ANALYZING AND EXTENDING THE DISTANCE-TO-MEASURE GRADIENT FLOW USING HIGHER ORDER VORONOI DIAGRAMS

by

Patrick Albert O'Neil
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Mathematics

Committee:

_____	Dr. Thomas Wanner, Dissertation Director
_____	Dr. Daniel Anderson, Committee Member
_____	Dr. Harbir Antil, Committee Member
_____	Dr. Lizette Zietsman, Committee Member
_____	Dr. David Walnut, Department Chair
_____	Dr. Donna M. Fox, Associate Dean, Office of Student Affairs & Special Programs College of Science
_____	Dr. Peggy Agouris, Dean, College of Science

Date: _____ Spring Semester 2017
George Mason University
Fairfax, VA

Analyzing and Extending the Distance-to-Measure Gradient Flow

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Patrick Albert O'Neil
Master of Science
George Mason University, 2015
Bachelor of Science
Virginia Polytechnic and State University, 2011

Director: Dr. Thomas Wanner, Professor
Department of Mathematical Sciences

Spring Semester 2017
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to all of my loved ones, including Gail and Patrick O'Neil, Brian O'Neil, and especially Jared Harris. Their relentless support made this possible.

Acknowledgments

Some of the material in Chapter 5 is based on data services provided by the OpenTopography Facility with support from the National Science Foundation under NSF Award Numbers 1226353 & 1225810.

Table of Contents

List of Figures

Abstract

ANALYZING AND EXTENDING THE DISTANCE-TO-MEASURE GRADIENT FLOW
Patrick Albert O'Neil, PhD
George Mason University, 2017
Dissertation Director: Dr. Thomas Wanner

Point cloud data arises naturally from 3D scanners, LiDAR sensors, and industrial computed tomography among other sources. Most point clouds obtained through experimental means exhibit some level of noise, inhibiting mesh reconstruction algorithms and topological data analysis techniques. To alleviate the problems caused by noise, smoothing algorithms are often employed as a preprocessing step before attempting to reconstruct the sampled measure. Moving least squares is one such technique, however it is designed to work on surfaces in $\mathbb{R}^3$. As many interesting point clouds naturally live in higher dimensions, we seek a method for smoothing higher dimensional point clouds. To this end, we turn to the distance-to-measure function.

In this dissertation, we provide a theoretical foundation for studying the gradient flow of the distance-to-measure function induced by the empirical measure, as introduced by Chazal, Cohen-Steiner, and Mérigot. In particular, we find a method for solving the distance-to-measure gradient flow using higher order Voronoi diagrams. Using this framework, we formulate the gradient flow system as a Filippov system. We are then able to characterize the dynamics of the system and produce a directed acyclic graph which reflects these dynamics. Using this diagram, one can follow the directed edges to determine where points in various regions of the system will flow.

Having established the theoretical foundation for the gradient flow system induced by the distance-to-measure function, we turn our attention to providing novel modifications to the system which alleviate some of the shortcomings of the original flow. In particular, we extend and incorporate some techniques from the field of surface reconstruction. This produces the normal bundle flow, a new technique for smoothing high dimensional point clouds. Additionally, we build a method for improving the sampling density of a point cloud by adding a diffusive term to the flow. This technique has immediate applications to data obtained from line scanners, where non-uniformity in the sampling can be a major concern. Next, we reformulate the gradient flow to adapt to the local geometry of the point cloud in an attempt to preserve high curvature features that may otherwise be smoothed away during the flow. This involves developing a new method of approximating curvature of a point cloud in high dimensions. As a final modification, we show that the normal bundle flow shares a connection with the gradient flow induced by the Mahalanobis distance, a function commonly used in statistics. With this connection in mind, we develop a generalized version of the gradient flow which has the Mahalanobis flow and the normal bundle flow as special cases.

Finally, we analyze the performance of the gradient flow on many different point clouds obtained through various means. First we analyze the gradient flow system on simple geometries such as circles and spheres. Then we apply the gradient flow to point clouds obtained from 3d scanning. These point clouds exhibit much richer geometries and therefore pose a more difficult smoothing task. Lastly, we look at large LiDAR point clouds and analyze the computational performance of a couple of the gradient flows we have explored.

Chapter 1: Introduction

There has been substantial research effort (for example, see [52], [28], [33], [49], [39]) concerned with analyzing point clouds drawn from an underlying manifold M of $\mathbb{R}^N$. Of primary focus has been the topological and geometric reconstruction of a manifold from a point cloud sampled from the manifold. Since most applications of point clouds involve noisy data collection processes, the points must be assumed to lie near the underlying manifold instead of on the surface. Therefore, algorithms are desired which produce both topologically and geometrically accurate results and are robust to varying levels of noise.

One approach to reducing noise is through the use of smoothing gradient flows. In the continuous case, examples of such gradient flows are mean curvature flow [56], Willmore flow [7], and Ricci flow [40]. In particular, the mean curvature flow of a surface S is given by

$$\frac{\partial S}{\partial t} = D\Delta S$$

where Δ is the Laplace-Beltrami operator on the surface S and D is a constant controlling the speed of the flow. Many approaches to point cloud smoothing construct a discrete version of the Laplace-Beltrami operator and induce a flow similar to the mean curvature flow. Inducing such a flow on a manifold will not change the topology of the manifold. However, since we are dealing with a zero-dimensional subset of Euclidean space, it is not clear how a gradient flow would affect the geometric and topological nature of meshes constructed from the point cloud evolving under the flow.

To investigate how a flow changes the topology of a mesh constructed from the point cloud, we turn to the field of topological data analysis. In particular, we use a topological descriptor known as the persistence diagram to summarize a point cloud's topological properties. Through the use of the stability theorem for persistence diagrams, we see that the

gradient flow on the point cloud induces a flow through the space of persistence diagrams. That is, when we evolve the point cloud, we evolve the persistence diagram in a continuous fashion. This creates so called vineyards, tracking topological changes throughout the gradient flow. Under this framework, we can begin to ask questions about the convergence of these gradient flows and whether inducing a gradient flow on a point cloud causes the persistence diagram of the evolving point cloud to converge to the persistence diagram associated with the sampled manifold.

In this dissertation, we investigate several gradient flows designed to smooth point clouds. We build a theoretical foundation for the gradient flow induced by the distance-to-measure function and introduce novel modifications to this gradient flow. We also present numerical results concerning the smoothing performance of these flows both in Euclidean space and in the space of persistence diagrams. Through this analysis, we hope to further the understanding of these gradient flows and how they may be applied to noisy and incomplete data.

1.1 Contributions

Although point cloud smoothing has been a subject of substantial research effort recently, the primary focus of the developed algorithms has been surface reconstruction. Since this problem is naturally posed for a 2-dimensional manifold embedded in $\mathbb{R}^3$, many of the available techniques were designed to work in low dimensions and become too computationally expensive in higher dimensions. When constructing a smoothing algorithm in higher dimensions, it is important to keep computational complexity in mind. In this dissertation, we analyze a gradient flow based on the so-call distance-to-measure function. Unlike existing methods, which involve expensive operations such as polynomial fitting, our techniques simply require one to compute a gradient of the nearest neighbors. This allows the algorithm to scale better to higher dimensions. Additionally, we build on this gradient flow through several extensions that are designed to improve the smoothing properties of the system.

