Principal Curves in High Dimensions

Alon Milchgrub

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Marina Meilă, Chair

Kevin Jamieson

Anna Karlin

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Principal Curves in High Dimensions

Alon Milchgrub

Chair of the Supervisory Committee:
Professor Marina Meilă
Department of Statistics

Principal curves are curves passing through the "center" of a dataset and generalize the notion of principal components to non-linear curves, thus providing more meaningful insights to the structure of the data. SCMS is an elegant and robust method for recovering the principal curves of a dataset. However, its dependence on evaluating the Hessian of the density function at every step and the need for an abundance of probes make it prohibitively computationally intensive when dealing with large and high-dimensional data. In this thesis, we present L-SCMS, an approximation algorithm for SCMS which reduces the computational complexity from n^2 to n, where n is the dimensionality of the data. We also present MorsePCS, an extension to SCMS that provides with additional information on the structure of the data, while reducing the number of steps required. Finally, we provide empirical evaluation of our MorsePCS, indicating that the runtime guarantees do realize in practice, while maintaining the quality of the curves produced.

TABLE OF CONTENTS

Page

List of Figures . . . iii

List of Algorithms . . . iv

Chapter 1: Introduction . . . 1
- 1.1 Outline of the dissertation . . . 3

Chapter 2: Background . . . 4
- 2.1 Principal Curves . . . 4
- 2.2 Kernel Density Estimators (KDEs) . . . 5
- 2.3 The Gaussian KDE and its derivatives . . . 7
- 2.4 The Mean Shift algorithm . . . 8
- 2.5 The SCMS algorithm for finding density ridges . . . 9
- 2.6 First idea: approximate KDE computation . . . 11

Chapter 3: L-SCMS - SCMS with low-rank Hessian approximation . . . 12
- 3.1 The L-BFGS method . . . 12
- 3.2 Second idea: SCMS using the L-BFGS approximation . . . 14
- 3.3 Third idea: a more precise LNC approximation . . . 15

Chapter 4: From a single ridge point to an entire ridge . . . 18
- 4.1 The original Principal Curves algorithm . . . 18
- 4.2 Fourth idea: ridge tracing . . . 19

Chapter 5: Related work . . . 25

Chapter 6: Preprocessing procedures . . . 29
- 6.1 Kernel width and neighborhood radius selection . . . 29
- 6.2 Dimensionality estimation . . . 30
- 6.3 ANN index construction . . . 32

Chapter 7: Experimental evaluation 34
7.1 Evaluation on synthetic data 34
7.2 Evaluation on real data 38
Chapter 8: Conclusion 43
Bibliography 44

LIST OF FIGURES

Figure Number — Page

6.1 Intrinsic dimension estimation for dataset 'O' ($N = 3000$) and various values of n. 32

7.1 The speedup of L-SCMS over SCMS. 37

7.2 The principal curves of 'Toluene'. 40

7.3 The principal curves of 'Chloromethane'. 40

7.4 The principal curves of 'mnist1'. 41

7.5 The principal curves of 'SDSS'. 42

LIST OF ALGORITHMS

Algorithm Number		Page
1	MeanShift step	9
2	SCMS	10
3	LncSubspace	11
4	ApproxLncSubspace0	15
5	ApproxLncSubspace	16
6	OriginalPCS	19
7	RidgeStep	20
8	TraceDirection	21
9	TraceRidge	21
10	MorsePCS	22
11	SparsePCS	23
12	SparseFindRidge	24
13	EstimateDimension	31

Chapter 1
INTRODUCTION

Nowadays, (very) high-dimensional data is prevalent in many fields of science: In computer vision, images of millions of pixels or even videos composed of thousands such images; In natural language processing, documents represented by hundred of thousands of features; In bioinformatics, DNA sequences consisting of millions of base pairs; And in chemistry, molecular data describing the behavior of the molecule as a function of the many interactions between pairs of atoms are just a few examples. However, while in many cases the data representation has many dimensions, the data itself is actually situated on (or close to) a much lower dimensional space or a *manifold*. Intuitively, a manifold is a topological space which behaves locally like a lower dimensional space. A prime example from everyday life is the Earth, which though it is known to be spherical, i.e. a 3-dimensional object, we can treat it locally as a 2-dimensional surface.

The presence of the seemingly high-dimensional data on a lower dimensional manifold means that its intrinsic dimensionality is, in fact, lower and gives rise to the possibility of recovering the lower-dimensional structure to gain insight on the data (e.g. the underlying topology of the data, the key parameters governing the data and their interactions). The intrinsic low-dimensionality also implies the possibility of transforming the data into a more structured lower-dimensional representation, which may accelerate and facilitate additional data analysis tasks, such as clustering or classification, down the line. These are the core objectives of the fields of *manifold learning* and *dimensionality reduction*.

Much of the research in the field (see chapter 5) has been focused dimensionality reduction, i.e. producing a lower-dimensional representation of the original data that is consistent with it in some sense. Indeed, recovering the underlying manifold itself is a complicated task, which may require an exponential amount of data [4]. Instead, one prominent line of research in manifold learning has focused on recovering the underlying lower dimensional structure of the data via so-called *principal curves* associated with it [6] as a surrogate to the manifold. In general, a principal curve (or, in more than one dimension - a principal surface) is a curve which is locally self-consistent in some sense, i.e. it is the collection of fixed points of some (appropriately defined) mapping from the space into itself. [9] define the principal curves of a distribution as the (local) maxima in the local subspace of the curve that are orthogonal to the gradient on the curve . That is, the principal curves are defined as the "ridges" of the distribution. Later, [4] analyzed this definition theoretically and provided upper bounds for the rate of convergence of the ridges to the underlying manifold.

In order to recover the principal curves they defined, [9] also introduced *Space Constrained Mean Shift (SCMS)*, an iterative algorithm that is guaranteed to converge to a point on a principal curve [5]. In essence, SCMS operates by projecting the step of the mode-seeking algorithm Mean Shift, to a subspace defined by eigenvectors of the Hessian at the current position. The main drawback of SCMS is its high computational cost, since it relies on computing the Hessian of the density at each iteration of the algorithm. Another drawback of SCMS is that starting from a single point in space the algorithm converges to a single point of a principal curve. Therefore, in order to trace the principal curves "completely" the algorithm needs to be restarted from many various locations. For large, high-dimensional data sets, where this algorithm would be most useful, its cost is prohibitive. In fact, we are not aware of experiments with the SCMS algorithm on more than a few thousand samples in up to 4 dimensions, except [11] which experimented with data of 50 dimensions but only 400 samples.

We take inspiration from optimization where approximations of the Hessian have been notably used to accelerate the runtime of quasi-Newton methods. Thus, we propose using approximations of Hessian in order to compute the projection step in SCMS. Doing so drastically improves the dependence of the SCMS update on the ambient dimension n from n^2 to n. By indexing the input data in data structure specialized for approximate nearest neighbor retrieval, we can further alleviate the dependence on the number of samples N. In our work we focus on the case of density ridges, i.e. when the dimension of desired manifold is $d = 1$, however the aforementioned results are can be generalized to $d > 1$. Additionally, for the case $d = 1$ we propose a simple variation to SCMS, that can find whole segments of the principal curve with a single probe.

1.1 Outline of the dissertation

In Chapter 2 we present the major concepts used in our work. In particular, we formally define principal curves and describe the original SCMS algorithm proposed by [9]. Chapter 3 presents our core idea, executing SCMS using an approximation of the constrained space and an efficient algorithm implementing it. Chapter 4 presents SPARSEPCS, an extension to SCMS that recovers an entire ridge segment using a single probe by tracing up and down it. Chapter 5 provides an overview of related methods and results. Chapter 6 presents technical procedures used in our experiments and analysis of the data. Chapter 7 presents the experimental evaluation of the performance of both SCMS and SPARSEPCS on synthetic and real data. Finally, Chapter 8 concludes with discussion concludes with summary and future work.

Chapter 2
BACKGROUND

2.1 Principal Curves

Originally, the concept of principal curves was introduced by [6] as an attempt to generalize the notion of principal components from lines to curves. Given a probability function $\mathbb{P}(X)$, [6] define a principal curve as curve that is *self-consistent* with respect to an expectation conditioned on the curve itself. Formally, let $\gamma : \mathbb{R} \to \mathbb{R}^n$ be a curve (that does not intersect with itself), γ is said to be self-consistent if $\mathbb{E}[X \,|\, \mathrm{proj}_\gamma(X) = y] = y$ for almost every y on γ, where $\mathrm{proj}_\gamma(x) = \gamma\left(\sup\{\lambda : \|x - \gamma(\lambda)\| = \inf_\mu \|x - \gamma(\mu)\|\}\right)$ is the projection of x onto the curve γ. [6] show that when a principal curve is a line then it is also a principal component of the distribution and thus their definition is indeed a generalization. However, the existence of principal curves was proved only to a limited set of well-behaved probability functions and their recovery proved to be computationally intensive.

Following the work of [6] there were several attempts to redefine principal curves, ultimately leading to the definition of [9], which instead of generalizing the notion of a principal component, generalize the concept of a (local) maximum to capture the notion of a (potentially multi-dimensional) ”ridge” of the probability density function. We will now present the definition of [9] which does not only define principal curves as 1-dimensional objects, but rather more generally d-dimensional principal ”surfaces”. We will later focus on the 1D case, which will be of interest for the rest of this work.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a twice continuously differentiable function. Denote by $\nabla f(x)$ and $\nabla^2 f(x)$ the gradient and Hessian of f at a point $x \in \mathbb{R}^n$, respectively; the eigenvalues of $\nabla^2 f(x)$ are denoted by $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ (with $\lambda_i \neq 0$ for all i), and the corresponding eigenvectors by $v_1, \ldots, v_n$.

Definition 2.1 (Critical Set)**.** For an integer d, $0 \leq d \leq n$, a point $x \in \mathbb{R}^n$ is in the *d-dimensional critical set* $\mathcal{C}^d$ of f if $\nabla f(x)$ is orthogonal to v_i for all $i > d$.

Note that for x to be in $\mathcal{C}^0$, $\nabla f(x)$ is required to be orthogonal to all vectors. Thus, $\mathcal{C}^0$ is set of critical points of f under the standard definition. Also note that it holds that $\mathcal{C}^0 \subseteq \mathcal{C}^1 \subseteq \ldots \subseteq \mathcal{C}^n$.

Definition 2.2 (Principal Set)**.** For an integer d, $0 \leq d \leq n$, a point $x \in \mathbb{R}^n$ is in the *d-dimensional principal set* $\mathcal{P}^d$ of f if $x \in \mathcal{C}^d$ and $\lambda_i < 0$ for all $i > d$.

Thus, we have that x is in $\mathcal{P}^d$ iff x is a local maximum of f in the affine space spanned by $\{v_{d+1}, \ldots, v_n\}$ at x . In particular, $\mathcal{P}^0$ is exactly the set of (local) maxima of f under the standard definition. Also, similarly to the critical sets, it holds that $\mathcal{P}^0 \subseteq \mathcal{P}^1 \subseteq \ldots \subseteq \mathcal{P}^n$.

Let $\mathcal{R}(A)$ denote the column space of matrix A. For any x, we call the subspace $\mathcal{R}([v_1 \ldots v_d])$ the *Least Negative Curvature (LNC)* subspace, and an orthogonal matrix V with column space $\mathcal{R}([v_1 \ldots v_d])$ the LNC basis at x. In particular, $[v_1 \ldots v_d]$ itself is a LNC basis.

In the particular case when $d = 1$, we get that a point x is on the principal curve $\mathcal{P}^1$ if both $\nabla f(x)$ is orthogonal to v_i and $\lambda_i < 0$ for all $i \neq 1$. This also implies that v_1 is a LNC basis. Thus, x is on the principal curve if it is a local maximum of f in all directions except the direction of ∇f, meaning that it is located on a "ridge" of f.

