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 21. The second component $P(\mathbf{f})$ is a penalty on the average curvature of the curve

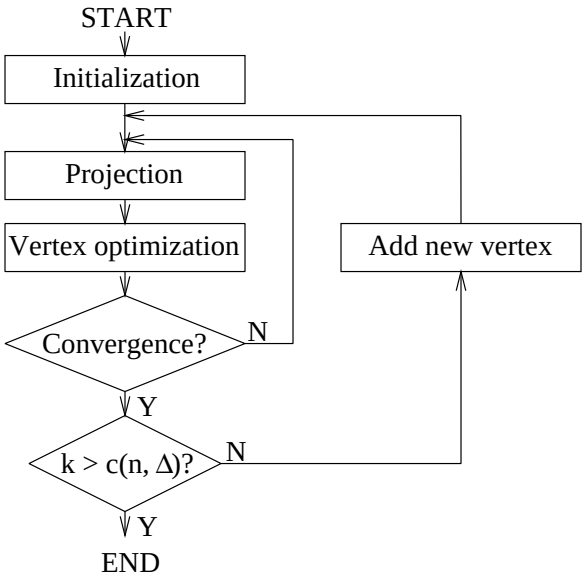

Figure 9: The fbw chart of the polygonal line algorithm.

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

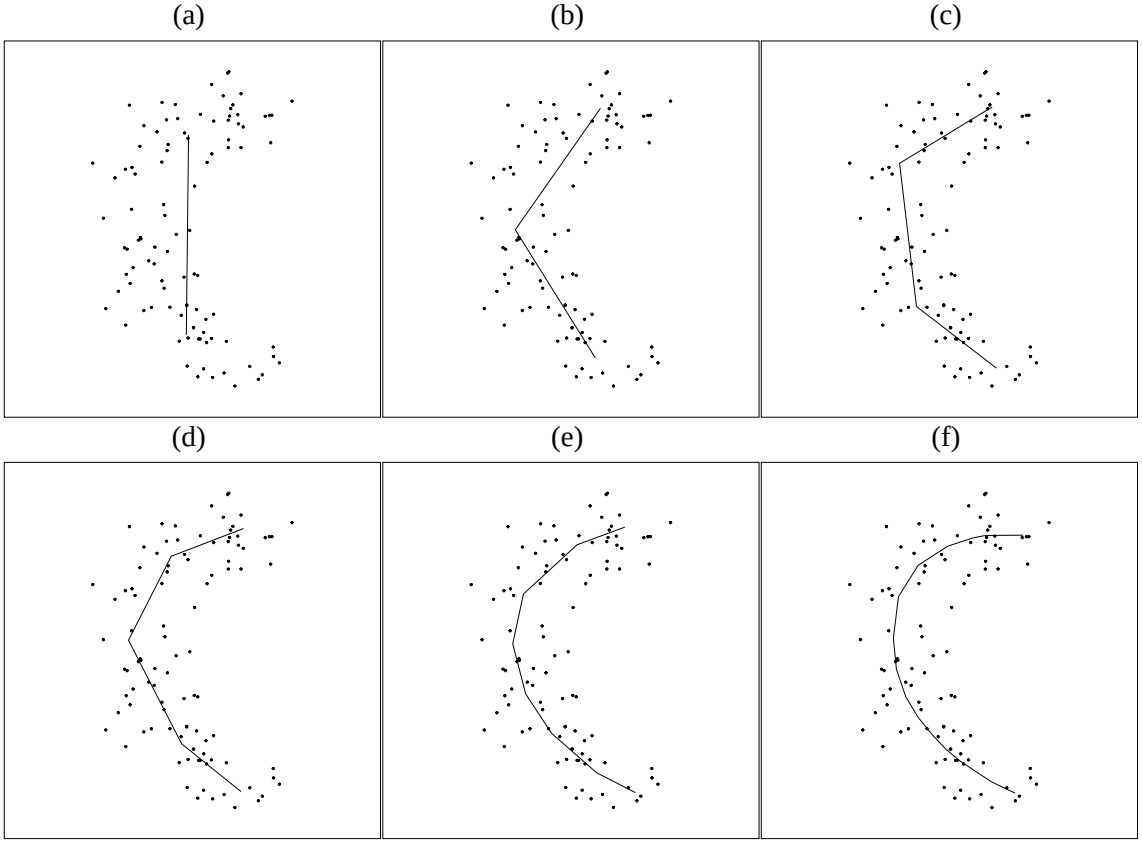

Figure 10: The curves $\mathbf{f}_{k,n}$ produced by the polygonal line algorithm for $n = 100$ data points. The data was generated by adding independent Gaussian errors to both coordinates of a point chosen randomly on a half circle. (a) $\mathbf{f}_{1,n}$, (b) $\mathbf{f}_{2,n}$, (c) $\mathbf{f}_{3,n}$, (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 21), 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