In what follows, we build a theoretical foundation for the study of the gradient flow

arising from the distance-to-measure function induced by the empirical measure. As shown by Chazal et al in [14], the distance-to-measure function reduces to a simple k-nearest neighbor function when the measure in question is the empirical measure. Chazal showed we can then take the gradient of the k-nearest neighbor function and induce a flow on the point cloud. Building on Chazal's work defining this gradient flow, we build a theoretical framework for studying the flow using higher order Voronoi diagrams, a generalization of the traditional Voronoi diagram to neighbor sets of cardinality greater than one.

Using the Voronoi structure, we are able to prove a great deal about the k-nearest neighbors gradient flow. For example, we find all the sinks of the system and give conditions for a Voronoi region to contain a sink. Next, we show that the system has no periodic orbits. Additionally, we establish the fact that the gradient flow system is a piecewise-smooth dynamical system, and in particular a Filippov system. From this, we are able to prove that the system does not exhibit any sliding motion, a critical result for numerical simulations of the flow. Furthermore, we are able to construct a flow graph which qualitatively encapsulates the dynamics of the system. The flow graph is a directed, acyclic graph which describes where points in each Voronoi region will travel under the influence of the induced gradient.

Taking a note from some of the existing surface reconstruction approaches, we then introduce a modification of the k-nearest neighbors gradient flow which attempts to approximate the local differentiable structre of the manifold from which the point cloud was sampled. Specifically, we try to approximate the normal bundle of the manifold around a point in the point cloud. Such an approach has been used in surface reconstruction problems and in this work, we extend this technique to higher dimensions. Once the normal bundle has been approximated, we project the k-nearest neighbor gradient vector field onto the approximated normal bundle, thus restricting movement of the point cloud to the normal directions of the sampled manifold. Furthermore, we introduce a novel way to use the approximated tangent vectors (i.e. those orthogonal to the approximated normal bundle) to force the point cloud to spread out and avoid clusters forming during the flow, a common

problem with the k-nearest neighbor gradient flow.

Since the distance-to-measure function is not the only function used to measure the distance between a point and a distribution, we also consider the Mahalanobis distance. This function, often used in statistics, serves a similar purpose to the distance-to-measure function. Thus, it seems natural to consider what happens to a point cloud under the gradient flow induced by the Mahalanobis distance. To this end, we introduce the Mahalanobis flow and investigate some of its properties. To our knowledge, this is the first investigation in the relevant literature of the Mahalanobis flow.

As it turns out, taking the gradient and projecting it along the approximated normal vectors induces a gradient flow which holds a strong connection to the flow induced by the Mahalanobis distance. In particular, comparing the projected distance-to-measure gradient with the gradient obtained from the Mahalanobis distance, it becomes clear that the projected distance-to-measure gradient flow is in some sense a differently weighted version of the Mahalanobis flow. This is due to the fact that the projected distance-to-measure gradient only permits motion along the approximated normal directions. On the other hand, the Mahalanobis gradient allows motion in all directions. Thus, while the distance-to-measure flow attempts to restrict flow along the normal directions, the Mahalanobis flow allows flow in all directions, but scales the amount of flow in each direction by the corresponding directional variance.

All of the above described techniques operate on the point cloud in a uniform fashion. However, it is often important to take local information, such as the curvature of the manifold, into account when evolving the point cloud. Techniques suggested for adaptively smoothing point clouds often only work in low dimensions because they rely on approximating the medial axis of a point cloud. To approximate this geometric object requires expensive operations such as the Voronoi diagram or sphere enlarging techniques (which require computing intersections of spheres). In this work, we introduce a new method for approximating the local curvature of a high dimensional point cloud. This method is based simply on distance calculations and one dimensional curve fitting and therefore scales to

high dimensions.

Finally, we give numerical results on the effectiveness of the above algorithms. We investigate the optimal selection of parameters: the neighborhood size k and the run time of the smoothing. To determine the *optimal* parameter values, we will use two notions of error. First, we will determine the geometric error in the smoothed point cloud. That is, how far do the points lie from the sampled manifold. Second, we will determine the topological error. That is, if we build a simplicial complex from the evolved point cloud, how topologically similar is this complex to the sampled manifold? By considering both notions of error, we will be sure to determine the correct parameter values for both geometric and topological reconstruction algorithms.

All numerical results in this work are obtained using the *GradSmooth* C++ library and the *TopoPack* python package, both created for this dissertation. *GradSmooth* smooths arbitrary dimensional point clouds using either the k-nearest neighbor flow or the normal bundle flow. It also contains all the modifications we develop in Chapter 4, allowing for experimentation with combinations of all the approaches. Furthermore, this implementation is multi-threaded, allowing for large point clouds to be processed quickly. *TopoPack* contains routines for topologically and geometrically analyzing the results of the k-nearest neighbor flow and the normal bundle flow. This package also allows one to analyze a two dimensional k-order Voronoi diagram and construct a flow diagram from the higher order Voronoi diagram. For all of the persistence diagram calculations, including those in *TopoPack*, we utillize Dionysus, a C++ library written by Dmitriy Morozov [41].

1.2 Manifold Sampling

Let M be an m-codimensional manifold in $\mathbb{R}^N$ with $0 < m < N$. The empirical measure ν_M on the manifold M as defined in [14] is given by the rescaled volume form on M. Sampling from the measure ν_M yields a uniformly dense sampling of the manifold M. When we encounter multiple submanifolds $M_1, ..., M_\ell \subset \mathbb{R}^N$ of varying intrinsic dimension, we can

define a measure ν on $\mathcal{M} = \cup_{i=1}^{\ell} M_i$ given by

$$\nu = \sum_{i=1}^{\ell} \lambda_i \nu_{M_i}$$

where $\{\lambda_i\}_{i=1}^{\ell}$ are constants summing to one. Noise can be added to this model by convolving ν with a Gaussian distribution, that is

$$\mu = \nu \star \mathcal{N}(0, \sigma^2)$$

where $\sigma > 0$ is a noise parameter known as the standard deviation of the noise.

Throughout this dissertation, the term **point cloud** will be used to denote a finite set $\mathcal{X}$ of points in $\mathbb{R}^N$ for some $N > 0$ with $N < |\mathcal{X}|$ which are in general position. That is, there exists no hyperplane in $\mathbb{R}^N$ containing more than N points of $\mathcal{X}$. We consider the point clouds to be drawn from a manifold M using the measure μ as just constructed. If $\sigma = 0$, then all of the sampled points lie directly on the manifold X. However, if $\sigma > 0$, then the point cloud is noisy and therefore our algorithms must be robust to noise.

Although the point cloud is generated from the probability measure μ, in practice, the measure μ is unknown. Therefore, we must approximate the measure from a point cloud $\mathcal{X}$. Suppose $\mathcal{X} = \{x_1, ..., x_n\} \subset \mathbb{R}^N$ is drawn independently and identically distributed (i.i.d.) according to some probability measure μ and let $\mu_{\mathcal{X}}$ be the **empirical probability measure** on $\mathcal{X}$. That is, for any Borel subset $B \subset \mathbb{R}^N$, we define

$$\mu_{\mathcal{X}}(B) = \frac{1}{n}|B \cap \mathcal{X}|$$

This probability measure is often used when infering an underlying distribution since, by the uniform law of large numbers, $\mu_{\mathcal{X}}$ converges to μ with probability 1 when $n \to \infty$ [14]. The fact that $\mu_{\mathcal{X}}$ converges to μ allows us to approximate the underlying distribution,

however we often have a fixed number of samples. Therefore, we seek a way of reducing the error in our approximation without increasing the sample size. To accomplish this, we will induce gradient flows which are designed to move the points of $\mathcal{X}$ closer to the underlying manifold which generated the samples.

