Martinetz, and K. Schulten. *Neural Computation and Self-Organizing Maps: An Introduction*. Addison-Wesley, Reading, Massachusetts, 1992.

[RN98] K. Reinhard and M. Niranjan. Subspace models for speech transitions using principal curves. *Proceedings of Institute of Acoustics*, 20(6):53–60, 1998.

[Row98] S. T. Roweis. EM algorithms for PCA and SPCA. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[RT89] J. Rubner and P. Tavan. A self-organizing network for principal component analysis. *Europhysics Letters*, 10:693–698, 1989.

[SA86] S. Suzuki and K. Abe. Sequential thinning of binary pictures using distance transformation. In *Proceedings of the 8th International Conference on Pattern Recognition*, pages 289–292, 1986.

[Sam69] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18:401–409, 1969.

[Sil85] B. W. Silverman. Some aspects of spline smoothing approaches to non-parametric regression curve fitting. *Journal of the Royal Statistical Society*, Ser. B, 47:1–52, 1985.

[SMS98] A. J. Smola, S. Mika, and B. Schölkopf. Quantization functionals and regularized principal manifolds. Technical Report NC2-TR-1998-028, NeuroCOLT2 Technical Report Series, 1998.
`http://www.neurocolt.com/abstracts/contents_1998.html.`

[SR97] D. Stanford and A. E. Raftery. Principal curve clustering with noise. Technical Report 317, Department of Statistics, University of Washington, 1997.

[SWP98] R. Singh, M. C. Wade, and N. P. Papanikolopoulos. Letter-level shape description by skeletonization in faded documents. In *Proceedings of the Fourth IEEE Workshop on Applications of Computer Vision*, pages 121–126. IEEE Comput. Soc. Press, 1998.

[SWS98] A. J. Smola, R. C. Williamson, and B. Schölkopf. Generalization bounds and learning rates for regularized principal manifolds. Technical Report NC2-TR-1998-027, NeuroCOLT2 Technical Report Series, 1998.
`http://www.neurocolt.com/abstracts/contents_1998.html.`

[Tal94] M. Talagrand. Sharper bounds for gaussian and empirical processes. *Annals of Probability*, 22:28–76, 1994.

[TB99] M.E. Tipping and C.M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society series B*, 1999. to appear.

[Tib92] R. Tibshirani. Principal curves revisited. *Statistics and Computation*, 2:183–190, 1992.

[TLF95] T. Tarpey, L. Li, and B. D. Flury. Principal points and self-consistent points of elliptical distributions. *Annals of Statistics*, 23(1):103–112, 1995.

[Vap98] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[VT68] H. L. Van Trees. *Detection, Estimation, and Modulation Theory*. Wiley, New York, 1968.

[Wil65] J. H. Wilkinson. *The algebraic eigenvalue problem*. Claredon Press, Oxford, England, 1965.

[WKIM98] N. G. Waller, H. A. Kaiser, J. B. Illian, and M. Manry. A comparison of the classification capabilities of the 1-dimensional Kohonen neural network with two partitioning and three hierarchical cluster analysis algorithms. *Psychometrika*, 63(1):5–22, 1998.

[YMMS92] N. Yamashita, M. Minami, M. Mizuta, and Y. Sato. A refined algorithm of principal curves and the evaluation of its complexity. *Bulletin of the Computational Statistics of Japan*, 5(1):33–43, 1992.