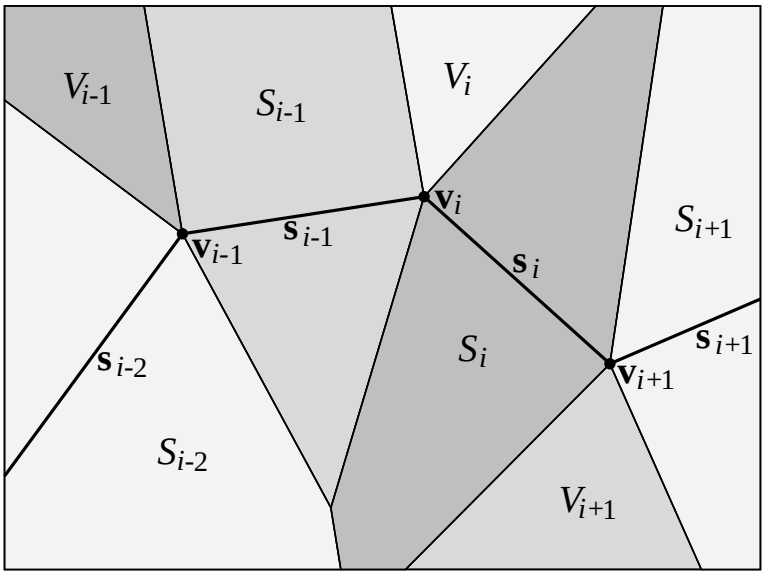

Figure 11: A nearest neighbor partition of $\mathbb{R}^2$ induced by the vertices and segments of $\mathbf{f}$. The nearest point of $\mathbf{f}$ to any point in the set V_i is the vertex $\mathbf{v}_i$. The nearest point of $\mathbf{f}$ to any point in the set S_i is a point of the line segment $\mathbf{s}_i$.

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 21, 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

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

and

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

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.

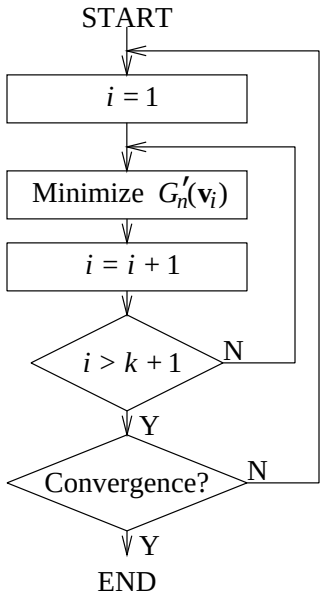

Figure 12: The flow chart of the optimization step.

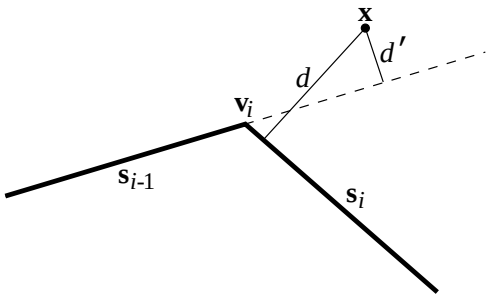

Figure 13: Assume that $\mathbf{x}$ belongs to S_{i-1}. The distance of $\mathbf{x}$ and the curve is d, while $\Delta_n(\mathbf{v}_i)$ measures the distance as d'.

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

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

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" number of nodes = number of points	"Discrete" 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.iro.umontreal.ca/~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 $\left(\begin{smallmatrix} 0.7 & 0.4 \\ -0.8 & 1.0 \end{smallmatrix}\right)$. In Figure 16(b) the transformation $\left(\begin{smallmatrix} -1.0 & -1.2 \\ 1.0 & -0.2 \end{smallmatrix}\right)$ 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

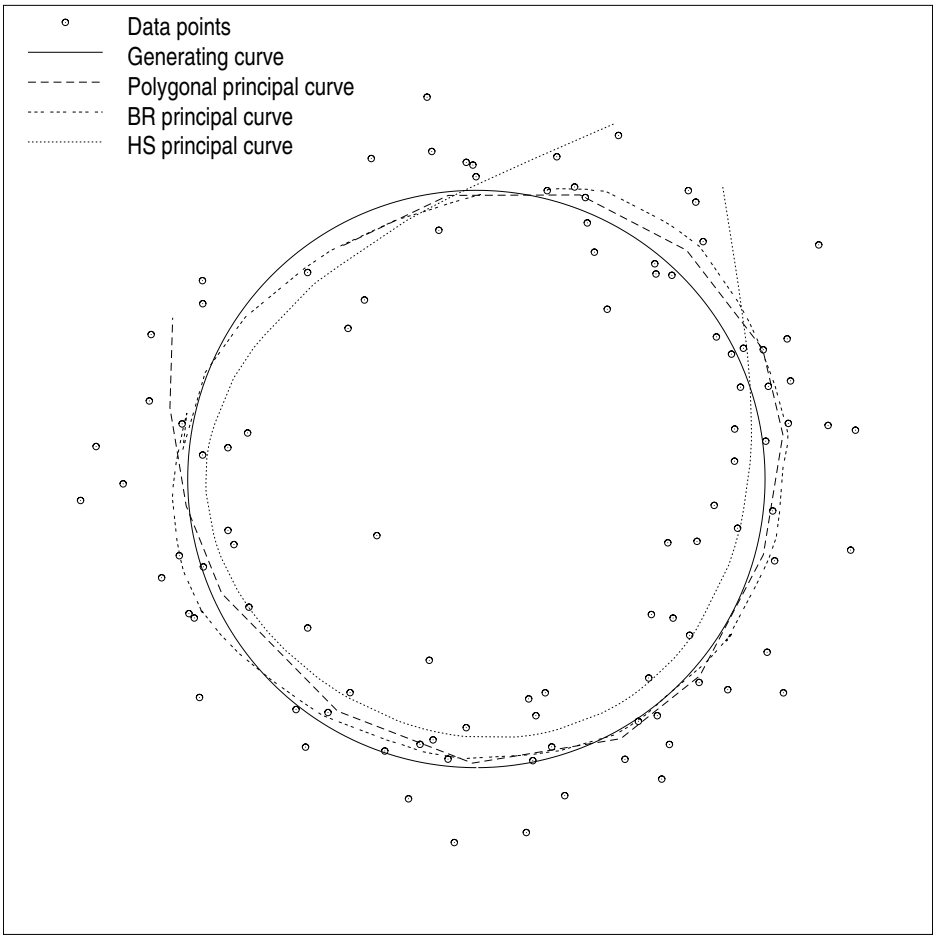

Figure 14: The circle example. The BR and the polygonal line algorithms show less bias than the HS algorithm.