It is important to note that although many of the methods presented in this work do not require the probability measure in question to be generated by a manifold, it is often more intuitive to consider our samples as being drawn in this way. We will make clear whenever the manifold assumption is required. As we will see, it can be difficult to recover the geometry of M from the point cloud $\mathcal{X}$, however the topology of M is more readily discovered.

For some of the error bounds that will be established later, we must make additional assumptions on the sampling. These assumptions will apply to point clouds sampled from surfaces in $\mathbb{R}^3$ and rely on the notion of local feature size. Before we give the definition of local feature size, recall that the medial axis of a surface S is the set of all points $y \in \mathbb{R}^3$ such that the set $\{z \in S : d(y,z) = d(y,S)\}$ has cardinality greater than one.

Definition 1.1. Let $S \subset \mathbb{R}^3$ be a smooth surface. The **local feature size** of a point $x \in S$ is given by $\text{lfs}(x) = d(x, M)$ where M is the medial axis of S.

As noted in [22], the function $\text{lfs}(\cdot)$ is 1-Lipschitz. We can use the local feature size to approximate the size of geometric features of the surface near a point. In particular, in the case where S is a sphere of radius r, the medial axis is simply the center of the sphere and $\text{lfs}(x) = r$ for every point $x \in S$. Intuitively, in a region with a large local feature size, we can perform more substantial smoothing than we can with a smaller feature size, where we would like to preserve the small features of the data. As we will see, this corresponds to taking a larger value of h in Equation 1.2 (see page 12) when the local feature size is large, and taking a smaller value for h when it is small. The following sampling condition is a noisy version of the sampling conditions found in [9]. This particular form can be found in [22].

Definition 1.2. For a surface S and a point $x \in \mathbb{R}^3$, let $\tilde{x} \in S$ be the closest point of the surface to x. A point cloud $P \subset \mathbb{R}^3$ is a **noisy** (ϵ, α)-**sample** of a surface S if the following hold

(a) For every $z \in S$, $d(z, P) < \epsilon \operatorname{lfs}(z)$

(b) For every $p \in P$, $d(p, S) < \epsilon^2 \operatorname{lfs}(z)$ where z is the closest point in S to p.

(c) Every point $p \in P$ is equipped with a normal vector v_p where the angle between v_p and the normal $n_{\tilde{p}}$ is less than ϵ, where $n_{\tilde{p}}$ is the surface normal at $\tilde{p}$.

(d) For every $x \in \mathbb{R}^3$,

$$|B(x, \epsilon \operatorname{lfs}(\tilde{x})) \cap P| < \alpha$$

where $B(c, r)$ is the ball of radius r centered at $c \in \mathbb{R}^3$.

1.3 Wasserstein Distances

Discussing convergence of probability measures requires a metric space setting. Therefore, we will use the Wasserstein distances, $W_p(\cdot, \cdot)$, to turn the set of all probability measures into a metric space. As noted in [8], the metric W_p is also known as the Kantorovich distance and was introduced as far back as 1958 [34]. The association with Wasserstein (correct transliteration Vasershtein) is due to a paper by Dobrushin [25], which incorrectly cited Vasershtein's paper [54] as the source of the metric. Historical inconsistencies aside, we will refer to W_p as the Wasserstein metric throughout this paper to remain consistent with modern literature.

The definition of the Wasserstein distance is rooted in optimal transport theory. In particular, for two Radon probability measures μ and ν on $\mathbb{R}^N$, a **transport plan** between μ and ν is a Radon probability measure π on $\mathbb{R}^N \times \mathbb{R}^N$, with marginals μ and ν. That is, for all $A, B \subseteq \mathbb{R}^N$, we have $\pi(A \times \mathbb{R}^N) = \mu(A)$ and $\pi(\mathbb{R}^N \times B) = \nu(B)$. For each transport

plan, we can associate a cost $C_p(\pi)$, and then optimal transport theory is concerned with finding the transport plan π of least cost. This is made precise in the following definitions.

Definition 1.3. Given $p \geq 1$, the p-cost of a transport plan π is defined as

$$C_p(\pi) = \left(\int_{\mathbb{R}^N \times \mathbb{R}^N} \|x - y\|^p d\pi(x, y) \right)^{1/p}$$

The **Wasserstein distance** (of order p), denoted $W_p(\mu, \nu)$, between two Radon measures μ and ν on $\mathbb{R}^N$ with finite p-moment is defined as

$$W_p(\mu, \nu) = \inf_{\pi} C_p(\pi)$$

where the infimum is taken over all transport plans π.

For general Radon measures μ, ν on $\mathbb{R}^N$, with finite p-moments, i.e. $\int_{\mathbb{R}^N} \|x\|^p d\mu(x) < \infty$ and $\int_{\mathbb{R}^N} \|x\|^p d\nu(x) < \infty$, we are ensured a finite p-cost for any transport plan π as shown in [8]. The Wasserstein distance is also symmetric, $W_p(\mu, \nu) = W_p(\nu, \mu)$, and $W_p(\mu, \nu) = 0$ only if $\mu = \nu$. Finally, the triangle inequality is verified in [8]. Thus, the Wasserstein distance is in fact a metric and hence $(\mathcal{P}(\mathbb{R}^N), W_p)$ forms a metric space over the set of Radon probability measures $\mathcal{P}(\mathbb{R}^N)$

As alluded to above, the empirical measure $\mu_{\mathcal{X}}$ on a point cloud $\mathcal{X}$ sampled from a measure μ converges to μ as $|\mathcal{X}| \to \infty$ with probability one. The following result shows that this general convergence statement holds in the W_p norm. This, and the two results following, can be found in [15].

Proposition 1.4. *If a measure μ is concentrated on a compact set, then $\mu_{\mathcal{X}}$ converges almost surely to μ in the W_p distance as $|\mathcal{X}| \to \infty$.*

As our model contains a convolution with a noise term, we would like to verify a similar result holds under the presence of noise. As a first step, we have the following result which

relates the convolved measure to the original measure.

Proposition 1.5. *If $\nu : \mathbb{R}^N \to \mathbb{R}^+$ defines a probability distribution with finite p-moment $\sigma^p = \int_{\mathbb{R}^N} \|x\|^p \nu(x)dx$, then*

$$W_p(\mu, \mu \star \nu) \leq \sigma$$

Putting these results together yields a powerful result on the convergence of the empirical measure of a noisy point cloud, $\mu_{\mathcal{X}}$, to the underlying generating measure ν.

Corollary 1.6. *If we consider the empirical measure $\mu_{\mathcal{X}}$ as described above, constructed from an underlying geometric measure ν convolved with a Gaussian distribution $\mathcal{N}(0, \sigma)$, we find that*

$$\lim_{N \to \infty} W_2(\mu_{\mathcal{X}}, \mu) \leq \sigma \quad \textit{with probability } 1$$

where $\mu = \nu \star \mathcal{N}(0, \sigma)$.

Therefore, we can bound the distance between the empirical measure, $\mu_{\mathcal{X}}$, and the generating distribution μ, by the noise σ present in the model. This implies that the distance between the measures converges to zero as we reduce the noise, an important, albeit completely unsurprising, result.