2.2 Kernel Density Estimators (KDEs)

Our goal in this work is to recover the underlying structure of a fixed distribution with a probability density function $p(z)$ by finding its ridges. However, we will adopt the machine learning setting, where we do not have direct access to p, but rather we are given a set of independent and identically distributed (i.i.d.) samples $\mathcal{D} = \{z_1, \ldots, z_N\} \in \mathbb{R}^n$ from p. Therefore, we are required to first produce an estimate of p from the data.

Given a dataset $\mathcal{D} = \{z_1, \ldots, z_N\}$, a basic estimate of the density function would be one where every sample is given a point mass of weight $\frac{1}{N}$:

$$\tilde{p}(z) = \frac{1}{N} \sum_i \delta(z - z_i)$$

where δ is Dirac's Delta function. We can see that it hold that

$$\begin{aligned}\int_{z\in\mathbb{R}^n}\tilde{p}(z)dz &= \int_{z\in\mathbb{R}^n}\frac{1}{N}\sum_i \delta(z-z_i)dz\\ &= \frac{1}{N}\sum_i\int_{z\in\mathbb{R}^n}\delta(z-z_i)dz = \frac{1}{N}\sum_{i=1}^{N}1 = 1\end{aligned}$$

Therefore, $\tilde{p}$ is a valid density estimator and for every sample z_i, we have that the probability of observing z_j under $\tilde{p}$ is

$$\begin{aligned}\mathbb{P}(Z=z_i) = \lim_{\varepsilon\to 0}\int_{z\in B(z_j,\varepsilon)}\tilde{p}(z)dz &= \lim_{\varepsilon\to 0}\int_{z\in B(z_j,\varepsilon)}\frac{1}{N}\sum_i\delta(z-z_i)dz\\ = \frac{1}{N}\sum_i\lim_{\varepsilon\to 0}\int_{z\in B(z_j,\varepsilon)}\delta(z-z_i)dz &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}[z_i=z_j] = \frac{n_j}{N}\end{aligned}$$

where $B(z,r)$ is the n-dimensional (open) ball of radius r around z; $\mathbb{1}[E]$ is the characteristic function taking a value of 1 when the predicate E holds and 0 otherwise; and n_i is the number of times the sample z_i appears in $\mathcal{D}$.

The estimator $\tilde{p}$ is what is known as the maximum likelihood estimator (MLE) as it is the probability distribution has the maximal probability of generating our data $\mathcal{D}$. Note, however, that $\tilde{p}$ is singular as it is concentrated completely on $\mathcal{D}$. Thus, assuming that our data originates from a continuous density function, $\tilde{p}$ overestimates the probability of observing already seen samples by an infinitely greater probability and heavily underestimates the probability of any yet unseen values.

A case such as this, where an estimator is heavily influenced by the sample data is said to be "overfitting" in the machine learning context. Instead, we would like an estimator which is more robust to small perturbations of samples. One method to achieve this is to to "blur" or "smooth" our singular MLE estimate $\tilde{p}$. This can be achieved by computing the convolution of $\tilde{p}$ with some filter or blurring kernel, i.e. a non-negative function $K:\mathbb{R}^n\to r$ with $\int_{\mathbb{R}^n}K=1$. Additionally, in order to control the amount of blurring produced by our kernel, it is convenient to parametrize it by a parameter h known as the *smoothing parameter* or the *kernel bandwidth*. Thus, we obtain a function $K_h(x)=\frac{1}{h^n}K(\frac{x}{h})$. Intuitively, the function K is compressed (along the of x) by a factor h. Thus, note that a factor $\frac{1}{h^n}$ in the definition of the kernel in terms of K_h is required to maintain the condition $\int K_h=1$.

By taking the convolution of $\tilde{p}(x)$ with K we obtain a kernel density estimator (KDE) $\hat{p}$:

$$\begin{aligned}\hat{p}(x) &= (\tilde{p} * K)(x) \\ &= \int_{z \in \mathbb{R}^n} \frac{1}{N} \sum_i \delta(z - z_i) K(x - z) dz \\ &= \frac{1}{N} \sum_i \int_{z \in \mathbb{R}^n} \delta(z - z_i) K(x - z) dz \\ &= \frac{1}{N} \sum_i K_h(x - z_i) = \frac{1}{N h^n} \sum_i K\left(\frac{x - z_i}{h}\right)\end{aligned}$$

In theory, any function meeting the aforementioned conditions can serve as a kernel function, however in practice several functions are commonly used depending on the setting of the problem at hand.

As we do not have access to the true underlying density function p, in what follows, p will instead denote the *estimated* density function for clarity.

2.3 The Gaussian KDE and its derivatives

Even though in theory SCMS can be applied to a more general set of distributions, [9] presents it using a Gaussian KDE to exploit its special form. In our presentation we further assume that all samples have a spherical kernel function K_h with a fixed bandwidth h. That is, $K_h(x) = \frac{1}{h^n} K\left(\frac{x}{h}\right)$ where $K(x) = \frac{1}{(\sqrt{2\pi})^n} \exp\left(-\frac{\|x\|^2}{2}\right)$.

Let the samples be $\mathcal{D} = \{z_1, \ldots, z_N\} \subseteq \mathbb{R}^n$, and let $x \in \mathbb{R}^n$ be an arbitrary point. Denote

$$u_i(x) = \frac{x - z_i}{h^2} \tag{2.1}$$

then, the expression for the spherical Gaussian with kernel width of h (or variance h^2 in all directions) centered at z_i is given by

$$\begin{aligned}c_i(x) \triangleq K_h(x - z_i) &= \frac{1}{h^n} K\left(\frac{x - z_i}{h}\right) \\ &= \frac{1}{(2\pi h^2)^{n/2}} \exp\left(-\frac{\|x - z_i\|^2}{2h^2}\right) \\ &= \frac{1}{(2\pi h^2)^{n/2}} \exp\left(-\frac{h^2}{2}\|u_i\|^2\right)\end{aligned} \tag{2.2}$$

where we drop the dependence of u_i on x for clarity. Then, the KDE for $\mathcal{D}$ is given by:

$$p(x) = \tfrac{1}{N}\sum_{i=1}^{N} K_h(x - z_i) = \tfrac{1}{N}\sum_i c_i \tag{2.3}$$

Let the function f denote the logarithm of p, i.e. $f(x) = \ln p(x)$, on the support of p. Since a monotone transformation leaves the sets $\mathcal{C}^d, \mathcal{P}^d$ invariant [9], the sets $\mathcal{P}^d$ of f are the same as those of p.

Note that $\nabla c_i = -c_i u_i$ and $\nabla u = \frac{I}{h^2}$, where I is the n-dimensional identity matrix. Thus, the derivatives of $f = ln(p)$ can be expressed as

$$\begin{aligned}\nabla f(x) &= \nabla \ln p(x) = \frac{\nabla p(x)}{p(x)} = \frac{\nabla(\frac{1}{N}\sum_i c_i)}{\frac{1}{N}\sum_i c_i}\\ &= \frac{-\frac{1}{N}\sum_i c_i u_i}{\frac{1}{N}\sum_i c_i} = -\frac{\sum c_i u_i}{\sum c_i} \triangleq g\end{aligned} \tag{2.4}$$

$$\begin{aligned}\nabla^2 f(x) &= \nabla^2(\ln p) = \frac{\nabla^2 p(x)}{p(x)} - \frac{\nabla p(x)\nabla p(x)^T}{p(x)^2}\\ &= -\frac{1}{\frac{1}{N}\sum_i c_i}\nabla\left(\frac{1}{N}\sum_i c_i u_i\right) - gg^T\\ &= \frac{1}{\sum_i c_i}\left(\sum_i c_i u_i u_i^T - \sum_i \frac{c_i}{h^2}I\right) - gg^T\\ &= \frac{\sum c_i u_i u_i^T}{\sum c_i} - gg^T - \frac{1}{h^2}I\end{aligned} \tag{2.5}$$

2.4 The Mean Shift algorithm

The Mean Shift algorithm is a clustering algorithm that maps every sample (and more generally every point in the space) to a mode of the kernel density estimate induced by the dataset [3]. Initially, a variable or a *probe* is initialized at the position of some sample (or other point in space). Then, the algorithm iteratively updates the position of the probe until it reaches a fixed point (at a mode). The algorithm's update rule at a given iteration is obtained by solving for x the equation $\nabla f = 0$ in (2.4) with the c_i coefficients held fixed. Thus, we obtain the equation $(\sum_i c_i)x = \sum_i c_i z_i$. By solving this equation we get that a probe at position x, should be moved to $x' = \frac{\sum_i c_i(x) z_i}{\sum_i c_i(x)}$ (we re-introduce the c_i's dependence on x to emphasize that they are fixed). It is obvious that x' is in the convex hull of its "neighbors" z_i. Moreover, it can be easily shown that for the Gaussian KDE the *mean shift step* $s = x' - x$ is a *gradient ascent* step.

Proposition 1. Let $x \in \mathbb{R}^n$ and x' the updated position of a probe at x, then for a Gaussian kernel the mean shift step $s = x' - x$ equals

$$s = h^2 \nabla f(x). \tag{2.6}$$

Proof.

$$\begin{aligned} x' - x &= \frac{1}{\sum_i c_i}\left(\sum_i c_i z_i - x \sum_i c_i\right) \\ &= \frac{1}{\sum_i c_i}\sum_i c_i \cdot (z_i - x) \\ &= \frac{1}{\sum_i c_i}\sum_i c_i \cdot (-h^2 u_i) = h^2 \nabla f(x) \end{aligned}$$

□

Algorithm 1 summarizes the MeanShift step, which is repeatedly applied until converges for a Gaussian kernel.

Algorithm 1 MeanShift step

```
Input: point $x$, data $\mathcal{D}$.
Parameters: kernel width $h$.
    compute $c_i, u_i$ for $i = 1, \ldots, N$ as in (2.1) and (2.2)
    $g \leftarrow \frac{\sum_i c_i u_i}{\sum_i c_i}$
    $s \leftarrow h^2 g$
    Output $s$
```

2.5 *The SCMS algorithm for finding density ridges*

Let $\mathcal{D}$ be a given dataset and p an (estimated) density function of $\mathcal{D}$. [9] introduced the SCMS algorithm, a simple iterative algorithm that updates the position of a probe until it converges to a point in the principal set $\mathcal{P}^d$ of p for any given dimensionality d. In essence, the algorithm operates similarly to mean shift, but instead of applying the mean shift update rule, it projects the update step on the space orthogonal to the LNC subspace. To elaborate, let x_k be the position of the probe on the k-th iteration, then the algorithm executes three main operations (see Algorithm 2). First, it computes the mean-shift step s_k in the direction of $\nabla f(x_k)$, where, as usual, $f(x) = \ln p(x)$. Second, it computes an orthonormal basis $Q \in \mathbb{R}^{n\times d}$ to the LNC subspace W (Algorithm 3) or in the case of $d = 1$ the LNC direction. Finally, the next iterate x_{k+1} is obtained by taking a step equal to $s_k^\perp = (I - QQ^T)s_k$, the projection of s_k onto $W^\perp$, the orthogonal complement of W. The algorithm terminates when the mean shift step $s_k \propto \nabla f$ and the projected step $s_k^\perp$ are approximately perpendicular, implying that the probe has reached $\mathcal{P}^d$.