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.

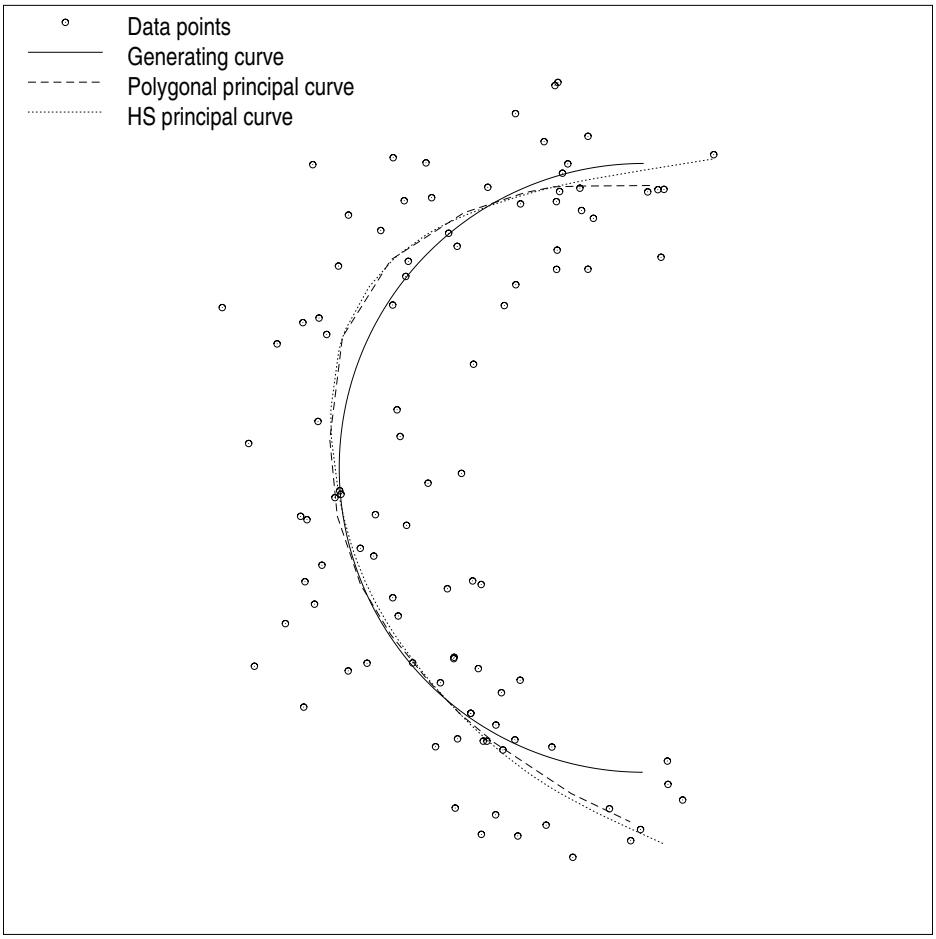

Figure 15: The half circle example. The HS and the polygonal line algorithms produce similar curves.

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

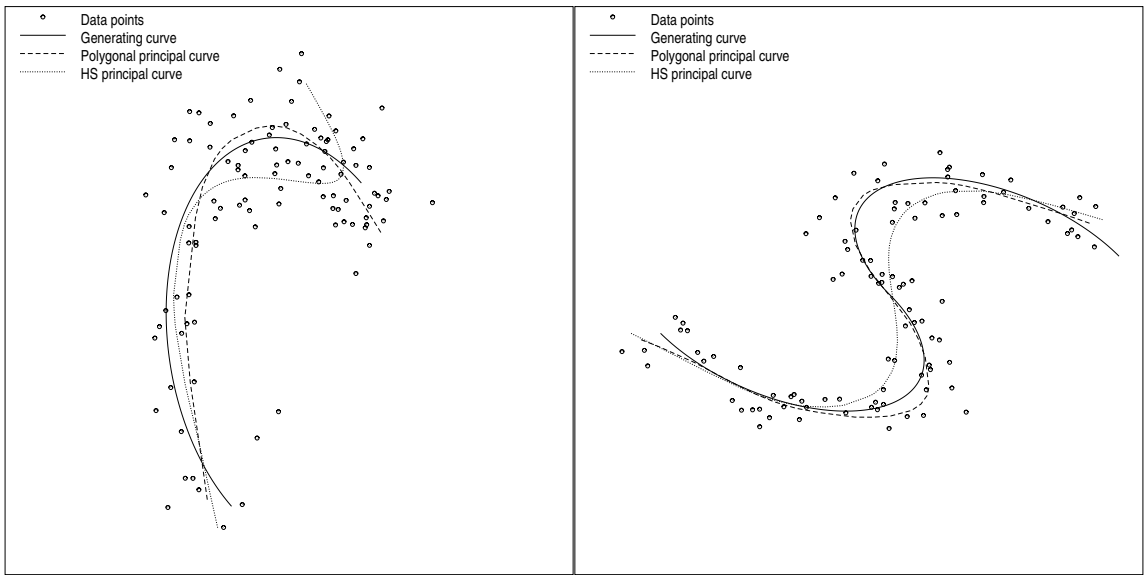

Figure 16: Transformed data sets. The polygonal line algorithm still follows fairly closely the "distorted" shapes.