1.4 Surface Reconstruction

As mentioned in the introduction, point clouds are often used to represent surfaces in $\mathbb{R}^3$. In this setting, one seeks to reconstruct the surface of an object from a point cloud which is assumed to have been drawn from a probability distribution concentrated around the surface. Such point clouds may be drawn from 3d scanners such as LiDAR systems (which mesaures distance using laser light), or x-ray computed tomography (which takes x-ray images at several angles to produce a 3d representation), to name a few examples. Due to the dimensionality assumptions, many surface reconstruction methods rely on approximating a normal vector to the surface at each of the sample points. When it comes to constructing the

approximating surface, many techniques have been proposed such as radial basis functions [13], multi-level partitions of unity [44], and natural neighbor interpolation of the distance function [9]. One technique of particular interest to this work is that of the *moving least squares* [4].

The method of moving least squares computes normal vectors and a bivariate polynomial at each point. The bivariate polynomial is meant to approximate the surface of the object. Once the polynomial is obtained, the original point cloud is projected onto the polynomials in the direction of the computed normal, with the goal of reducing the noise inherent in the sample. Additionally, once one obtains the polynomials, it is straightforward to upsample or downsample the point cloud by adding or removing points along the polynomial. Of course, the dimensionality assumptions only come into play with the choice to use a bivariate polynomial and a single normal vector. To scale the technique to higher dimensions, one would use a multivariate polynomial and approximate the normal bundle at each point, instead of just a normal vector. However, fitting a multivariate polynomial becomes difficult in higher dimensions, as we will see. For now, we will focus on surfaces in $\mathbb{R}^3$.

Suppose we have a smooth surface $S \subset \mathbb{R}^3$ and we have drawn points $\mathcal{X} = \{p_1, ..., p_n\}$ from S with some level of noise. The goal of the moving least squares method is to construct an approximation of S from the points $\mathcal{X}$, which we will denote $S_{\mathcal{X}}$. Given a point $r \in \mathbb{R}^3$, we would like to project r down to the surface S, that is we would like to find a point $q \in \mathbb{R}^3$ which is approximately the projection of r onto S. The first step is to approximate a tangent plane to the surface S near the point r. That is, we compute a local plane $H = \{x | \langle n, x \rangle - D = 0, x \in \mathbb{R}^3\}$ with $n \in \mathbb{R}^3$, $\|n\| = 1$, and D the distance to the origin, by minimizing a weighted sum of squared distances of the sample points $p_i \in \mathcal{X}$ to the plane H. In particular, we find H by minimizing

$$\sum_{i=1}^{N} (\langle n, p_i \rangle - D)^2 \theta(\|p_i - q\|)$$

where θ is a smooth weighting function and q is the projection of r onto H via the normal vector n. To be precise, we set $q = r + tn$ for some $t \in \mathbb{R}$, yielding

$$\sum_{i=1}^{N} \langle n, p_i - r - tn \rangle^2 \theta(\|p_i - r - tn\|) \tag{1.1}$$

In practice, a Gaussian weight is often used. That is, we set

$$\theta(d) = e^{-\frac{d^2}{h^2}} \tag{1.2}$$

where h determines how quickly the influence of points decreases with their distance to q. We will discuss the choice of h shortly, however for now just consider it to be a parameter of the smoothing algorithm. Finally, define $\mathcal{P}(r) = q = r + tn$ to be the local minimum of Equation 1.1 with the smallest t and local tangent plane H near r. That is, $\mathcal{P}$ projects points onto the approximate tangent plane of S near r.

From here, we compute the bivariate polynomial which approximates the surface S. We let $q_i = \mathcal{P}(p_i)$ for $p_i \in \mathcal{X}$ and set $f_i = n \cdot (p_i - q_i)$, the height of p_i over the tangent plane H_i. Let g be a bivariate polynomial (whose order is yet to be specified), which minimizes the weighted least squares error

$$\sum_{i=1}^{N} (g(x_i, y_i) - f_i)^2 \theta(\|p_i - q_i\|) \tag{1.3}$$

where (x_i, y_i) represents q_i in the local coordinate system specified by H. We define the surface projection map $\mathscr{P}$, which projects points r onto the approximated surface $S_{\mathcal{X}}$, by $\mathscr{P}(r) = q + g(0,0)n = r + (t + g(0,0))n$. As noted in [5], the map $\mathscr{P}$ is in fact a projection map, which implies that the set and the order of the points we project does not change the surface. Finally, the approximated surface $S_{\mathcal{X}}$ is defined as the set of all points in $\mathbb{R}^3$ which project to themselves under $\mathscr{P}$.

To compute the actual projection, we must minimize Equation 1.1, which in general has many local minima. However, since we are taking the minimum with the smallest t, we assume the plane must be near r. An iterative scheme is employed in [5], wherein they start with $t = 0$ and approximate the normal vector. Thus, the weights $\theta_i = \theta(\|p_i - r\|)$ are fixed. Let $B \in \mathbb{R}^{3\times 3}$ be the r-centered weighted covariance matrix, that is $B = \{b_{jk}\}$ with

$$b_{jk} = \sum_{i=1}^{N} \theta_i \left(p_{i_j} - r_j\right)\left(p_{i_k} - r_k\right)$$

Then using B, we can rewrite Equation 1.1 as the following bilinear minimization problem

$$\min_{\|n\|=1} n^T B n$$

whose solution is given by the eigenvector of B corresponding to the smallest eigenvalue of B. From the solution to this problem, we can see that approximating normals is similar to local principal component analysis (PCA) where we locally find the directions of greatest variance and project the data onto these dimensions, thereby performing dimensionality reduction. Unlike PCA, instead of performing dimensionality reduction on the data, we treat the direction with least variance as the normal direction. In this sense, scaling up the normal approximation to higher dimensions is simple: instead of taking the single eigenvector corresponding to the smallest eigenvalue, we take the m eigenvectors corresponding to the m smallest eigenvalues, where m is the codimension of the object of study.

Once we have computed the normal vector for $t = 0$, we fix this vector and minimize Equation 1.1 with respect to t. As noted in [5], this is a non-linear optimization problem which could have many local minimia. However, in practice, there is often only one local minimum when $t \in [-h/2, h/2]$ where h is the same smoothing parameter used in Equation 1.2. Therefore, we constrain t to fall within this interval. Then the partial derivative can

be computed as

$$2\sum_{i=1}^{N}\langle n, p_i - r - tn\rangle \left(1 + \frac{\langle n, p_i - r - tn\rangle^2}{h^2}\right) e^{\|p_i - r - tn\|^2/h^2} \tag{1.4}$$

and following an iterative minimization scheme as described in [5], the projection $\mathcal{P}(r)$ can be found. With the projection and normal computed, we have the tangent plane approximation H. From here, we are left with a linear least squares optimatization problem to compute the polynomial g. This yields a system of linear equations where we have the same number equations as we have coefficients of g. Of course, for a degree three bivariate polynomial, this yields a system of 10 equations. However, for a polynomial of degree d with n variables, we must solve a system of

$$\binom{n+d}{d}$$

equations. Clearly for high dimensions, this system becomes enormous. Therefore, fitting a polynomial to represent the surface of an object becomes problematic when we are dealing with high dimensional data. Later, we will discuss a gradient flow based modification of this algorithm which works in arbitrary dimensions since its complexity is much lower with respect to the dimensionality of the underlying manifold.

The above procedure relies on a single parameter, $h > 0$, in the Gaussian weighting function. The lower this parameter value, the less smoothing will occur. Obviously, there are cases where setting h globally does not make sense. For example, suppose our data consists of two spheres S_1 and S_2 where the radius of S_1 is much smaller than the radius of S_2. Under a global parameter h, we can either set h to apply an appropriate level of smoothing to S_1 or we can set it to appropriately smooth S_2 by using a larger value of h. However, with a large enough value of h, the smaller sphere S_1 will collapse toward the center of S_1 since in this case, every point sampled from S_1 would contribute a significant

amount to the calculation of the tangent plane. This would cause the computed barycenter to be near the circle's center. Thus, when the polynomial is fitted, it will run through this barycenter and hence, through the center of the circle. When the points of S_1 are projected onto the polynomial, they will fall toward the center of the circle.