Algorithm 2 SCMS

```
Input: initial point $x_1$, data $\mathcal{D}$.
Parameters: tolerance $t_{ridge}$.
1: for $k = 1, 2, \dots$ do
2:     $s_k \leftarrow \textsc{MeanShift}(x_k, \mathcal{D})$
3:     $Q \leftarrow \textsc{LncSubspace}(x_k, \mathcal{D})$
4:     $s_k^\perp \leftarrow (I - QQ^T)s_k$
5:     $x_{k+1} \leftarrow x_k + s_k^\perp$
6:     until $\frac{s_k \cdot s_k^\perp}{\|s_k\|\|s_k^\perp\|} \le t_{ridge}$        ▷ $s_k$ is approximately perpendicular to $s_k^\perp$
7: end for
8: Output the last $x_k$
```

The computational complexity of SCMS For N data points in n dimensions, the computation of u_i, c_i, and the mean shift step s by (2.6) takes $\mathcal{O}(Nn)$ operations for a given position x; evaluating B by (2.5) is $\mathcal{O}(Nn^2)$, and evaluating its LNC basis Q by LncSubspace takes $\mathcal{O}(n^2 d)$ operations; finally, taking a projected step using step 4 in algorithm 2 takes $\mathcal{O}(nd)$ operations (as this involves multiplying a vector with matrices of size $d \times n$). In total, a single iteration of SCMS requires $\mathcal{O}\left(n^2(N + d)\right)$ operations. In

Algorithm 3 LncSubspace

```
Input: a point $x$, data $\mathcal{D}$.
Parameters: principal curve dimensionality $d$.
1: $B \leftarrow \nabla^2 f(x)$ calculated using (2.5)
2: $Q \leftarrow$ The $d$ normalized eigenvectors corresponding to the largest eigenvalues of $B$
                                                        ▷ $Q \in \mathbb{R}^{n \times d}$ is a LNC basis.
3: Output $Q$
```

particular, we see that the operations involving the Hessian dominate the run time of the SCMS algorithm.

So far, the quadratic dependence of the runtime on the ambient dimensionality n has prevented the application of this very useful algorithm to truly large or high-dimensional data sets. The applications we are aware of do not exceed 10^4 samples and about 10 dimensions, except [11] which experimented with data of up to 50 dimensions, but with only 400 samples. Yet, it is the high-dimensional and large data that would most benefit from this elegant dimension reduction and geometric analysis method.

2.6 *First idea: approximate KDE computation*

Note that the influence of a sample on the Gaussian KDE used for this task and its derivatives diminishes rapidly as we get further away from it. Hence, the KDE and its derivatives at a point x can be approximated by only considering samples which are at most a fixed distance r_{comp} from x or, alternatively, using only the k nearest samples to x. Thus, with careful indexing of the data so that the nearest neighbors of an arbitrary point x can be efficiently retrieved, it is possible to remove the dependence on the sample size N from MeanShift or SCMS-type algorithms.

The challenge that remains is to reduce the quadratic dependence on the dimension n to linear dependence. The next section explains how this can be achieved.

Chapter 3
L-SCMS - SCMS WITH LOW-RANK HESSIAN APPROXIMATION

As we have seen in section 2, the computational complexity of a single iteration of SCMS is $\mathcal{O}\left(n^2(N+d)\right)$, where the quadratic dependence on the ambient dimensionality n stems from the need to compute the Hessian $\nabla^2 f(x)$ at the probe, an $n \times n$ matrix.

In section 2 we have already suggested eliminating the dependence of the runtime on the number of samples N by using a sophisticated nearest neighbors method. In order to reduce the dependence of the runtime to sub-quadratic in n, we turn to numerical optimization, a field where this problem has been extensively studied. Our idea is to approximate the LNC subspace spanned by $v_1(x), \cdots, v_d(x)$ by approximating the Hessian, thus *bypassing the evaluation of the exact Hessian.*

3.1 The **L-BFGS** *method*

Matrix secant methods, also known as quasi-Newton mehods, evaluate the Hessian of a function iteratively, by rank 1 updates, to an $n \times n$ positive definite matrix, amounting to $\mathcal{O}(n^2)$ operations per iteration. Of these, the best studied and empirically most effective is the Boyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [8]. In practice, a good approximation to the Hessian is reached in a number of iterations much smaller than n. This has led to the more practical *Limited Memory BFGS (L-BFGS)* method, in which the Hessian is approximated by a matrix of rank $2m$, with $m \ll n$. In a standard quasi-Newton method, the inputs to the L-BFGS algorithm are $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1})$. The algorithm maintains the following $n \times m$ matrices

$$S_k = [s_{k-m} \ \dots \ s_{k-1}] \tag{3.1}$$

$$Y_k = [y_{k-m} \ \dots \ y_{k-1}] \tag{3.2}$$

Given $\nabla f(x_k)$, updating these matrices takes $\mathcal{O}(mn)$ operations on every iteration. The variable m is a parameter of the L-BFGS algorithm called the *memory size*. This needs to be larger than principal set dimensionality d; empirically, in optimization contexts, it was found that $m \approx 0.05n$ is a recommended accuracy speed trade-off. In all of our experiments we use $m = 5$ for $d = 1$.

Using these matrices one can compute a low rank approximation of the inverse Hessian $H_k \approx \left[\nabla^2 f(x_k)\right]^{-1}$ can be computed directly using the following formulas [1]:

$$(R_k)_{i,j} = \begin{cases} s_{k-m-1+i}^T y_{k-m-1+j} & \text{if } i \leq j, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

$$D_k = \operatorname{diag}\left\{s_{k-m}^T y_{k-m}, \ldots, s_{k-1}^T y_{k-1}\right\} \tag{3.4}$$

$$\gamma_k = s_{k-1}^T y_{k-1} / \|y_{k-1}\|^2 \tag{3.5}$$

$$\Psi_k = \begin{bmatrix} S_k & \gamma_k Y_k \end{bmatrix} \in \mathbb{R}^{n \times 2m} \tag{3.6}$$

$$M_k = \begin{bmatrix} R_k^{-T}(D_k + \gamma_k Y_k^T Y_k) R^{-1} & -R_k^{-T} \\ -R_k^{-1} & 0 \end{bmatrix} \in \mathbb{R}^{2m \times 2m} \tag{3.7}$$

$$H_k = \gamma_k I + \Psi_k M_k \Psi_k^T \tag{3.8}$$

Alternatively, a low-rank approximation of the Hessian B_k can be computed directly using the following formulas [1]:

$$(L_k)_{i,j} = \begin{cases} s_{k-m-1+i}^T y_{k-m-1+j} & \text{if } i > j, \\ 0 & \text{otherwise.} \end{cases} \tag{3.9}$$

$$\delta_k = 1/\gamma_k \tag{3.10}$$

$$\Phi_k = \begin{bmatrix} \delta_k S_k & Y_k \end{bmatrix} = \delta_k \Psi_k \in \mathbb{R}^{n \times 2m} \tag{3.11}$$

$$M'_k = \begin{bmatrix} \delta_k S_k^T S_k & L_k \\ L_k^T & -D_k \end{bmatrix} \in \mathbb{R}^{2m \times 2m} \tag{3.12}$$

$$B_k = \delta_k I - \Phi_k (M'_k)^{-1} \Phi_k^T \tag{3.13}$$

Empirically, L-BFGS algorithms converge fast to a low-rank approximation of the (inverse) Hessian [1].

3.2 Second idea: SCMS *using the* L-BFGS *approximation*

In order to to mitigate the computational cost of evaluation $\nabla^2 f$ we suggest using the L-BFGS approximation B_k to efficiently estimate the LNC at x_k. That is, our proposal is to use equation (3.13) to obtain an approximation to the LNC subspace, replacing step 1 of algorithm LncSubspace. Note that the computation of B_k (which takes $\mathcal{O}(mn^2)$) is not necessary, as will be explained shortly. Hence, by returning just the matrices Ψ_k and M_k the runtime becomes $\mathcal{O}(nm + m^2)$, or $\mathcal{O}(nm)$ when $m \ll n$ and constant.

The parameter m of the L-BFGS algorithm is the *memory size*. This needs to be larger than d; empirically, in optimization contexts, it was found that $m \approx 0.05n$ is a recommended accuracy speed trade-off. In all of our experiments we use $m = 5$ for $d = 1$.

We now show how to obtain the LNC basis Q. Remember that the LNC is the span of the d eigenvectors corresponding to the d largest eigenvalues of the Hessian. Thus, since $\nabla^2 f \approx \delta I - \Phi(M')^{-1}\Phi^T$, these eigenvectors can be approximated by the eigenvectors of $\Phi(M')^{-1}\Phi^T$ corresponding to the d *smallest* eigenvalues (note the sign flip due to the minus sign in the approximation for the Hessian). Furthermore, let $W \in \mathbb{R}^{n\times 2m}$ be a matrix with orthonormal columns and $R \in \mathbb{R}^{2m\times 2m}$ be an upper triangular matrix, such that $\Phi = WR$. It can be verified that for every eigenvector q of $R(M')^{-1}R^T \in \mathbb{R}^{2m\times 2m}$ corresponding to eigenvalue λ, Wq is an eigenvector of $\Phi(M')^{-1}\Phi^T$ corresponding to eigenvalue λ as well. Thus, it is sufficient to find the eigenvectors corresponding to the d smallest eigenvalues of $R(M')^{-1}R^T$, as is done in ApproxLncSubspace0 (algorithm 4). We call the SCMS variant that uses algorithm ApproxLncSubspace0 instead of LncSubspace L-SCMS0.

Computation requirements Assuming that $\nabla f(x_k)$ is given, the L-BFGS update and the computation of the matrices Ψ_k and M'_k require $\mathcal{O}(nm + m^2)$ operations as can be seen from the expression defining these objects. Performing the QR decomposition and eigendecomposition in ApproxLncSubspace0 require $\mathcal{O}(m^2 n + m^3)$ operations. Hence, in general, the number of operations required in every iteration is $\mathcal{O}(m^2 n + m^3)$ assuming that $\nabla f(x_k)$ is given. In our case of a Gaussian KDE, computing the gradient takes $\mathcal{O}(nN)$ operations and therefore the total runtime complexity of each iteration is $\mathcal{O}(m^2 n + m^3 + nN)$.

Algorithm 4 ApproxLncSubspace0

```
Input: a point $x$, data $\mathcal{D}$.
Parameters: principal curve dimensionality $d$.
    Compute $\Phi_k$ using (3.11)
    Compute $M'_k$ using (3.12)
    $W,\ R \leftarrow$ QR decomposition of $\Phi_k$                    $\triangleright\ W \in \mathbb{R}^{n\times 2m},\ R \in \mathbb{R}^{2m\times 2m}$
    $A \leftarrow R(M'_k)^{-1}R^T$
    $\widetilde{Q} \leftarrow$ The $d$ eigenvectors corresponding to the *smallest* eigenvalues of $A$
                                                                      $\triangleright\ \widetilde{Q} \in \mathbb{R}^{2m\times d}$
    Output $Q = W\widetilde{Q}$
```

L-BFGS must be initialized with full-rank matrices $S, Y \in \mathbb{R}^{n\times 2m}$. Also, since every iteration depends on the previous iterate, x_0 must be defined for the first iteration. The initialization method finds the $m+1$ nearest neighbors of x_1 in $\mathcal{D}$ (excluding x_1), denoted $z_1, \dots z_{m+1}$, computes $\nabla f(z_j)$ for $j = 1, 2, \dots m+1$, then sets $x_0 = z_1$ and $s_j = z_1 - z_{j+1}$, $y_j = \nabla f(z_1) - \nabla f(z_{j+1})$ for $j = 1, 2, \dots m$. Note that this step is equivalent to $m+1$ gradient computations, hence it adds a significant constant to the run time of L-SCMS0 (as well as any L-BFGS-based algorithm).

3.3 Third idea: a more precise LNC approximation

The L-SCMS0 algorithm reduces the time per iteration, by introducing an approximation in the LNC estimation. In experiments, the approximation has little effect on the number of iterations to reach a density ridge. However, the less desirable effect of the approximation is that due to residual error L-SCMS0 fails to detect when the ridge is reached, thus adding unnecessary iterations.

Assume for the sake of intuition that $d = 1$. L-SCMS0 computes the LNC direction of the *approximate Hessian*. Our second idea is to replace this with the LNC direction of the *exact Hessian*, *restricted* to the $2m$-subspace obtained by L-BFGS. This is done efficiently by ApproxLncSubspace as shown in algorithm 5.