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

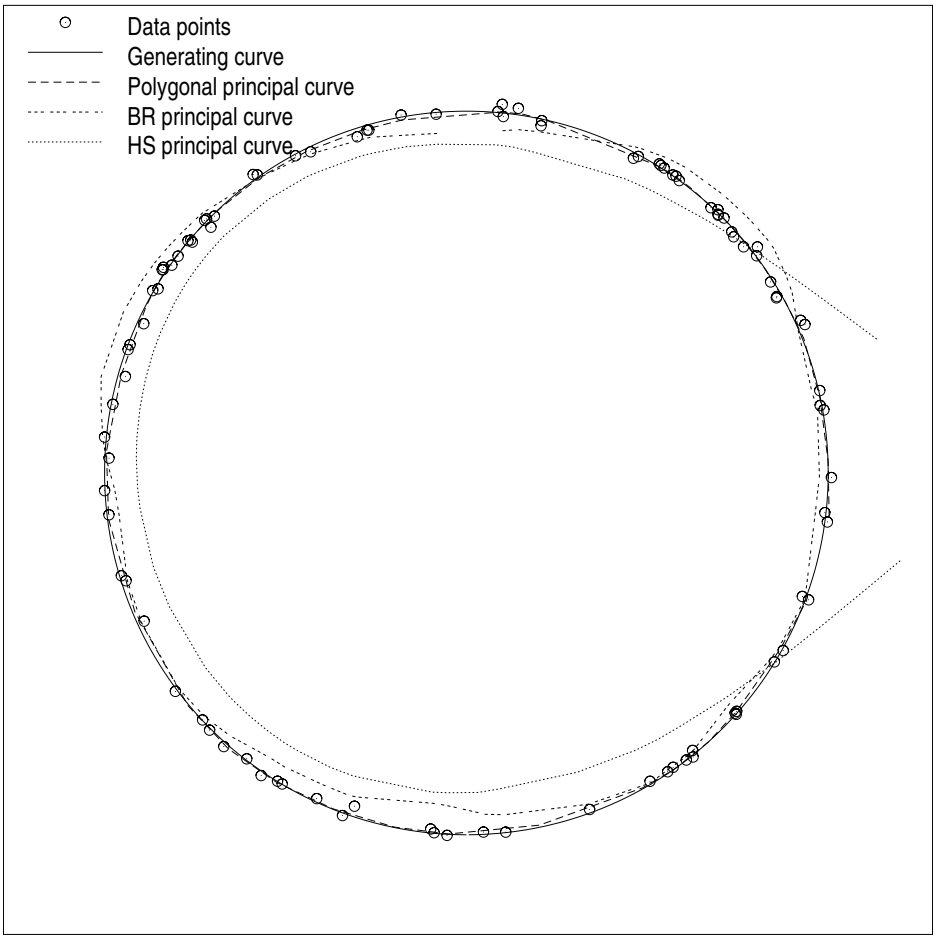

Figure 17: Small noise variance. The polygonal line algorithm follows the generating curve more closely than the HS and the BR algorithms.

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

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

where $\mathbf{f}$ is parameterized by its arc length. The measurements were then averaged over the experiments to obtain $\overline{r}_{n,\sigma}$ 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 $\overline{r}_{n,\sigma}$ and $\overline{RMSE}_{n,\sigma}$ to the theoretical values obtained by HS,