Such problems suggest an adaptive approach to setting the smoothing parameter h. Tamal Dey and Jian Sun consider exactly this in their work [22] which uses moving least squares to approximate the surface, but uses the local feature size, as defined in Section 1.2, to adapt the smoothing parameter h. In particular, for a point $x \in \mathbb{R}^3$, they take the weight θ_p associated with a point $p \in \mathcal{X}$ to be

$$\ln \theta_p(x) = -\frac{\|x-p\|^2}{\rho_e^2 f(\tilde{p}) f(\tilde{x})}$$

where $\tilde{x}$ and $\tilde{p}$ are the projections of x and p onto the surface S respectively, ρ_e controls the level of smoothing, and the function f, described in a moment, is a smooth approximation of $\text{lfs}(\cdot)$. From the definition, it is clear that larger values of ρ_e induce greater smoothing. Turning to the definition of f, since the local feature size depends on the medial axis, it is not smooth on the surface S. In particular, it fails to be smooth when S intersects the medial axis of its own medial axis. To overcome this issue, [22] uses the fact that smooth real-valued functions over smooth manifolds are dense in the family of continuous functions to construct a smooth approximation $f(\cdot)$ of $\text{lfs}(\cdot)$ where for arbitrarily small β, we have

$$|f(x) - \text{lfs}(x)| < \beta \, \text{lfs}(x)$$

for all $x \in \mathbb{R}^N$. The paper goes on to define the surface implictly as the zero level set of

$$\mathcal{N}(x) = \sum_{p \in \mathcal{X}} \langle x - p, v_p \rangle \theta_p(x)$$

where θ_p is defined using the adaptive weights. One benefit of this approach is that smaller

geometric features will be preserved under the smoothing operation so long as ρ_e is set properly. Of course, to use this approach, one needs to compute some approximation of the medial axis of the surface. In [21], they use the Voronoi diagram to approximate the medial axis for a surface in $\mathbb{R}^3$. Later, in Chapter 4, we will develop a new adaptive smoothing technique. Instead of using the local feature size of an object, we will approximate the curvature of the object and set our weights accordingly. The motivation behind this approach is that for higher curvature regions of the manifold, we would like to use fewer neighbors to approximate the tangent space. Additionally, since computation of high dimensional Voronoi diagrams is expensive [50], we will need an approach that works in higher dimensions, with lower computational complexity.

Although all of the above techniques were designed for surface reconstruction, they can be applied to high dimensional data using the modifications discussed. However, in doing so, we are bound to run into computational complexity issues due to the exponential reliance on the dimension of the data. Therefore, methods for smoothing data and reconstructing surfaces which scale polynomially with the dimensionality of the data are required. In Chapter 4 we will develop a smoothing technique that scales much better with the dimension of the data set. To do so, we will use the normal bundle approximation techniques described in this section and induce a gradient flow that respects the approximated surface normals. Since the gradient we introduce is trivial to compute in arbitrary dimensions once the nearest neighbors are computed, the polynomial fitting issues are completely avoided.

1.5 Distance-to-Measure

Many results in computational geometry use the distance function. Given a compact set $K \subset \mathbb{R}^N$, the distance, $d_K(x)$, of a point $x \in \mathbb{R}^N$ to the set K is defined in the usual way,

$$d_K(x) = \inf_{y \in K} d(x, y)$$

Note that the zero level set of d_K is simply the set K. Thus, the set d_K contains geometric information about the set K. By considering the sublevel sets of this function, we can construct the r-offsets of K, denoted K^r, according to the definition

$$K^r = \{x \in \mathbb{R}^N : d_K(x) < r\}$$

As just noted, since $d_K(x) \geq 0$ for all $x \in \mathbb{R}^N$, the 0-offset, K^0, is simply the original set K. Although the r-offsets contain a great deal of geometric information, they are extremely sensitive to noise. For example, adding a single isolated point $x \in \mathbb{R}^N \backslash K$ to the set K will completely change the geometry and topology (adding a new connected component) of the r-offsets. Therefore, inferring topological and geometric information directly from the r-offsets fails under the presense of noise. Thus, the traditional distance function is not ideal for applications involving noisy point clouds. To alleviate these concerns, we turn to the notion of the distance-to-measure function as defined in [15]. This function is similar to the distance function, however it is robust to noise. Before we define the distance-to-measure function, we first present a weaker notion of this distance, aptly referred to as the pseudo-distance function.

Definition 1.7. Let μ be a probability measure on $\mathbb{R}^N$. For $m \in [0, 1]$, we define the **pseudo-distance** to the measure μ to be the map

$$\delta_{\mu,m} : x \in \mathbb{R}^N \mapsto \inf\{r > 0 : \mu\left(\bar{B}(x, r)\right) > m\}$$

As the name suggests, the pseudo-distance function is not what we would typically consider a distance function. However, for $m = 0$, the definition of $\delta_{\mu,m}$ reduces to that of the distance function to the support of μ. On the other hand, as m approaches one, we see that $\delta_{\mu,m}$ grows, bounded by the diameter of the support of μ. Of course, since μ is a probability measure, we can never have $\mu(\bar{B}(x, r)) > 1$ for any radius r. Thus, if the

support of μ is all of $\mathbb{R}^N$, then $\delta_{\mu,m}$ approaches infinity as m approaches one. We take the L^2 average of $\delta_{\mu,m}$ over a range $[0, m_0]$ of the parameter m to gain both Wasserstein-stability and some level of regularity, .

Definition 1.8. Let μ be a probability measure on $\mathbb{R}^N$ and let $m_0 \in (0, 1]$. Define the **distance function to μ with parameter** m_0 to be the function $d_{\mu,m_0} : \mathbb{R}^N \to \mathbb{R}^+$ where

$$d^2_{\mu,m_0}(x) = \frac{1}{m_0} \int_0^{m_0} \delta_{\mu,m}(x)^2 dm$$

As before, this function measures distance to an underlying measure, however it now takes a multi-scale view of the data. Since a point cloud is a zero-dimensional subset of $\mathbb{R}^N$ and is therefore inherently of measure zero, this multi-scale property is important as we do not know the correct scale a priori.

Having defined the distance-to-measure function, we now turn to the stability properties of d_{μ,m_0}. One of the key benefits of moving to the distance-to-measure function over the pseudo-distance function is that we now have Wasserstein stability for $p = 2$ under the L^∞ norm as shown in [15].

Theorem 1.9. *If μ and ν are two probability measures on $\mathbb{R}^N$ and $0 < m_0 \leq 1$, then*

$$\|d_{\mu,m_0} - d_{\nu,m_0}\|_{L^\infty(\mathbb{R}^N)} \leq \frac{1}{\sqrt{m_0}} W_2(\mu, \nu)$$

Note that this dependence scales with the reciprocal of $\sqrt{m_0}$. In particular, as we send $m_0 \to 0$, we see that the upper bound approaches infinity and the strength of this statement disappears.

For the point cloud case, combining this theorem with Corollary 1.6 gives the following estimate relating the functions of $\mu_{\mathcal{X}}$, the uniform distribution of $\mathcal{X}$, and the generating

measure μ.