Algorithm 5 ApproxLncSubspace

```
Input: a point $x$, data $\mathcal{D}$.
Parameters: principal curve dimensionality $d$.
    Compute $\Psi_k$ using (3.6)
    $W, R \leftarrow$ QR decomposition of $\Psi_k$          ▷ $W \in \mathbb{R}^{n\times 2m}$, $R \in \mathbb{R}^{2m\times 2m}$
    Compute $g = \nabla f(x)$ using (2.4)
    $\widetilde{g} \leftarrow W^T g$
    Compute $U = \left[u_1 \cdots u_N\right]$ using (2.1)          ▷ $U \in \mathbb{R}^{n\times N}$
    $\widetilde{U} \leftarrow W^T U$
    $\widetilde{B} \leftarrow \frac{1}{\sum c_i}\sum c_i \widetilde{u}_i \widetilde{u}_i^T - \widetilde{g}\widetilde{g}^T - \frac{1}{h^2} I$          ▷ Compare with Eq. (2.5)
    $\widetilde{Q} \leftarrow$ The $d$ eigenvectors corresponding to the *smallest* eigenvalues of $\widetilde{B}$
                                                              ▷ $\widetilde{Q} \in \mathbb{R}^{2m\times d}$
    Output $Q = W\widetilde{Q}$
```

Proposition 2. Let p be defined as in section 2 with kernel width $h > 0$:

$$p(x) = \frac{1}{N}\sum_{i=1}^{n} c_i(x) = \frac{1}{N(2\pi h^2)^{n/2}}\sum_{i=1}^{n}\exp\left(-\frac{h^2}{2}\|u_i\|^2\right)$$

and let $f = \ln(p)$, $g = \nabla f(x)$, $B = \nabla^2 f(x)$ for some $x \in \mathbb{R}^n$ and $W \in \mathbb{R}^{n\times 2m}$ be a matrix with orthonormal columns. Denote by $Q \in \mathbb{R}^{n\times d}$ the matrix with orthonormal columns in $\mathcal{R}(W)$ that maximizes $\operatorname{trace}\left(Q^T B Q\right)$. Then, Q can be obtained by algorithm ApproxLncSubspace in $\mathcal{O}\left(mnN + m^2N + m^2n + m^3\right)$ operations.

Proof. Any vector $q \in \mathcal{R}(W)$ can be written as $q = W\xi$, therefore $Q = VX$, where the columns of $X \in \mathbb{R}^{2m\times d}$ are orthonormal. Then, we can find X as the solution of

$$\max_{\substack{X\in\mathbb{R}^{2m\times d}\\ X \text{ orthogonal}}} \operatorname{trace}\left(X^T W^T B W X\right). \tag{3.14}$$

Hence, X is given by the d eigenvectors of $A = W^T BW$ corresponding to the highest eigenvalues, which can be obtained in $\mathcal{O}(dm^2)$ operations.

We now show that in the case of a Gaussian kernel, A can be obtained without computing

B. Denote

$$\begin{aligned}
\widetilde{g} &= W^T g \\
U &= [u_1 \cdots u_N] \\
\widetilde{U} &= W^T U = \left[W^T u_1 \cdots W^T u_N\right] = [\widetilde{u}_1 \cdots \widetilde{u}_N] \in \mathbb{R}^{2m \times N}
\end{aligned}$$

Note that given g and U, $\widetilde{g}$ and $\widetilde{U}$ can be computed in $\mathcal{O}(mnN)$ operations. From the expression of B given in (2.5) we have that,

$$\begin{aligned}
A &= W^T B W \\
&= W^T \left(\frac{1}{\sum c_i} \sum c_i u_i u_i^T - g g^T - \frac{1}{h^2} I \right) W \\
&= \frac{1}{\sum c_i} \sum c_i W^T u_i u_i^T W - W^T g g^T W - \frac{1}{h^2} W^T W \\
&= \frac{1}{\sum c_i} \sum c_i (W^T u_i)(W^T u_i)^T - (W^T g)(W^T g)^T - \frac{1}{h^2} I \\
&= \frac{1}{\sum c_i} \sum c_i \widetilde{u}_i \widetilde{u}_i^T - \widetilde{g} \widetilde{g}^T - \frac{1}{h^2} I
\end{aligned} \tag{3.15}$$

Therefore, if $\widetilde{g}$, $\widetilde{U}$ and c_i for $i = 1, \ldots, N$ are given, the matrix A can be obtained using (3.15) with $\mathcal{O}(m^2 N)$ operations. Performing the QR decomposition and eigendecomposition in ApproxLncSubspace require $\mathcal{O}(m^2 n + m^3)$ operations. Thus, the total runtime complexity of each iteration of ApproxLncSubspace is $\mathcal{O}\left(mnN + m^2 N + m^2 n + m^3\right)$. Moreover, whenever N can be reduced to a bounded number of neighbors of x, the dependence in N disappears as well.

$\square$

Chapter 4

FROM A SINGLE RIDGE POINT TO AN ENTIRE RIDGE

So far, we have shown how to reduce the run time of a single step of the SCMS algorithm from quadratic to linear in the data dimension n, by using ideas from second order optimization. Now we will examine the outer iterations needed to find, first, a single principal curve, and next, all the principal curves supported by $\mathcal{D}$. In this section, we restrict ourselves to the case $d = 1$. The ideas apply both to our accelerated implementation L-SCMS and to the original SCMS. In what follows, we will use the term FindRidge to refer to a general ridge recovery algorithm, i.e. an algorithm that maps a given point $x \in \mathbb{R}^n$ to a point on the density ridge. We will use the term LncSubspace to generally refer to the basic LncSubspace algorithm (algorithm 3) or to any of its variants.

4.1 The original Principal Curves algorithm

We must start by clarifying what we mean by "finding one (all) principal curves". Given that the density p is a KDE from a finite dataset $\mathcal{D}$, the set $\mathcal{P}^1$ of all principal curves implicitly corresponds to the current density estimate p. The topology of $\mathcal{P}^1$ will clearly depend on the choice of kernel width h, but this is out of the scope of the present work. Once p is fixed, $\mathcal{P}^1$ will be in general a union of manifolds of dimension 0 and 1, of which a non-parametric algorithm such as SCMS can only find a finite subset. The OriginalPCS (algorithm 6) operates by iteratively recovering a single ridge point at a time by starting a ridge recovery algorithm FindRidge from every point in our dataset.

The advantage of OriginalPCS is that it does not make any assumptions about the topology of the density ridges, but the disadvantage is that the set $\mathcal{P}^1_{orig}$ contains no topological information about $\mathcal{P}^1$. From the computational point of view, initializing at all points in the data set is extremely expensive when N is large. A typical remedy to this problem is to initialize at a subset of $\mathcal{D}$, but this reduces the number of points in $\mathcal{P}^1$ that

Algorithm 6 OriginalPCS

```
Input: data $\mathcal{D}$.
  for $i = 1 : N$ do
      $x_i \leftarrow$ FindRidge($z_i, \mathcal{D}$)
  end for
  Output $\mathcal{P}^1_{orig} = \{x_i\}_{i=1}^N$
```

will be returned, and it increases the chance of missing an ridge entirely.

Here we propose a simple heuristic that will augment the set $\mathcal{P}^1_{orig}$ with topological information, and in the same time it will drastically reduce the number of restarts of the algorithm while exploring the density landscape

4.2 Fourth idea: ridge tracing

OriginalPCS restarts a ridge recovery algorithm at each data point (or on a mesh), and stops when the ridge is reached. The core of our idea is to *follow the ridge* from the ridge points recovered by OriginalPCS. Evidently, since on the ridge the gradient is aligned with the ridge direction, following the ridge amounts to gradient ascent or descent. Following a discovered ridge makes optimal use of the previous information (the ridge) instead of agnostically restarting the algorithm. It also provides topological information to which OriginalPCS is oblivious.

RidgeStep (algorithm 7) starts from a previously found ridge point and takes a gradient step along the principal curve (either up or down the ridge as defined by the *direction* parameter). Note that in order to compute the parallel step the LNC basis is required, and it can be obtained by any of the methods mentioned previously such as LncSubspace or its approximated variants. Also, note that $s = s^{\perp} + s^{\parallel}$. Thus, given either the parallel step $s^{\parallel}$ or the perpendicular step $s^{\perp}$, the other one can be easily obtained. After stepping along the ridge, the algorithm verifies that the probe is still on the ridge by calling some ridge recovery method FindRidge, as a form of projected gradient ascent.

In order to recover the ridge segment in one direction, TraceDirection (algorithm 8)

Algorithm 7 RidgeStep

```
Input: a ridge point $x$, data $\mathcal{D}$, $direction \in \{\text{UP}, \text{DOWN}\}$
Parameters: momentum decay factor $\alpha$
Maintains: the previous momentum value $m$, initially 0 (see text).
  $s \leftarrow \text{MeanShift}(x, \mathcal{D})$
  $Q \leftarrow \text{LncSubspace}(x, \mathcal{D})$
  $s^{\|} \leftarrow QQ^T s$
  if $direction$ = up then
      $x' \leftarrow x + s^{\|}$
  else
      $x' \leftarrow x - (1-\alpha)s^{\|} + \alpha m$
  end if
  $x' \leftarrow \text{FindRidge}(x', \mathcal{D})$
  $m \leftarrow (1-\alpha) \cdot (x' - x) + \alpha m$
  Output $x'$
```

repeatedly steps along the ridge until either a critical point (generally, either a maximum or a saddle point) is found as observed by the mean-shift step decreasing below the threshold t_{stop} or until the probe reaches a point which is sufficiently close to a previously discovered ridge as characterized by d_{min}, implying that two ridges intersect at a non-critical point.

A maximum is stable for gradient ascent, while a saddle point is unstable. This is confirmed empirically, by the fact that during descent towards saddles the algorithm might not terminate, as the the probe circles around the saddle. Therefore, a momentum term is added to the *descent* steps in RidgeStep to reduce the oscillatory behavior.

The distance $d(x, S)$ between a point x and a ridge (segment) $S = \{x_i\}_{i=1}^r$ is the minimum distance between x and the piecewise-linear curve obtained by connecting a line between every x_i and the consecutive x_{i+1}. Explicitly,

$$d(x, S) = \min_{\substack{1 \le i < r \\ 0 \le \lambda \le 1}} \left\| x - \left[(1-\lambda)x_i + \lambda x_{i+1}\right] \right\| \tag{4.1}$$

Naturally, the distance $d(x, \mathcal{P})$ between x and a *set* of ridge segments $\mathcal{P}$ is the minimal

Algorithm 8 TraceDirection

```
Input: initial ridge point $x$, data $\mathcal{D}$, $direction \in \{\textsc{up}, \textsc{down}\}$, the set of previously discovered ridges $\mathcal{P}$.
Parameters: thresholds $t_{stop}$, $d_{min}$.
  $x_1 \leftarrow x$
  for $k = 1, 2, \ldots$ do
      $s \leftarrow \textsc{MeanShift}(x_k, \mathcal{D})$
      until $\|s\| \leq t_{stop}$ or $d(x_k, \mathcal{P}) \leq d_{min}$
      $x_{k+1} \leftarrow \textsc{RidgeStep}(x_k, \mathcal{D}, direction)$
  end for
  Output current unidirectional ridge segment $\{x_i\}_{i=1}^{k}$
```

distance to any ridge in $\mathcal{P}$, that is

$$d(x, \mathcal{P}) = \min_{S \in \mathcal{P}} d(x, S) \tag{4.2}$$

Algorithm 9 TraceRidge

```
Input: initial ridge points $x$, data $\mathcal{D}$, the set of previously discovered ridges $\mathcal{P}$.
  $S^{up} \leftarrow \textsc{TraceDirection}(x, \mathcal{D}, \textsc{up}, \mathcal{P})$
  $S^{down} \leftarrow \textsc{TraceDirection}(x, \mathcal{D}, \textsc{down}, \mathcal{P})$
  $S \leftarrow \left[\textsc{Reverse}(R^{down}_{2:end}), R^{up}\right]$
  Output current ridge segment $S$
```

The entire ridge segment stretching from a saddle to a maximum (or intersecting another ridge at either end), is reconstructed by concatenating the two unidirectional segments obtained from the execution of TraceDirection in both directions starting from the same initial ridge point as shown in TraceRidge (algorithm 9).