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

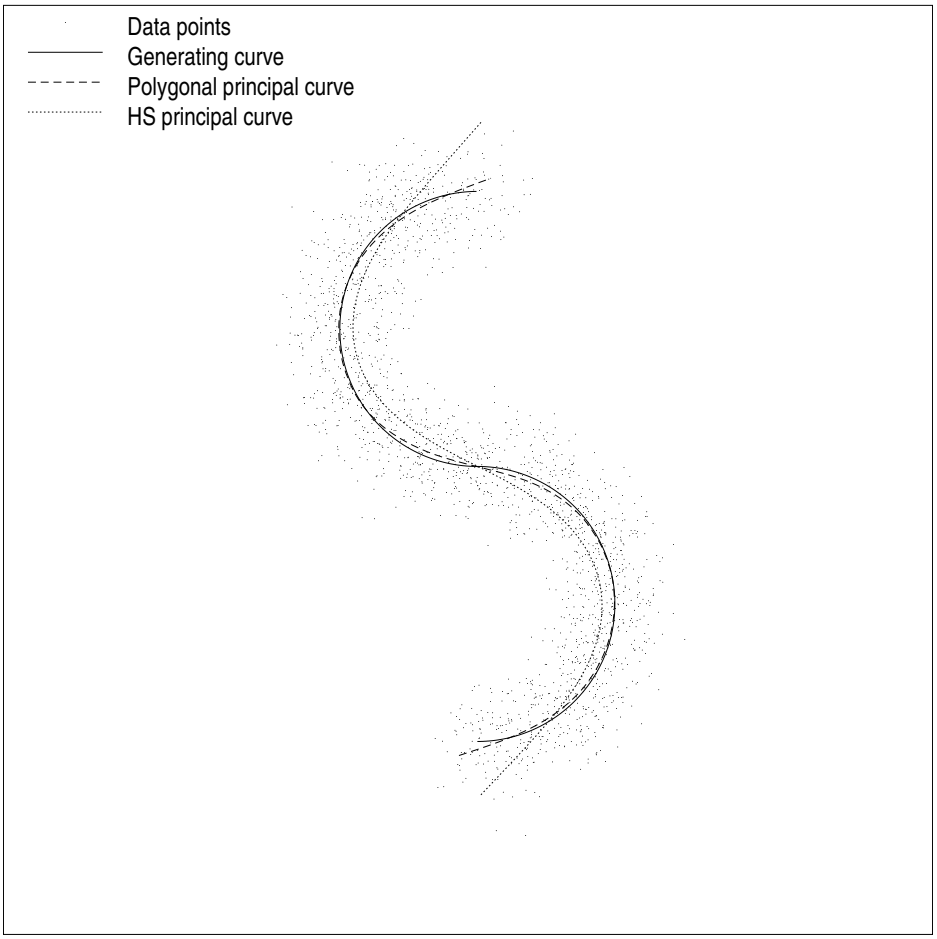

Figure 18: Large sample size. The curve produced by the polygonal line algorithm is nearly indistinguishable from the generating curve.

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.

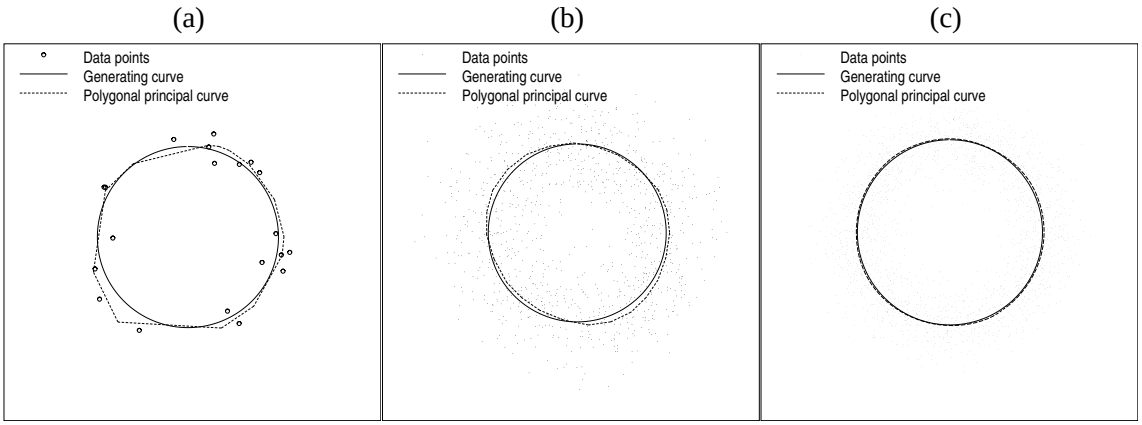

Figure 19: Sample runs for the quantitative analysis. (a) $n = 20$, $\sigma = 0.1$. (b) $n = 1000$, $\sigma = 0.3$. (c) $n = 10000$, $\sigma = 0.2$.

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.

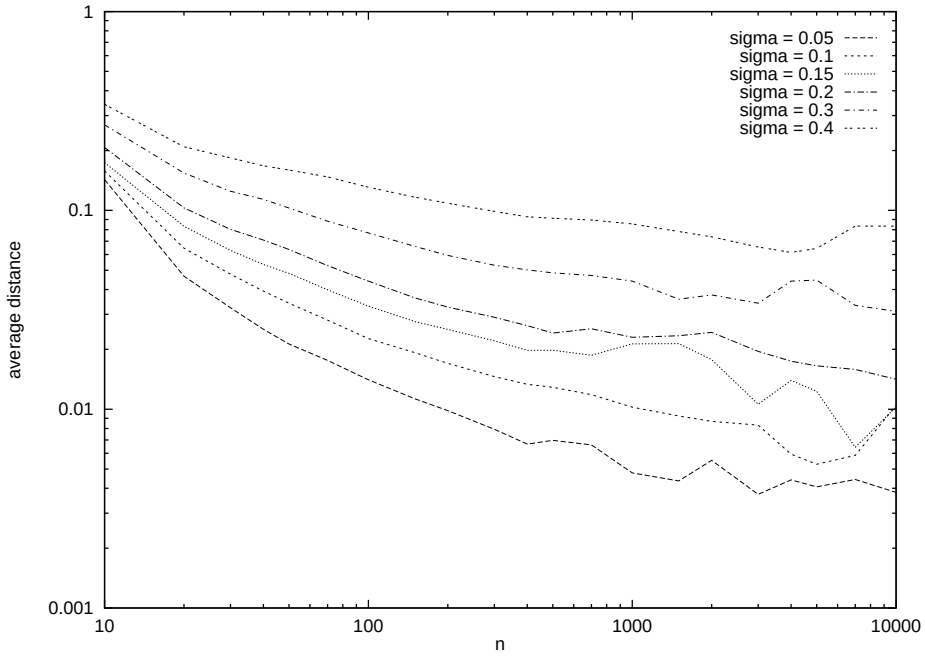

Figure 20: The average distance $\overline{\delta}_{n,\sigma}$ of the generating curve and the polygonal principal curve in terms of σ and n. The approximation improves as the data size grows, and as the variance of the noise decreases.