Corollary 1.10. *Let $\mathcal{X} \subset \mathbb{R}^N$ be a noisy point cloud drawn from a probability measure $\mu = \nu \star \mathcal{N}(0, \sigma)$ and define $\mu_{\mathcal{X}}$ to be the uniform distribution of $\mathcal{X}$. Then we have the following estimate*

$$\lim_{|\mathcal{X}|\to\infty} \|d_{\mu_{\mathcal{X}},m_0} - d_{\mu,m_0}\|_{L^\infty(\mathbb{R}^N)} \leq \frac{1}{\sqrt{m_0}}\sigma$$

From this estimate, we see that for fixed $0 < m_0 < 1$, the difference between $d_{\mu_{\mathcal{X}},m_0}$ and d_{μ,m_0}, as we grow the sample size of the point cloud $\mathcal{X}$, approaches a multiple of the noise inherent in the model. This is to be expected since $\mu_{\mathcal{X}}$ converges to $\mu = \nu \star \mathcal{N}(0, \sigma)$.

Furthermore, we can deduce regularity results from the definition of $d_{\mu,m}$. In particular, we see that the function is Lipschitz and the square of the function is 1-semiconcave. First, we remind the reader of the definition of semiconcave functions, as given in [12].

Definition 1.11. Let $A \subset \mathbb{R}^n$ be an open set. We say that a function $u : A \to \mathbb{R}$ is **semiconcave with linear modulus** if it is continuous in A and there exists $C \geq 0$ such that

$$u(x + h) + u(x - h) - 2u(x) \leq C|h|^2$$

for all $x, h \in \mathbb{R}^N$ such that $[x - h, x + h] \subset A$. The constant C is called a **semiconcavity constant** for $u \in S$.

As the name implies, semiconcave functions are related to concave functions. Additionally, they are almost C^2 in the sense made precise below. We can further classify semiconcave functions with the following proposition from [12].

Proposition 1.12. *Given $u : A \to \mathbb{R}$ with $A \subset \mathbb{R}^N$ open and convex, and given $C \geq 0$, the following are equivalent:*

(a) u is semiconcave with linear modulus in A with semiconcavity constant C;

(b) u satisfies

$$\lambda u(x) + (1-\lambda)u(y) - u(\lambda x + (1-\lambda)y) \leq \frac{1}{2}C\lambda(1-\lambda)|x-y|^2$$

(c) The function $x \mapsto u(x) - \frac{1}{2}C|x|^2$ is concave in A;

(d) There exist two functions $u_1, u_2 : A \to \mathbb{R}$ such that $u = u_1 + u_2$ where u_1 is concave and $u_2 \in C^2(A)$ and satisfies $\|D^2 u_2\|_\infty \leq C$, where $D^2 u_2$ is the Hessian of u_2;

(e) For any $v \in \mathbb{R}^n$ such that $|v| = 1$, we have $\frac{\partial^2 u}{\partial^2 v} \leq C$ in A in the sense of distributions, i.e.

$$\int_A u(x)\frac{\partial^2 \phi}{\partial v^2}(x)dx \leq C\int_A \phi(x)dx$$

(f) u can be represented as $u(x) = \inf_{i\in\mathcal{I}} u_i(x)$ where $\{u_i\}_{i\in\mathcal{I}}$ is a family of functions in $C^2(A)$ such that $\|D^2 u_i\|_\infty \leq C$ for all $i \in \mathcal{I}$.

From the above definitions, we gather that d_K^2 is semiconcave on $\mathbb{R}^N$. This is proven in [12], where the authors observe that

$$d_K^2(x) - \|x\|^2 = \inf_{y\in K} \|x-y\|^2 - \|x\|^2 = \inf_{y\in K} \|y\|^2 - 2\langle x, y\rangle$$

by the definition of d_K. Hence, since the infimum of linear functions is concave, by Proposition 1.12 we see that d_K^2 must be semiconcave. Furthermore, it is semiconcave with semiconcavity constant 2. In [12], the authors also establish that d_K is semiconcave on $\mathbb{R}^N \backslash K$. Thus, since the distance function is semiconcave, we naturally desire the distance-to-measure function to exhibit semiconcavity as well. In fact, this will be a requirement for functions classified as *distance-like*, which we define in Section 1.6.

Of course, the square of the distance-to-measure function, d_μ^2, satisfies property (c) and

is therefore semiconcave with linear modulus as pointed out in [15]. Property (e) shows that semiconcave functions are those functions whose second weak derivaties are bounded above (to be contrasted with concave functions whose second derivatives are nonpositive). Furthermore, [12] includes the following useful result.

Theorem 1.13. *A semiconcave function $u : S \to \mathbb{R}$ is locally Lipschitz continuous in the interior of S.*

As noted above, the square of the distance-to-measure function is semiconcave. We include this result from [15] which establishes that the square of the distance-to-measure function is in fact 1-semiconcave and describes some of the implications. Of particular interest to our focus is the result describing the gradient of d^2_{μ,m_0}. For this equation, we will need to define the set $\mathcal{R}_{\mu,m_0}(x)$, which is the set of all submeasures, μ_{x,m_0}, with $\mu_{x,m_0}(\mathbb{R}^N) = m_0$ and such that

$$\text{supp}(\mu_{x,m_0}) \subseteq \bar{B}(x, \delta_{\mu,m_0}(x)) \qquad \text{and} \qquad \mu_{x,m_0}\big|_{B(x,\delta_{\mu,m_0}(x))} = \mu$$

Note that in the first condition, the support is contained within the closed ball $\bar{B}(x, \delta_{\mu,m_0}(x))$ while in the second condition, we restrict the measure μ_{x,m_0} to the open ball $B(x, \delta_{\mu,m_0}(x))$. We see in [15] that $|\mathcal{R}_{\mu,m_0}(x)| = 1$ if and only if

$$\left|\text{supp}\left(\mu\big|_{\partial B(x,\delta_{\mu,m_0}(x))}\right)\right| \leq 1$$

We are now ready to state the semiconcavity results.

Corollary 1.14. *The function d^2_{μ,m_0} is 1-semiconcave. Moreover:*

(i) d^2_{μ,m_0} is differentiable at a point $x \in \mathbb{R}^N$ if and only if the suppport of the restriction of μ to the sphere $\partial B(x, \delta_{\mu,m_0}(x))$ contains at most one point.

(ii) d^2_{μ,m_0} *is differentiable almost everywhere in* $\mathbb{R}^N$ *with gradient defined by*

$$\nabla_x d^2_{\mu,m_0} = \frac{2}{m_0} \int_{h \in \mathbb{R}^N} [x - h] d\mu_{x,m_0}(h)$$

where μ_{x,m_0} *is the only measure in* $\mathcal{R}_{\mu,m_0}(x)$*.*

(iii) The function $x \in \mathbb{R}^N \mapsto d_{\mu,m_0}(x)$ *is* 1*-Lipschitz.*

The distance-to-measure function has proved useful for many applications since it is robust to noise, a common issue in manifold reconstruction. As we will see later, when we treat each point in the point cloud as a Dirac mass and use the empirical measure on the point cloud, the distance-to-measure function simplifies to a k-nearest neighbor function. For now, we stick with the general notion of distance-to-measure function and use this function to reconstruct manifolds from noisy point clouds, the details of which are presented in the next section.

1.6 Reconstruction Theory

Given a point cloud $\mathcal{X}$ sampled from some underlying measure μ, let K be the support of μ. We would like to infer the topology of K from the point samples in $\mathcal{X}$. To do this, we turn to the reconstruction method described by Chazal et al. in [15]. The theory presented in Chazal's work gives conditions for when the sublevel sets of the distance-to-measure function induced by $\mu_{\mathcal{X}}$, i.e. $d_{\mu_{\mathcal{X}},m_0}$, are homotopy equivalent to the offsets, $K^r = \{x \in \mathbb{R}^N : d_K(x) < r\}$, of the support of μ.