Finally, MorsePCS (algorithm 10) attempts recovering as many ridges as possible by starting TraceRidge from multiple starting points. Note that in contrast to OriginalPCS which produces a set of points on the ridges, MorsePCS produces as set of *ridge segments*,

Algorithm 10 MorsePCS

```
Input: a set of initial ridge points $\{x_i\}_{i=1}^{r}$, data $\mathcal{D}$.
  $\mathcal{P} \leftarrow \emptyset$
  for $i = 1, 2, \ldots,$ r do
      $S_i \leftarrow$TraceRidge $(x_i, \mathcal{D}, \mathcal{P})$
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{S_i\}$
  end for
  Output the set of ridge segments $\mathcal{P}$
```

where each segment is a sequence of consecutive ridge points stretching from a saddle to a maximum (or intersecting another ridge segment).

As the set of initial ridge points we can use the set of all the ridge points obtained by OriginalPCS $\mathcal{P}^1_{orig}$, i.e., all the ridge points reached by running FindRidge from every data point in our dataset. However, this is substantially excessive as in practice most ridge points discovered in this way are likely to be on the same ridge segments. Therefore, we use SparsePCS (algorithms 11), a revised implementation of OriginalPCS (algorithm 6), which uses SparseFindRidge (algorithm 12) instead of FindRidge (compare with algorithm 2) to produce only a subset of $\mathcal{P}^1_{orig}$ which will be used as the initial ridge points.

The key idea is that we would like to have a subset of ridge points which are separated by at least some fixed distance. At the beginning of every iteration, SparseFindRidge recovers the set `NN` of data points in a neighborhood of radius r_{nbhd} around the current probe position x and checks if they were *visited* in a previous execution of SparseFindRidge. If any point in `NN` was previously *visited*, the current execution of SparseFindRidge is terminated early (without reaching the ridge and producing a ridge point) and SparsePCS moves on to running SparseFindRidge from the next data point.

As described so far the algorithm operates by initially finding a (sparsified) set of ridge points, and then tracing the ridge segments from these ridge points. However, in order to completely eliminate the discovery of multiple ridge points on the same segment the processes of recovering an initial ridge point and tracing the ridge can be intertwined. Thus, during

Algorithm 11 SPARSEPCS

```
Input: data $\mathcal{D} = \{z_i\}$.
  visited $\leftarrow \emptyset$
  for $i = 1, \ldots, N$ do
      $x_i$, current_visited $\leftarrow$ SPARSEFINDRIDGE($z_i, \mathcal{D}$, visited)
      visited $\leftarrow$ visited $\cup$ current_visited
  end for
  Output $\mathcal{P}^1_{sparse} = \{x_i : x_i \neq \text{NULL}\}$
```

the execution of the algorithm from every data point operates in (at most) two stages: In the first stage, SPARSEFINDRIDGE is run from a data point. If the probe reaches a ridge the probe switches the second stage of tracing the ridge via TRACERIDGE and keeps marking the r_{nbhd}-neighborhood of the probe as *visited*. However, it should be noted that the execution of the algorithm will not be terminated in the second stage, if a previously *visited* data point is encountered. The reason for this is that due to the "curse of dimensionality" in order to be find a non-trivial neighborhood of data points we need to use a value r_{nbhd} which is substantially larger than the scale of the ridges and of d_{min}. Hence, a probe might be significantly far from previously discovered ridges and still encounter *visited* samples.

Algorithm 12 SPARSEFINDRIDGE

Input: initial point z, data $\mathcal{D}$, the set of previously visited data points `visited`.

Parameters: tolerance t_{ridge}, sparsification neighborhood radius r_{nbhd}.

```
current_visited $\leftarrow \emptyset$
$x_1 \leftarrow z$
for $k = 1, 2, \ldots$ do
    NN $\leftarrow$ FINDNN$(x_k; \mathcal{D}, r_{nbhd})$
    current_visited $\leftarrow$ current_visited $\cup$ NN
    if visited $\cap$ NN $\neq \emptyset$ then
        $x_{k+1} \leftarrow$ NULL
        Break
    end if
    $s_k \leftarrow$ MEANSHIFT$(x_k, \mathcal{D})$
    $Q \leftarrow$ LNCSUBSPACE$(x_k, \mathcal{D})$
    $s_k^{\perp} \leftarrow (I - QQ^T)s_k$
    $x_{t+1} \leftarrow x_k + s_k^{\perp}$
    until $\frac{s_k \cdot s_k^{\perp}}{\|s_k\|\|s_k^{\perp}\|} \leq t_{ridge}$        ▷ $s_k$ is approximately perpendicular to $s_k^{\perp}$
end for
Output the last $x_k$, current_visited
```

Chapter 5
RELATED WORK

Possibly the most common algorithm for dimensionality reduction is Principal Component Analysis (PCA) which linearly projects the data onto a linear subspace so to preserve as much as possible of the variance in the projection. Equivalently, PCA produces the linear projection, such that the mean-squared error (MSE) between the data points and their projection is minimized. As PCA is a linear process it preforms well when the data is spread on a low-dimensional subspace. However, it would produce a sub-optimal representation, if the data has a complex structure that cannot be linearly projected to a space of the dimensionality of the data.

Kernel PCA is one approach to overcome the linearity limitation of PCA. Kernel PCA operates by implicitly mapping the data into an even higher-dimensional space (possibly infinite-dimensional), so that the mapped data is well-suited for PCA. Thus, PCA is performed on the transformed representation implicitly by what is known as *the kernel trick*.

Another related method is Multidimensional Scaling (MDS). Like PCA, MDS finds mapping of a high-dimensional data into a lower-dimensional space (usually, 2- or 3-dimensional). However, while PCA finds a mapping that minimizes the MSE between the data points and their images, MDS observes the distances between all data-pairs, and produces maps the data so to minimize the MSE between data-pair distances and mapped-data-pair distances.

A popular extension of MDS is Isomap. In Isomap, a neighborhood graph is constructed either by connecting every data point to all other data points at most ε away, or to its k closest neighbors. Then, the shortest path distance between every two data points as an approximation of the Geodesic distance between the points along the (unknown) manifold. Finally, an MDS-like process is applied to the resulting distance matrix. Thus, Isomap tries to preserve in the mapping the Geodesic distance between data-pairs instead of the direct distances between them. [14]

In contrast to the aforementioned methods of dimensionality reduction which explicitly aim to preserve properties of the data both globally and locally, local methods focus on preserving only local properties with the expectation that the global invariant will emerge from the local constraints.

One prominent local method is Local Linear Embedding (LLE)[10]. Given a dataset $\mathcal{D} = \{z_1, \ldots, z_N\} \in \mathbb{R}^n$, LLE first selects for every sample z_i its neighbors $\mathcal{N}(i)$. It then finds a representation of z_i as a linear combination of its neighbors so to minimize the loss function

$$\varepsilon(W) = \sum_i \left| z_i - \sum_{j \in \mathcal{N}(i)} W_{ij} z_j \right|^2$$

by solving a least-squares problem. Finally, it solves for the lower-dimensional representation y_i of z_i while requiring that the representation via the neighbors is still maintained as closely as possible. Specifically, the following loss function is minimized

$$\Phi(y) = \sum_i \left| y_i - \sum_{j \in \mathcal{N}(i)} W_{ij} y_j \right|^2$$

by solving an eigenvalue problem.

Perhaps most relevant to our work is [13] which use SCMS to project the data points onto the density ridges defined by various eigenvectors of the Hessian. Similar to our proposal, from the projection point the ridge is traced up to the mode. This is done in order to estimate the curve length from the projection point to the mode which is used as coordinate of the point in a curvilinear coordinate system later used to unwrap the data. However, unlike us, in [13] the ridge is tracer by solving a differential equation, rather than taking a step in the space defined by the Hessian.

[5] provides an explicit formal proof to the convergence of the SCMS algorithm, which is only glazed over in [9]. Specifically, they show that under some (reasonable) conditions, the following holds for the sequence $\{x_i\}$ generated by SCMS:

1. The sequence $\{f(x_k)\}$ is non-decreasing and convergent.
2. $\lim_{k\to\infty} \|x_{k+1} - x_k\| = 0$

3. $\lim_{k\to\infty} \|(Q_k^{\perp})^T \nabla f(x_k)\| = 0$, where $Q_k^{\perp}$ is the $n \times (n-d)$ matirx whose columns span $W^{\perp}$ at x_k, i.e. the columns of $Q_k^{\perp}$ are the eigenvectors corresponding to the $n-d$ smallest eigenvalues of $\nabla^2 f(x_k)$.

This implies that SCMS converges and that when it does converges the probe will be on a principal surface. However, we have no guarantee that it converges to points in $\mathcal{P}^d \setminus \mathcal{P}^{d-1}$. This statement also implies that the stopping condition can either be the common condition on the step size $\|y_{k+1} - y_k\|$ becoming small enough or the gradient becoming close enough to perpendicular to the LNC (in fact, [5] suggest a simpler criterion for perpendicularity than the one presented in algorithm 2).

In addition, [5] also evaluates speed and accuracy of 4 different variants of SCMS on simple 2-d and 3-d synthetic datasets. The variants considered differ in the object on which the eigenvectors for the LNC basis are computed. The objects used are the following:

1. The Hessian of $f(x) = \ln p(x)$, as in the original SCMS algorithm.
2. The Hessian of $p(x)$ directly.
3. The estimated local covariance matrix using the κ nearest samples.
4. The estimated local covariance matrix using the κ nearest previous probe locations.

Note that the algorithm resulting from computing LNC basis as the eigenvectors of a local covariance matrix instead of Hessian is, in fact, local PCA. In their evaluation, they have found that while the resulting ridges are similar with respect to accuracy, as measured by average distance from the ground truth, the runtime of SCMS using the covariance matrices is about 5-6 times faster.

As is noted above, the basic SCMS algorithm relies on repeatedly estimating $\nabla \ln(p(x))$ and $\nabla^2 \ln(p(x))$, which involves computing the ratios of the first and second order derivatives of $p(x)$ to $p(x)$. One approach to estimating these values is taking the first and second order derivatives of KDE $p(x)$ and directly computing the ratios by dividing the relevant objects. As claimed by [12] this method might be flawed due to two main reasons: Firstly, the derivatives of KDE itself may not be an accurate estimators of the corresponding derivatives of the underlying probability distribution. Secondly, the inaccuracies in the original estimator may be amplified by the division. Instead, they propose directly estimating the density-derivative-ratios of interest with specially constructed estimators. For this purpose, they formulate the ratio estimation problem as a least-squares estimation problem. As stated by the relevant representer theorem [15], the minimizer of the least-squares problem is a linear combination of the employed kernel and its derivatives. Thus, the corresponding coefficients can be estimated from the data.

Note that given the coefficients of the minimizer computing the k-th order density-derivative-ratio estimate using N data points in n dimensions requires $\mathcal{O}(Nn^k)$. That is, the complexity of the resulting algorithm is comparable with that of basic SCMS. However, solving the LS problem to obtain the coefficient in the first place requires inverting a $2N$-by-$2N$ matrix for each dimension of the relevant ratio. Thus, the initial complexity of finding all the coefficients required for estimating the ratio of the Hessian to the probability density is $\mathcal{O}(N^6n^2)$. To avoid this complexity blow-up, [12] resort to subsampling the data points and using no more than 100 points of their experiments.