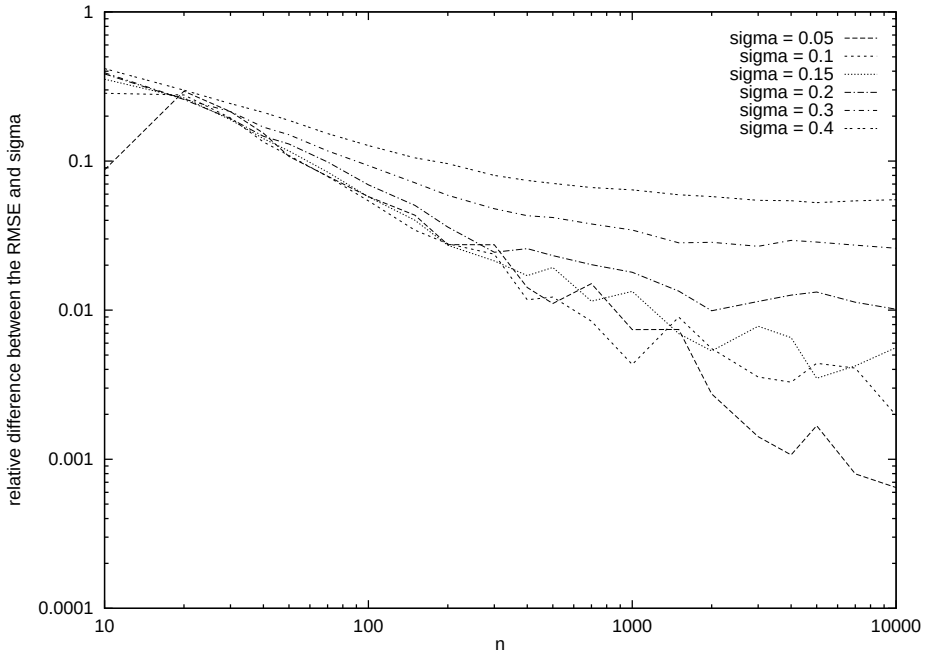

Figure 21: The relative difference $\overline{\varepsilon}_{n,\sigma}$ between the standard deviation of the noise σ and the measured RMSE.

5.2.3 Failure Modes

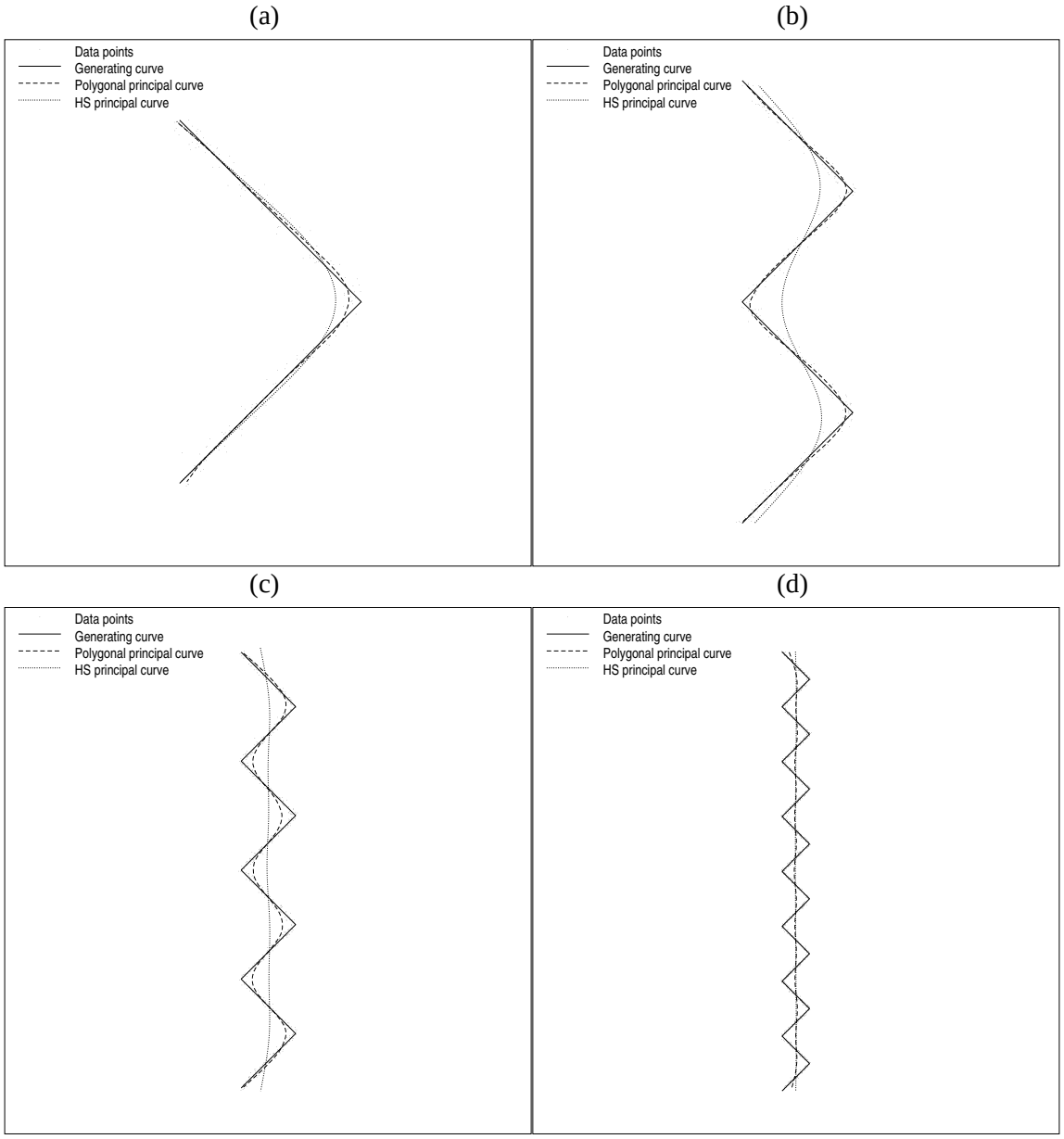

Figure 22: Abrupt changes in the direction of the generating curve. The polygonal line algorithm oversmoothes the principal curve as the number of direction changes increases.