The results in this section provide important motivation for the work that follows in this dissertation. In particular, we are interested in the connection between the distance-to-measure function and our ability to *reconstruct* the measure from which a given point cloud was sampled. For our purposes, we are concerned with two probability measures μ and μ' where we view μ as a geometric measure on some subset of $\mathbb{R}^N$ and μ' as the empirical

measure associated with a point cloud noisily drawn from μ. In this setting, the ultimate result of this section is a condition for when the r-sublevel sets of d_{μ',m_0} and the offsets, $K^\eta = \{x \in \mathbb{R}^N : d_K(x) < \eta\}$, are homotopy equivalent, for appropriate values of r and η.

The condition for homotopy equivalence is based on the Wasserstein distance, $W_2(\mu, \mu')$, between the measures μ and μ'. Thus, if the two measures μ and μ' are close, then the sublevel sets and the offsets will be homotopy equivalent. As we saw in Section 1.2, if μ has Gaussian noise, the distance $W_2(\mu, \mu')$ is bounded, in the limit, by the standard deviation σ of the Gaussian noise component of μ, where $\mu = \nu \star \mathcal{N}(0, \sigma)$. We also saw in Section 1.5, when the two measure μ and μ' are close in the Wasserstein sense, the associated distance-to-measure functions will be close. Hence, when we attempt to induce a gradient flow which minimizes the distance-to-measure value of the points in a point cloud, we are effectively attempting to improve our ability to *reconstruct* the original sampling measure μ.

In this section, we review the reconstruction results of [15] for a more general class of functions, which exhibit properties similar to the traditional distance function d_K. This class of functions is referred to as the *distance-like functions*. All of the results in this section are established in [15].

Definition 1.15. A function $\varphi : \mathbb{R}^N \to \mathbb{R}$ is called a *distance-like* function if the following hold.

(a) $\varphi \geq 0$

(b) φ^2 is 1-semiconcave

(c) $\varphi(x) \to \infty$ as $|x| \to \infty$

Let φ be a distance-like function. Since φ^2 is 1-semiconcave, we can construct the gradient vector $\nabla_x \varphi$, defined for almost all $x \in \mathbb{R}^N$ as shown in [12]. In analogy to the r-offsets defined earlier, for a distance-like function φ, we let φ^r denote the r-sublevel set of φ. That is,

$$\varphi^r = \varphi^{-1}([0, r])$$

Many of the results in reconstruction theory rely on geometric measurements of the data. We recall the following definitions from [15].

Definition 1.16. Let φ be a distance-like function.

(a) Let $x \in \mathbb{R}^N$ such that

$$\varphi^2(x+h) \leq \varphi^2(x) + 2\alpha|h|\varphi(x) + |h|^2 \qquad \forall h \in \mathbb{R}^N$$

with $\alpha \in [0,1]$. Then x is an ***α*-critical point**.

(b) If $x \in \mathbb{R}^N$ is a 0-critical point, it is simply called a critical point. Additionally, $\varphi(x)$ is a **critical value**.

(c) Given $r \in [0,\infty)$, the **weak feature size** of φ at r, denoted $\text{wfs}_\varphi(r)$, is the maximum $s > 0$ such that there are no critical values of φ between r and $r+s$.

(d) The ***α*-reach** ($\alpha \in (0,1)$) of φ, denoted $\text{reach}_\alpha(\varphi)$, is the maximum r such that $\varphi^{-1}((0,r])$ contains no α-critical points.

A few consequences of the above definitions are established in [15]. For starters, since φ is a distance-like function and φ^2 is 1-semiconcave, we see that

$$\|\nabla_x \varphi\| = \inf\{\alpha \geq 0 : x \text{ is } \alpha\text{-critical}\}$$

Additionally, from the definitions alone it is concluded in [15] that the α-reach of φ is always bounded below by the weak feature size of φ at 0.

With the notion of α-critical points established, we can now establish the first result concerning the sublevel sets, $\{\varphi^r : r \in [0,\infty)\}$. In particular, the following lemma establishes when two sublevel sets φ^{r_1} and φ^{r_2} are isotopy equivalent. Recall, two topological spaces X and Y are isotopic (or isotopically equivalent) if X can be continuously deformed to Y, where there always exists a homeomorphism between the (deformed) X and Y. From

the definitions we see that isotopy is a stronger condition than homotopy, since a homotopy does not require the homeomorphism assumption.

Lemma 1.17. *Let φ be a distance-like function and $r_1 < r_2$ be two positive numbers such that φ has no critical points in the subset $\varphi^{-1}([r_1, r_2])$. Then all the sublevel sets $\varphi^{-1}([0, r])$ are isotopic for $r \in [r_1, r_2]$.*

We can establish a result for the weaker notion of homotopy equivalence when we place certain restrictions on the weak feature size of two distance-like functions as well as their difference under the L^∞ norm.

Proposition 1.18. *Let φ and ψ be two distance-like functions, such that*

$$\|\varphi - \psi\|_{L^\infty(\mathbb{R}^N)} \leq \epsilon$$

Suppose also that $\mathrm{wfs}_\varphi(r) > 2\epsilon$ *and* $\mathrm{wfs}_\psi(r) > 2\epsilon$. *Then for every $0 < \eta \leq 2\epsilon$, the offsets $\varphi^{r+\eta}$ and $\psi^{r+\eta}$ have the same homotopy type.*

This tells us that when the distance-like functions are close in the L^∞ sense, we can find offsets which are homotopically equivalent. However, we also see that the distance required between φ and ψ for this result to hold depends on the weak feature sizes of φ and ψ. This is not surprising since the topology of offsets can change drastically when the weak feature size is small.

Turning our attention to critical points, we see that when two distance-like functions are close in the L^∞ sense, the critical points cannot be too far apart. The condition $\|\varphi - \psi\|_{L^\infty(\mathbb{R}^N)}$ occurs frequently, hence we will call φ and ψ ϵ-close when this holds.

Proposition 1.19. *Let φ and ψ be two distance-like functions with $\|\varphi - \psi\|_{L^\infty(\mathbb{R}^N)} \leq \epsilon$ (i.e. let φ and ψ be ϵ-close). For any α-critical point x of φ, there exists an α'-critical point x' of ψ with*

$$\|x - x'\| \leq 2\sqrt{\epsilon\varphi(x)} \qquad \text{and} \qquad \alpha' \leq \alpha + 2\sqrt{\frac{\epsilon}{\varphi(x)}}$$

We are now ready for the first reconstruction theorem. Given two distance-like functions, we would like to know how close these functions need to be in the L^∞ sense to ensure the sublevel sets φ^r and ψ^r are homotopy equivalent for small enough r. As it turns out, we require a bound on the α-reach of one of the distance-like functions.

Theorem 1.20. *Let φ and ψ be two ϵ-close distance-like functions with*

$$\text{reach}_\alpha(\varphi) \geq R$$

for some $\alpha > 0$. Then for any $r \in [4\epsilon/\alpha^2, R - 3\epsilon]$ and $0 < \eta < R$, the sublevel sets ψ^r and φ^η are homotopy equivalent provided

$$\epsilon \leq \frac{R}{5 + 4/\alpha^2}$$

Now we turn our attention to the special case of the distance-to-measure function. Since distance-to-measure functions are distance-like functions, all the previous results hold. However, we can take the results a step further due to the additional constraints on the distance-to-measure function. We begin with some definitions describing the growth of a measure.