Chapter 6

PREPROCESSING PROCEDURES

6.1 Kernel width and neighborhood radius selection

The kernel width h was chosen by following a procedure inspired by the *coverage risk minimization* method of [2]. Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two datasets sampled from the same distribution. Given a kernel width h, denote by $\hat{\mathcal{P}}_{1,h}$ and $\hat{\mathcal{P}}_{2,h}$ the sets of principal curves obtained by PCS when estimating the probability density with kernel width h on $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Then the *estimated coverage risk* of the distribution with kernel width h is $\hat{R}_h = \frac{1}{2}\mathbb{E}[W_{1,h} + W_{2,h}]$, where $W_{1,h}$ is the distance of a point sampled uniformly at random from $\hat{\mathcal{P}}_{1,h}$ to the set of curves $\hat{\mathcal{P}}_{2,h}$, and $W_{2,h}$ is defined symmetrically. [2] argue that one should pick the minimizer of $\hat{R}_h$ as the kernel width h.

In practice, $\mathcal{D}1$ and $\mathcal{D}2$ are generated by randomly partitioning the given dataset into two disjoint equally sized subsets. In addition, the principal curves are approximated by a set of piecewise-linear curves defined by sets of consecutive ridge points. To be precise, a *ridge segment* is a sequence of ridge points $S = (x_{S,i})_{i=1}^{l}$ (ordered by their appearance along the ridge from a saddle to a maximum, as in algorithm 9). Let S be a ridge segment of size l, then for every $i = 1, \ldots, l-1$ denote $\mathrm{len}_S(i) = \|x_{S,i} - x_{S,i+1}\|$ the distance between the i'th and the $(i+1)$'th points in S (and define $\mathrm{len}_S(0) = \mathrm{len}_S(l) = 0$). Define the weight $w_{S,i}$ of a point $x_{S,i}$ as $w_{S,i} = \frac{1}{2}\left(\mathrm{len}_S(i-1) + \mathrm{len}_S(i)\right)$. Let $\mathrm{len}(S) = \sum_{i=1}^{l} \mathrm{len}_S(i) = \sum_{i=1}^{l} w_{S,i}$ be the total length of S as a piecewise-linear curve.

The output of our algorithm is a set of ridge segments $\mathcal{P} = \{S_i\}_{i=1}^{k}$. Denote by $\mathrm{len}(\mathcal{P}) = \sum_{S\in\mathcal{P}} \mathrm{len}(S) = \sum_{S\in\mathcal{P}} \sum_{i=1}^{|S|} w_{S,i}$ the total length of the curves in $\mathcal{P}$.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two sets of ridge segments, and let $d(x, \mathcal{P})$ be the distance of a point x to a set of ridge segments $\mathcal{P}$ as defined in (4.2). Then the *simple* empirical expected distance

$\hat{W}_1^{(s)}$ of $\mathcal{P}_1$ from $\mathcal{P}_2$ is the (simple) average distance of a ridge point x in $\mathcal{P}_1$ to $\mathcal{P}_2$:

$$\hat{W}_s(\mathcal{P}_1, \mathcal{P}_2) = \frac{1}{\sum_{S \in \mathcal{P}_1} |S|} \sum_{S \in \mathcal{P}_1} \sum_{i=1}^{|S|} d\left(x_{S,i}, \mathcal{P}_2\right)$$

The *weighted* average distance $\hat{W}_1^{(w)}$ of $\mathcal{P}_1$ from $\mathcal{P}_2$ is the weighted average distance of a point x in $\mathcal{P}_1$ from $\mathcal{P}_2$, where the weight of a point $x_{S,i}$ is proportional to $w_{S,i}$:

$$\hat{W}_w(\mathcal{P}_1, \mathcal{P}_2) = \frac{1}{\operatorname{len}(\mathcal{P}_1)} \sum_{S \in \mathcal{P}_1} \sum_{i=1}^{|S|} w_{S,i} \cdot d\left(x_{S,i}, \mathcal{P}_2\right)$$

In our experiments we computed two estimates of the coverage risk: The simple coverage risk:

$$\hat{R}_h^{(s)} = \tfrac{1}{2}\left[\hat{W}_s(\mathcal{P}_1, \mathcal{P}_2) + \hat{W}_s(\mathcal{P}_2, \mathcal{P}_1)\right]$$

and the weighted coverage risk:

$$\hat{R}_h^{(w)} = \tfrac{1}{2}\left[\hat{W}_w(\mathcal{P}_1, \mathcal{P}_2) + \hat{W}_w(\mathcal{P}_2, \mathcal{P}_1)\right]$$

Risk minimization was used to set both the kernel width h and the neighborhood radius r_{nbhd}. Thus, we tuned h and r_{nbhd} by means of a grid search and obtained a matrix of the risk for every combination of h and r_{nbhd} in our search space. In our experiments, the risks tends to slightly present a major increase at some threshold in when varying either one of our parameters. We set the parameters to the values just before this increase. Note that, as the risk estimations are noisy, the values we pick may not be the absolute minimizers of the risk. In our experiments we have varied h is a logarithmic scale in a range that seemed suitable to each dataset. r_{nbhd} was varied linearly between $0.2 \cdot h$ and $2 \cdot h$.

When operating on synthetic datasets, in order to avoid recomputing h when embedding the same distribution in spaces of different number of dimensions, we assume that $h \propto \sqrt{n}$ similarly to the expected magnitude of the noise $\mathbb{E}[||z||]$. Thus, we can find h once and rescale it for different dimensions.

6.2 Dimensionality estimation

Before running any of the algorithms described, we would first like to verify that the intrinsic dimensionality of the data is indeed approximately 1. Thus, as a preprocessing step we run a simple dimensionality estimation algorithm 13.

Algorithm 13 EstimateDimension

```
Input: ridge points $\{x_i\}_{i=1}^N$, data $\mathcal{D} = \{z_j\}$, a range of radii $\mathcal{R} = \{r_k\}_{k=1}^K$.
  for $k = 1, \dots, K$ do
      $C(k) \leftarrow \frac{1}{N} \sum_i \sum_j \mathbb{1}\{\|x_i - z_j\| < r_k\}$
  end for
  $\bar{d} \leftarrow$ slope of the linear regression of $\log(C)$ against $\log(\mathcal{R})$
  Output dimensionality estimate $\bar{d}$
```

Let x be a point in $\mathbb{R}^n$ and $p(x)$ the probability density at x. Assume that our samples $\mathcal{D}$ are distributed uniformly on a manifold of dimension d in a close enough neighborhood of x, then the number of samples $c(x, r)$ we would observe in a ball of radius r around x is

$$c(x, r) \approx Np(x)V(d)r^d$$

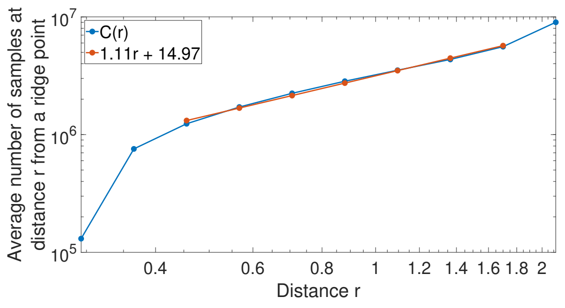
where $V(d)$ is the volume of the unit ball in d-dimensions. By taking the log of the expression on both side, we obtain

$$\log(c(x, r)) \approx C + d\log(r)$$

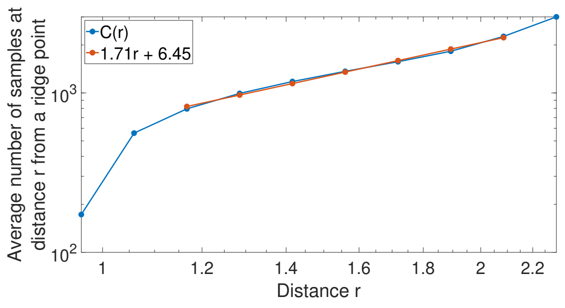
For some constant C. That is, $\log(c)$ and $\log(r)$ have a linear relationship, whose slope is (approximately) the intrinsic dimension d. Therefore, we can estimate d by regressing on $\log(c)$ on $\log(r)$ over a suitable range of r.

A more robust estimate of $c(x, r_k)$ can be obtained by computing the average number of samples in a ball of radius r, $C(r_k) = \frac{1}{N} \sum_i c(x_i, r_k)$ over a set of points $\mathcal{X} = \{x_i\}_{i=1}^N$ for every fixed radius r_k. For the purpose of our objective, we are interested in the distribution of samples close to potential ridge points, and therefore as the set $\mathcal{X}$ we use the set of ridge points points obtained by some variant of PCS.

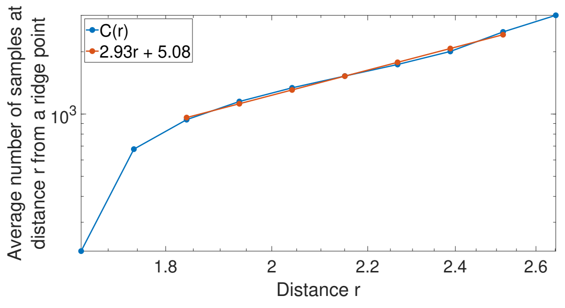
The resulting plots of the sample count $c(r)$ versus the neighborhood radius r for the dataset 'O' and various ambient dimensionalities are presented in figure 6.1. Recall that the dimensionality estimation is the slope of the linear approximation (in orange). We can see that the estimate is reasonably accurate at low ambient dimensionalities but the estimated dimensionality increases from 1.11 to 4.18 as the ambient dimensionality increases from 100 to 5000.



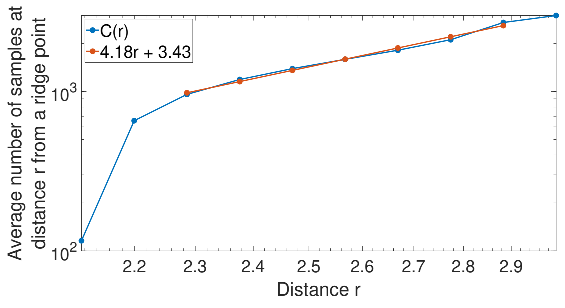
(a) dataset 'O' (n = 100; N = 3000)



(b) dataset 'O' (n = 1000; N = 3000)



(c) dataset 'O' (n = 3000; N = 3000)



(d) dataset 'O' (n = 5000; N = 3000)

Figure 6.1: Intrinsic dimension estimation for the 'O' dataset with $N = 3000$ and various values of ambient dimension. Each plot shows the average number of samples at various distances away from preliminary ridges points (blue) and the regression on the linear section of curve (orange), whose slope is the estimate of the dimension. Note that both axes are in logarithmic scale.

6.3 ANN index construction

In our experiments we used the FLANN library [7] to perform the approximate nearest-neighbor search (see section 2.6). The radius of the neighborhood of a point used for the computations r_{comp} was set to $r_{comp} = 3.5h$, where h is the kernel width used for the specific dataset.

Using FLANN requires indexing every dataset prior to the execution of the algorithms.

The time required for constructing the indices t_{pp} is not included in the following runtime measurements of the algorithms. For references, following are the values of t_{pp} for the the 'O' dataset with $N = 3000$, $3 \cdot 10^4$ in various number of dimensions n.

N	n			
	100	1000	3000	5000
3000	33	71	142	333
$3 \cdot 10^4$	116	984	1172	7529

Table 6.1: Preprocessing time t_{pp} in seconds of the dataset 'O' with $N = 3 \cdot 10^3$, $3 \cdot 10^4$.