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 penalty factor λ_p is proportional to the number of line segments, the penalty relatively increases as the number of 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).

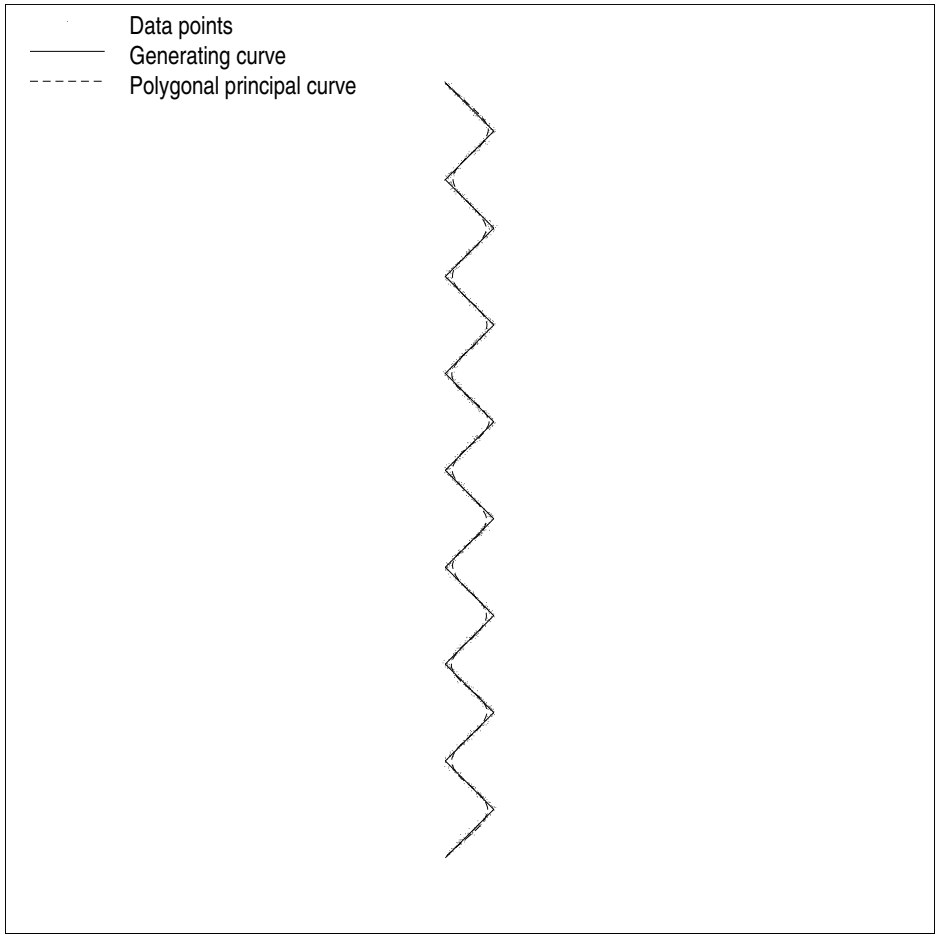

Figure 23: The polygonal principal curve follows the zig-zagging generating curve closely if the penalty coeffi cient is decreased.

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.