Chapter 7
EXPERIMENTAL EVALUATION

7.1 Evaluation on synthetic data

7.1.1 Data generation

In the following we present experimental evaluation of OriginalPCS and MorsePCS on synthetic datasets. To allow clearer visualization of the results, the datasets where generated by sampling points on a one-dimensional curve which is naturally embedded in $\mathbb{R}^2$, specifically, a circle with a radius of 1 (dataset 'O') and a 'Z' shaped curve (dataset 'Z') within the box $[-1, 1] \times [-1, 1]$. We set the distribution over the curve to be non-uniform, in order to break symmetries and induce a hilly topography even in the presence of symmetries in the shape of the curve. Then, the intrinsically two-dimensional data is embedded in $\mathbb{R}^n$ by applying a random rotation on all the samples. For each of the two distributions 'O' and 'Z' we sampled datasets with an ambient space dimensionality $n \in \{100, 1000, 3000, 5000\}$ and a sample size $N \in \{3000, 3 \cdot 10^4\}$. Finally, n-dimensional i.i.d. Gaussian noise $z \sim \mathcal{N}(0, \sigma_{noise}^2 I)$ is added to every sample. For dataset 'O' $\sigma_{noise} = 0.03$ and for dataset 'Z' $\sigma_{noise} = 0.02$.

7.1.2 Evaluation method

OriginalPCS was run twice on every dataset: Once with FindRidge being SCMS and once with FindRidge being L-SCMS. We compare the performance of the algorithms for each instantiation of FindRidge with respect to runtime and evaluate accuracy of the output.

Runtime The runtime of each algorithm can be roughly broken down into 3 categories: Time spent on core operations executed by all the algorithms (e.g., projecting the mean shift step onto the LNC space), denoted by t_{core}; Time spent on algorithm-specific operations (e.g., computing the LNC space, updating the L-BFGS matrices), denoted by t_{alg}; and time spent recovering the nearest neighbors of a given probe for various purposes (e.g., estimating the gradient or the Hessian at a probe position, recovering the neighborhood of the probe), denoted by t_{nn}. We denote by t_{total} the total runtime, so that $t_{total} = t_{core} + t_{alg} + t_{nn}$.

Accuracy Note that the principal curves need not overlap the underlying manifold [4]. Thus, even though the manifold from which data is sampled is known, the true principal curves are unavailable for these datasets. Therefore, it is hard to evaluate the accuracy of any algorithm precisely. For this reason, we resort to evaluating the accuracy of L-SCMS in comparison to the outputs produced by SCMS. The comparison between the outputs is conducted both qualitatively, by observing the outputs, and quantitatively, by measuring the average (simple) distance $\hat{W}_s$ between the output of the considered variant to the other variant.

7.1.3 Results

Tables 7.1 and 7.2 summarize the results of the execution of OriginalPCS with either SCMS or L-SCMS on dataset 'O' with $N = 3000$ and $N = 30,000$ respectively. As can be seen from the table, the SCMS variant is faster than L-SCMS when $n = 100$. However, as the number of dimensions increases the SCMS variant becomes slower at a much faster rate than the L-SCMS variant. As a result, L-SCMS is faster for $n \geq 1000$. A closer inspection of the runtime breakdown shows that for $N = 3000$ almost all of the runtime of SCMS is spent on algorithm-specific operations, namely computation of the LNC subsapce. In contrast, only about half the runtime of L-SCMS is spent on algorithm-specific operations. Instead, more time is spent on core operation and NN queries, as more steps are required due to the inaccuracy of the algorithm.

For $N = 30,000$, we see that the algorithm-time goes to about $\frac{3}{4}$ of the total runtime. presumably, this is due the increase in the computational cost of operations on the density estimate, such taking its derivative. The increased cost follows from the fact that the number of samples used in computations is bigger for two reason: Firstly, there are potentially more samples in the neighborhood of the probes; And, secondly, the number of samples used for computations is capped by $0.1 \times N$, and thus increasing N results in a higher cap on the number of samples used.

Observing the average distance between the resulting ridge point sets, we see that the resulting ridge points are a few orders of magnitude closer than the expected width of the ridges due to the noise, meaning that the two results are in agreement.

n	FindRidge algorithm	$\hat{W}_s$	t_{alg} [sec]	t_{nn} [sec]	t_{total} [sec]	#steps	$\frac{time}{step}$
100	SCMS	0.0023	94	10.8	134.6	12,220	0.011
	L-SCMS	0.0025	148	31.3	222.2	18,126	0.012
1000	SCMS	0.0097	$1.97 \cdot 10^3$	17	$2.05 \cdot 10^3$	9273	0.22
	L-SCMS	0.0118	$0.48 \cdot 10^3$	97	$0.84 \cdot 10^3$	28,269	0.03
3000	SCMS	0.0206	$20.5 \cdot 10^3$	43	$20.67 \cdot 10^3$	10,332	2
	L-SCMS	0.0154	$1.5 \cdot 10^3$	416	$2.87 \cdot 10^3$	72,623	0.04
5000	SCMS	0.0343	$81.9 \cdot 10^3$	$0.13 \cdot 10^3$	$82.1 \cdot 10^3$	11,486	7.2
	L-SCMS	0.0276	$3.2 \cdot 10^3$	$1.50 \cdot 10^3$	$6.14 \cdot 10^3$	106,663	0.058

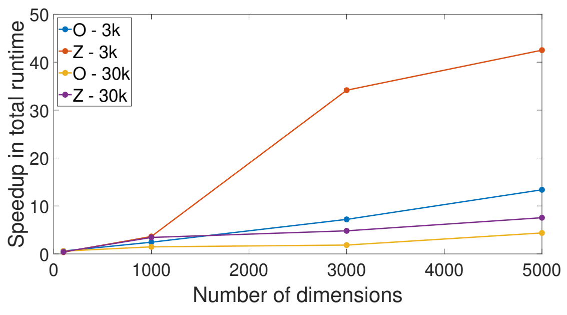
Table 7.1: Results on OriginalPCS using different ridge recovery methods on dataset 'O' with $N = 3000$.

Similar results can be seen in tables 7.3 and 7.4, which summarize the results on dataset 'Z' with $N = 3000$ and $N = 30,000$ respectively.

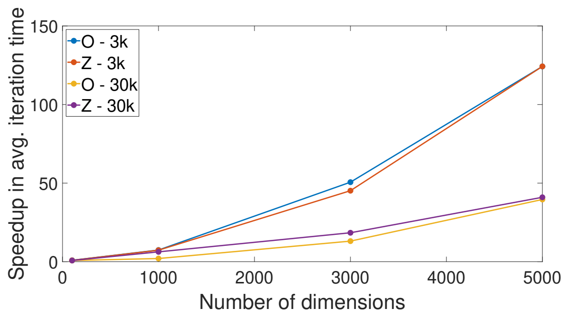
Figure 7.1 presents the speedup of L-SCMS over SCMS for the execution of OriginalPCS. As expected, the speedup increases as the ambient dimensionality n increases and reaches a speedup of the total runtime of more than $40\times$ for the 'Z' dataset with $N = 3000$. The runtime per iteration of L-SCMS is more than 120 times faster than SCMS for both dataset 'O' and 'Z' with $N = 3000$ when $n = 5000$.

n	FindRidge algorithm	$\hat{W}_s$	t_{alg} $[10^3 \cdot sec]$	t_{nn} $[10^3 \cdot sec]$	t_{total} $[10^3 \cdot sec]$	#steps	$\frac{time}{step}$
100	SCMS	0.0011	1.85	0.35	2.58	91,419	0.028
	L-SCMS	0.0013	3.34	1.15	4.39	119,528	0.037
1000	SCMS	0.0036	32.3	1.14	34.1	84,672	0.403
	L-SCMS	0.0056	20.2	4.92	23.1	116,368	0.199
3000	SCMS	0.0057	386	5.17	393	151,065	2.604
	L-SCMS	0.0064	153	51.7	213	1,072,716	0.199
5000	SCMS	0.0088	1550	9.95	1562	108,097	14.45
	L-SCMS	0.0099	252	97.1	357	978,595	0.37

Table 7.2: Results on OriginalPCS using different ridge recovery methods on dataset 'O' with $N = 30,000$.



(a) Speedup



(b) Speedup per step

Figure 7.1: The speedup of L-SCMS over SCMS for OriginalPCS as a function of the dimensionality n. The speedup can be observed both in the total runtime (left) and the time per step (right).

n	FindRidge algorithm	$\hat{W}_s$	t_{alg} [sec]	t_{nn} [sec]	t_{total} [sec]	#steps	$\frac{time}{step}$
100	SCMS	0.0035	81	8	114	12,100	0.010
	L-SCMS	0.0030	216	39	319	29,073	0.011
1000	SCMS	0.0126	$3.00 \cdot 10^3$	26	$3.12 \cdot 10^3$	13,815	0.226
	L-SCMS	0.0090	$0.49 \cdot 10^3$	97	$0.85 \cdot 10^3$	27,855	0.031
3000	SCMS	0.0243	$69.6 \cdot 10^3$	128	$70.1 \cdot 10^3$	37,452	1.873
	L-SCMS	0.0115	$1.15 \cdot 10^3$	269	$2.1 \cdot 10^3$	49,558	0.041
5000	SCMS	0.0519	$258 \cdot 10^3$	$0.3 \cdot 10^3$	$258 \cdot 10^3$	36,903	7.003
	L-SCMS	0.0233	$3 \cdot 10^3$	$1.17 \cdot 10^3$	$6 \cdot 10^3$	107,885	0.056

Table 7.3: Results on OriginalPCS using different ridge recovery methods on dataset 'Z' with $N = 3000$.

7.2 Evaluation on real data

7.2.1 Molecular data

Locating the ridges of the energy levels as the configuration of molecules changes interests chemists as these signify phase transitions. We experimented with simulated data of two molecules whose structures induce 1-dimensional manifolds: Toluene and Chloromethane.

The Toluene dataset consists of $N = 49,199$ samples in $n = 10$ dimensions. Figure 7.2 shows the principal curves recovered by SCMS and L-SCMS variants of MorsePCS. As can be seen, the data forms a circle, which is recovered by both algorithms, and the outputs of the algorithms are close to each other with a simple risk of $\hat{R}^{(s)} = 0.005$.

One difficulty with this dataset is that the topography near the ridge is close to a plateau. Therefore, if the data is over-smoothed, the ridge-tracing algorithm may fail, as there is no gradient to follow along the ridge. In the future, some more robust method for handling this challenge should be considered.

n	FindRidge algorithm	$\hat{W}_s$	t_{alg} $[10^3 \cdot sec]$	t_{nn} $[10^3 \cdot sec]$	t_{total} $[10^3 \cdot sec]$	#steps	$\frac{time}{step}$
100	SCMS	0.0011	2.59	0.43	3.46	99,741	0.035
	L-SCMS	0.0013	6.16	1.61	7.90	173,525	0.046
1000	SCMS	0.0036	105	1.52	108	105,661	1.02
	L-SCMS	0.0056	25.8	6.70	31	192,960	0.16
3000	SCMS	0.0057	621	5.25	628	117,951	5.33
	L-SCMS	0.0064	102	30.41	130	451,211	0.29
5000	SCMS	0.0105	1,448	6.80	1,498	114,783	13.05
	L-SCMS	0.0087	151	47.69	1,984	623,096	0.32

Table 7.4: Results on OriginalPCS using different ridge recovery methods on dataset 'Z' with $N = 30,000$.

The Chloromethane dataset consists of $N = 23,306$ samples in $n = 6$ dimensions. Figure 7.3 shows the principal curves recovered by SCMS and L-SCMS variants of MorsePCS. Again, both algorithm recover similar principal curves with a simple risk of $\hat{R}^{(s)} = 0.013$.

7.2.2 MNIST

The MNIST dataset is a classic dataset in machine learning containing $N = 60,000$ 28×28 images of digits along with labels for the depicted digit. Thus, the ambient dimensionality of the samples in this dataset is $n = 28^2 = 784$. The high-dimensionality of this dataset makes it relevant for our application. We have experimented with subsets of the dataset consisting of the images depicting one of several digits. However, in the end, we to focus on images depicting the number '1', as these seemed to be lying closest to a 1-dimensional manifold.

Figure 7.4 shows the principal curves recovered by the SCMS variant of MorsePCS. It also presents key images corresponding to position along the principal curve. As we can see the prominent property governing the transition along the principal curve is the slant of the

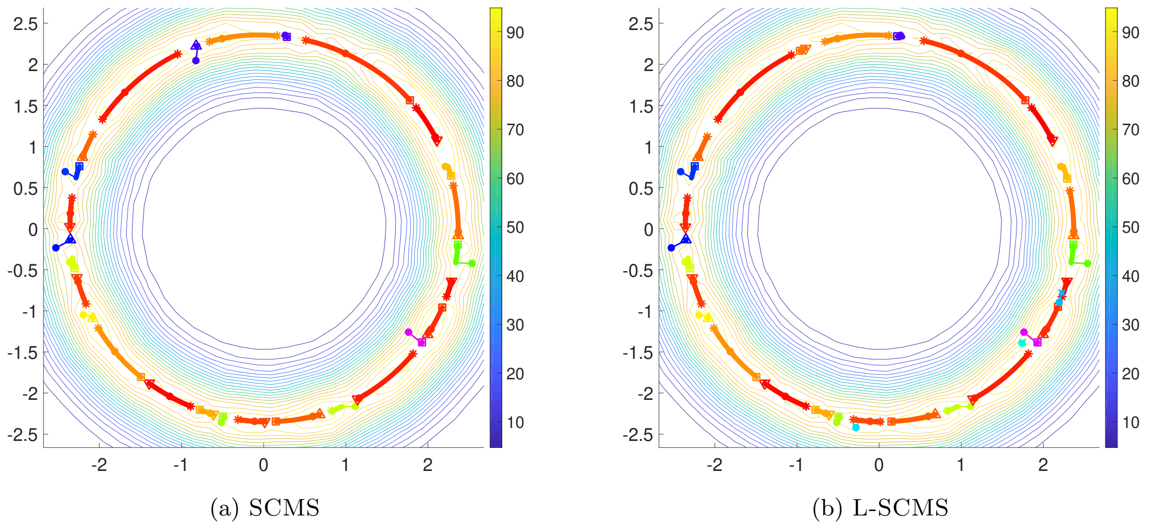

(a) SCMS (b) L-SCMS

Figure 7.2: The principal curves of 'Toluene' recovered by the SCMS (left) and L-SCMS (right) variants of MorsePCS. △ denotes a curve ending in the ascent direction (maximum) and ▽ denotes a curve ending in the decent direction (saddle). The contour lines show (an approximation) of the the probability density.

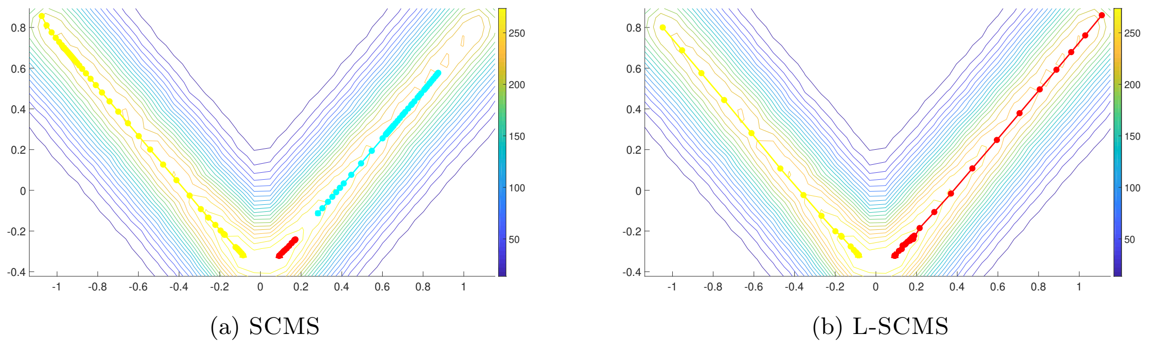

(a) SCMS (b) L-SCMS

Figure 7.3: The principal curves of 'Chloromethane' recovered by the SCMS (left) and L-SCMS (right) variants of MorsePCS. △ denotes a curve ending in the ascent direction (maximum) and ▽ denotes a curve ending in the decent direction (saddle). The contour lines show (an approximation) of the the probability density.

digit, which constitutes a non-linear manifold in the data.

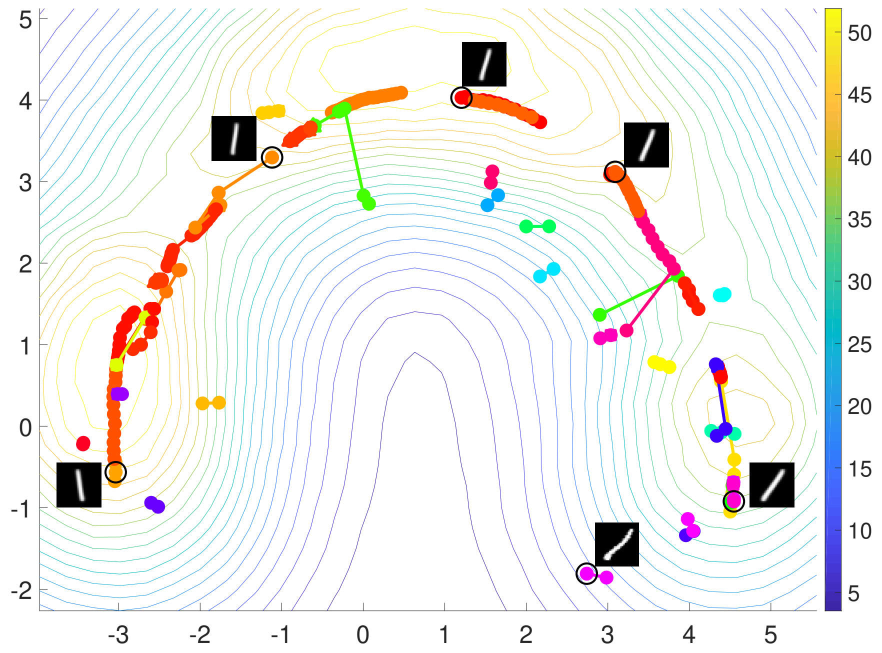


Figure 7.4: The principal curves of 'mnist1' recovered by the SCMS variant of MorsePCS and the images corresponding to select positions along the principal curve.

7.2.3 Sloan Digital Sky Survey Dataset

The Sloan Digital Sky Survey (SDSS) dataset registers the angular position and speed of 573,945 astronomical objects with respect to Earth in the form (RA, dec, z) where:

- **Right Ascension (RA)** is like a longitude and it runs from 0 to 360 degrees.
- **Declination (dec)** is like a latitude and it runs from -90 to 90 degrees. In our dataset nearly all the object have a declination value between -11 and 70.
- **Redshift (z)** is a measure of how fast an object is moving away from Earth. In our dataset nearly all objects have a redshift value between 0 and 0.3.

For the purpose of our experiments we picked a small slice of objects with redshift between 0.075 and 0.080 leaving 25,311 objects. Further, we focused on a big cluster lying in the RA interval $[100, 275]$ leaving 23,099 objects. The range of redshift value was chosen such that it is small enough that we can it and treat all the objects as lying on a 2D plane while retaining as many of them as possible at the same time.

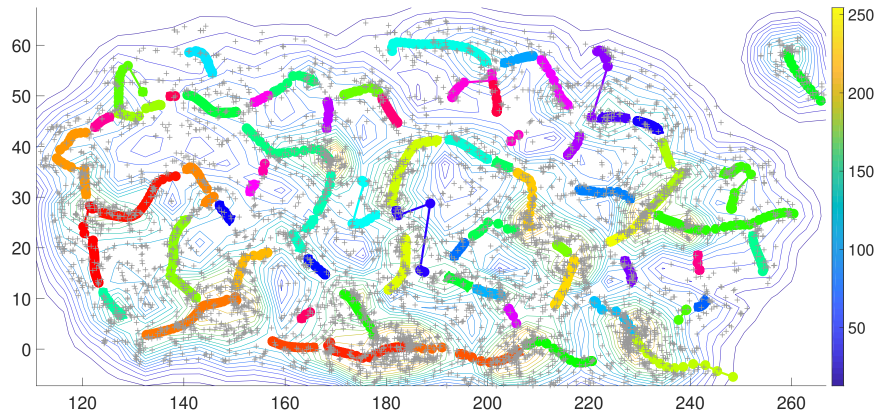


Figure 7.5: The principal curves of 'SDSS' recovered by the SCMS variant of MorsePCS.

Figure 7.5 presents the principal curves recovered by the SCMS variant of MorsePCS. We can see that even though the topography is quite bumpy and noisy the algorithm manages to recover many of the major filaments in the data.

Chapter 8
CONCLUSION

In this thesis, we have presented L-SCMS, an approximation algorithm to SCMS that computes the LNC of a density function by using a projection of its Hessian on the space spanned by L-BFGS. Thus, L-SCMS is able to achieve a speedup of more than 100x over SCMS.

We have also presented TraceRidge, an extension of SCMS, that instead of stopping whenever a ridge is reached, transitions to performing gradient (ascent or descent) along the ridge, while maintaining it by using the LNC subspace. In addition to making the algorithm more efficient, this also allowed us the recover more information about the structure of the underlying distribution and in particular recover its maxima and saddle points.

We have provided compelling results, showing that in high-dimensions our algorithms can indeed recover the density ridges faster than existing algorithms without significant impact to the quality of the results. Indeed, we have a shown that MorsePCS is able to recover ridge segments and identify critical points along ridge.

Still, finding more robust methods for traversing the ridge and identifying saddle points in particular should be explored in the future as means to enhancing the results generated by MorsePCS.

BIBLIOGRAPHY

[1] Richard H Byrd, Jorge Nocedal, and Robert B Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1-3):129–156, 1994.

[2] Yen-Chi Chen, Christopher R Genovese, Shirley Ho, and Larry Wasserman. Optimal ridge detection using coverage risk. In *Advances in Neural Information Processing Systems*, pages 316–324, 2015.

[3] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975.

[4] C. Genovese, M. Perone-Pacifico, I. Verdinelli, and L. Wasserman. Nonparametric ridge estimation. *ArXiv*, abs/1212.5156, 2012.

[5] Y. A. Ghassabeh, T. Linder, and G. Takahara. On some convergence properties of the subspace constrained mean shift. *Pattern Recognit.*, 46:3140–3147, 2013.

[6] Trevor Hastie and Werner Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84(406):502–516, 1989.

[7] Marius Muja and David G Lowe. Flann, fast library for approximate nearest neighbors. In *International Conference on Computer Vision Theory and Applications (VISAPP'09)*, volume 3. INSTICC Press, 2009.

[8] Jorge Nocedal and Stephen J Wright. Numerical optimization 2nd, 2006.

[9] Umut Ozertem and Deniz Erdogmus. Locally defined principal curves and surfaces. *Journal of Machine learning research*, 12(Apr):1249–1286, 2011.

[10] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290 5500:2323–6, 2000.

[11] H. Sasaki, T. Kanamori, and Masashi Sugiyama. Estimating density ridges by direct estimation of density-derivative-ratios. In *AISTATS*, 2017.

[12] Hiroaki Sasaki, Takafumi Kanamori, Aapo Hyvärinen, Gang Niu, and Masashi Sugiyama. Mode-seeking clustering and density ridge estimation via direct estimation of density-derivative-ratios. *The Journal of Machine Learning Research*, 18(1):6626–6672, 2017.

[13] M. Shaker, Jonas Nordhaug Myhre, M. Kaba, and Deniz Erdoğmuş. Invertible nonlinear cluster unwrapping. *2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, 2014.

[14] J. Tenenbaum, V. De Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290 5500:2319–23, 2000.

[15] Ding-Xuan Zhou. Derivative reproducing properties for kernel methods in learning theory. *Journal of computational and Applied Mathematics*, 220(1-2):456–463, 2008.