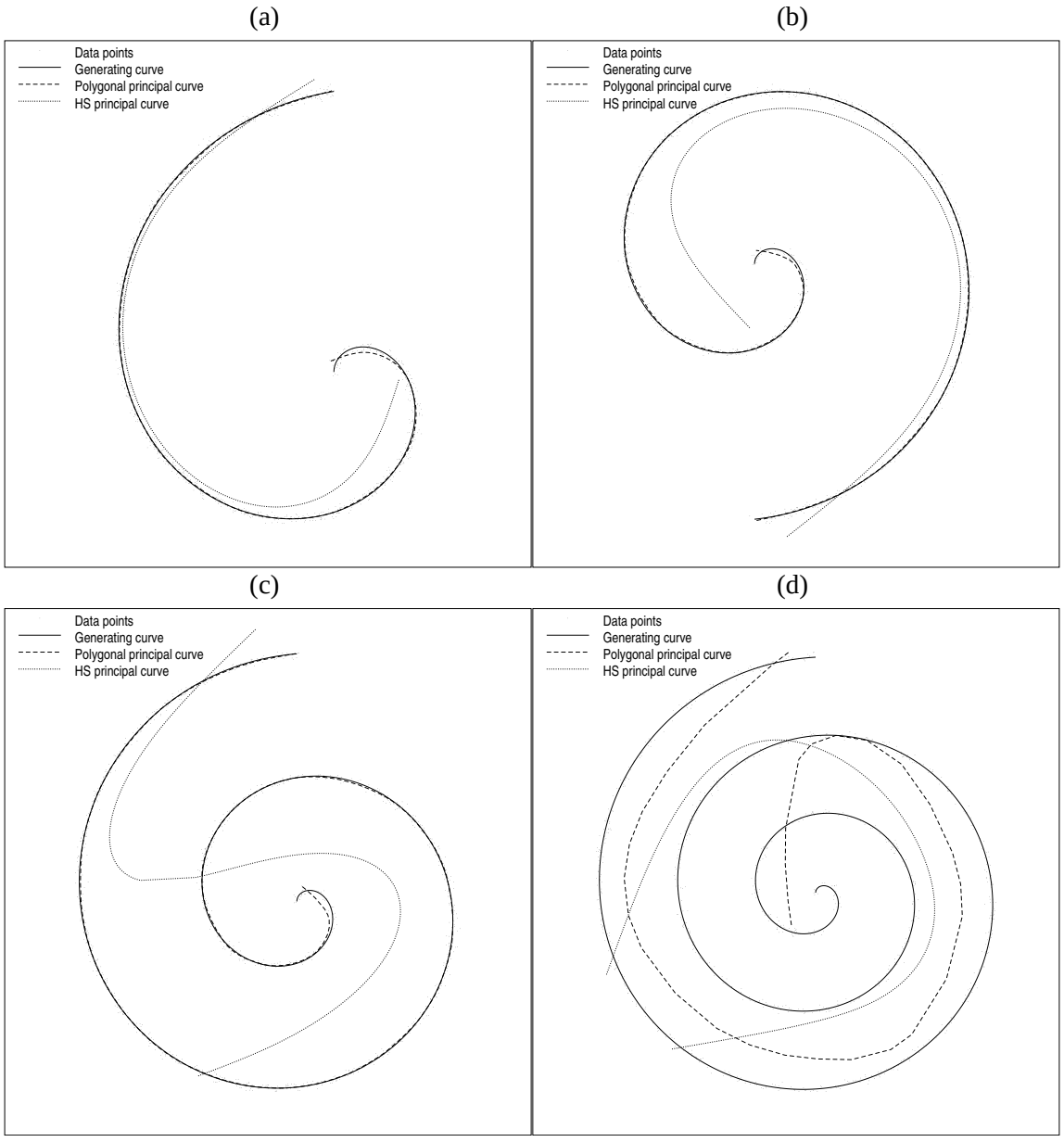

Figure 24: Spiral-shaped generating curves. The polygonal line algorithm fails to find the generating curve as the length of the spiral is increased.

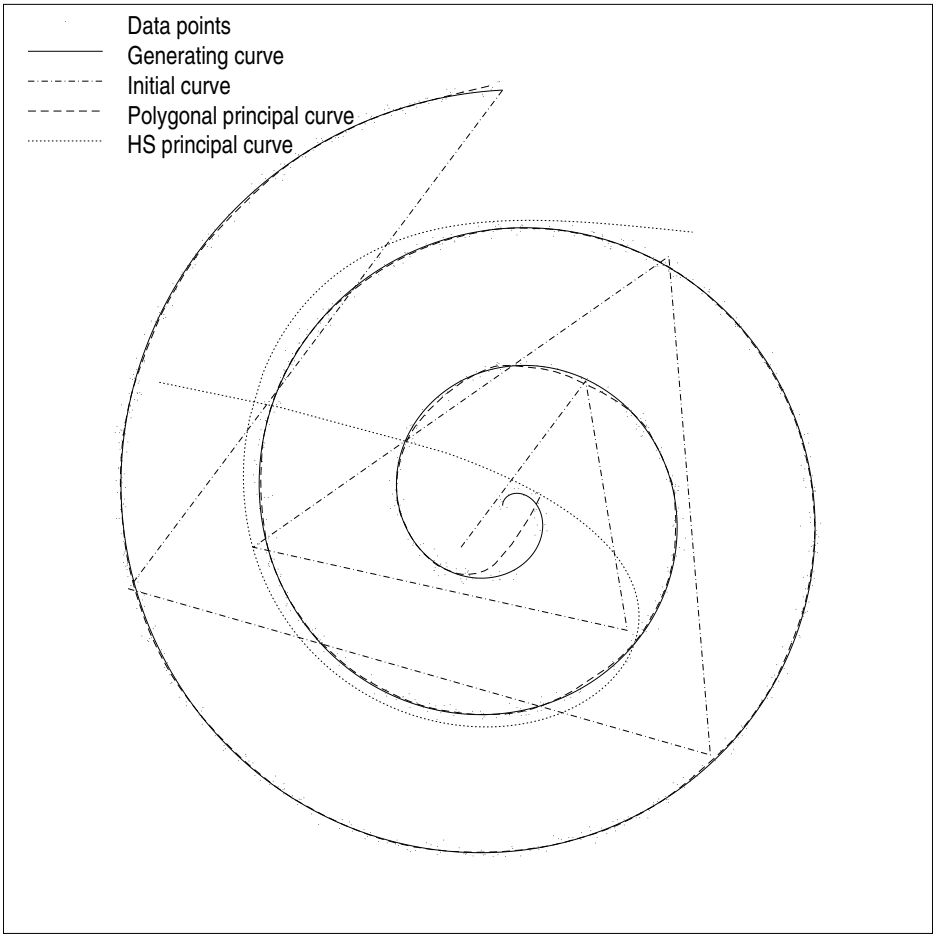

Figure 25: The performance of the polygonal line algorithm improves signifi cantly if the global structure of the generating curve is captured in the initialization step.