Definition 1.21. Let μ be a measure and let $f : \mathbb{R}^+ \to \mathbb{R}^+$ be a nondecreasing positive function such that for every point $p \in \text{supp}(\mu)$, and every $\epsilon > 0$, we have

$$\mu(B(p, \epsilon)) \geq f(\epsilon)$$

where $B(p, \epsilon)$ is the ball of radius ϵ centered at p. Then f is called a **uniform lower bound on the growth of μ**. The measure μ has **dimension at most k** if there exists a constant

$C(\mu)$ depending on μ such that for sufficiently small ϵ,

$$f(\epsilon) = C(\mu)\epsilon^k$$

is a uniform lower bound on the growth of μ.

In the following proposition, the function d_S is the distance to the set S defined in the traditional way. Here, we see how the distance-to-measure function d_{μ,m_0} relates to the distance function d_S where S is the support of μ.

Proposition 1.22. *(a) If $S = \text{supp}(\mu)$ is compact, then d_S is the uniform limit of d_{μ,m_0} as $m_0 \to 0$.*

(b) If μ has dimension at most $k > 0$, then

$$\|d_{\mu,m_0} - d_S\| \leq C(\mu)^{-1/k} m_0^{1/k}$$

Now let μ be a probability measure with compact support $K \subset \mathbb{R}^N$ and let $d_K : \mathbb{R}^N \to \mathbb{R}^+$ be the distance function to K, that is

$$d_K(x) = \inf_{y \in K} \|x - y\|$$

Given another probability measure μ', we have

$$\|d_K - d_{\mu',m_0}\|_{L^\infty(\mathbb{R}^N)} \leq \|d_K - d_{\mu,m_0}\|_{L^\infty(\mathbb{R}^N)} + \|d_{\mu,m_0} - d_{\mu',m_0}\|_{L^\infty(\mathbb{R}^N)}$$

$$\leq C(\mu)^{1/k} m_0^{1/k} + \frac{1}{\sqrt{m_0}} W_2(\mu, \mu')$$

where we have used the triangle inequality, Proposition 1.22, and Theorem 1.9. Then combining this inequality and Theorem 1.20 yields a useful corollary.

Corollary 1.23. *Let μ be a measure and K its support. Suppose that μ has dimension at most k and that* $\text{reach}_\alpha(d_K) \geq R$ *for some $R > 0$. Let μ' be another measure, and let ϵ be an upper bound on the uniform distance between d_K and d_{μ',m_0}. Then for any $r \in [4\epsilon/\alpha^2, R - 3\epsilon]$, the r-sublevel sets of d_{μ',m_0} and the offsets K^η, for $0 < \eta < R$, are homotopy equivalent, as soon as*

$$W_2(\mu, \mu') \leq \frac{R\sqrt{m_0}}{5 + 4/\alpha^2} - C(\mu)^{-1/k} m_0^{1/k+1/2}$$

Thus, we have conditions for when the offset K^η and the r-sublevel sets of d_{μ',m_0} are homotopy equivalent. In particular, we require the Wasserstein distance for $p = 2$ to be smaller than some function which depends on our choice of α, the α-reach of d_K, k, and the distance between d_K and d_{μ',m_0}.

Chapter 2: Topological Data Analysis

As the quantity of available data increases exponentially, increasing efforts are being made to make sense of this massive deluge of data. For geometric data, and in particular point cloud data, the field of topological data analysis has arisen as a means to systematically quantify topological aspects of a data set. Methods in this field seek to determine the topology of a given geometric data set using tools from algebraic topology. The focus on topological characterizations is important because many of these data sets are inherently noisy and thus present problems for algorithms attempting to determine geometric information from the data.

In this section, we will outline some of the basics of topological data analysis. In particular, we will establish the definitions of persistent homology, a tool which allows us to characterize the homology of a simplicial complex built from a point cloud. What makes persistent homology useful is the homological characterization is multiscale. That is, instead of obtaining a topological description of the data at a particular scale, we can see how the topology changes as we vary the scale. This proves useful since we often do not know the inherent scale of the data.

For our purposes, we will use topological data analysis and persistence diagrams to determine the topological accuracy of a point cloud, after we apply our smoothing algorithms. In particular, if we noisily sample a point cloud from a manifold whose topology is known to us, and then we apply our smoothing algorithms, how does the topology of a manifold constructed from the smoothed point cloud match the topology of the sampled manifold.

2.1 Simplicial Approximations

Complex shapes are hard to analyze mathematically. Furthermore, most continuous shapes cannot even be represented on a computer and therefore provide little room for computational analysis of the shape directly. Therefore, one often represents topological spaces using approximations in the form of complexes. Of particular interest, due to their simplicity, are simplicial complexes. These complexes consist of simplices of varying dimensions. A k-simplex is a k-dimensional polytope which is the convex hull of $k+1$ affinely independent points. To form a simplicial complex, one must ensure that the intersection of two simplices of the complex is another (lower dimensional) simplex of the complex.

Simplicial complexes have found a wide range of applications in mathematics, including applications in algebraic topology, computational geometry, and the numerical analysis of partial differential equations. Here, we will create simplicial complexes from point cloud data and treat the resulting complex as an approximation of the underlying geometric measure. Many of these methods can be classified under the field of computational topology. First, we give some examples of common simplicial complexes arising from analyzing point clouds.

Maybe the most straightforward method of constructing a simplicial complex from a set of points is to form what is known as the **Čech complex** of a point cloud $\mathcal{X}$. This complex, whose definition is given in [26], will be denoted $\check{\text{C}}\text{ech}_{\mathcal{X}}(r)$. First, let $B_x(r)$ denote the closed ball of radius $r > 0$ centered at $x \in \mathbb{R}^N$. Given some threshold $r > 0$, the n-simplex $\sigma = (p_0, ..., p_n)$ formed from the points $p_0, ..., p_n \in \mathcal{X}$, is contained in $\check{\text{C}}\text{ech}\,\mathcal{X}(r)$ if and only if the closed balls of radius r centered at the points $p_0, ..., p_n$ have non-empty intersection. That is,

$$\check{\text{C}}\text{ech}_{\mathcal{X}}(r) = \left\{ \sigma \subset \mathcal{X} : \bigcap_{p \in \sigma} B_p(r) \neq \emptyset \right\}$$

An important point to make here is that for $0 < r_1 < r_2$, we will have $B_x(r_1) \subseteq B_x(r_2)$ for all $x \in \mathcal{X}$ and so $\check{\text{C}}\text{ech}_{\mathcal{X}}(r_1) \subseteq \check{\text{C}}\text{ech}_{\mathcal{X}}(r_2)$. When we turn to the notion of filtrations, this property will be crucial. All of the other simplicial complexes presented in this section

will share this property. The simplicial complex $\check{\text{C}}\text{ech}_{\mathcal{X}}(r)$ is actually the nerve of the balls $\{B_x(r) : x \in \mathcal{X}\}$. Recall the definition of the nerve of a covering: given a covering $\mathcal{U} = \{U_i\}$ of a topological space X where I is an index set, the **nerve** of the covering $\mathcal{U}$ is the set of all finite subsets J of I such that

$$\bigcap_{j \in J} U_j \neq \emptyset$$

Studying the nerve of the union of closed balls and studying the union itself are the same up to homotopy type due to the nerve theorem, which is given in [26].

Theorem 2.1. *Let F be a finite collection of closed, convex sets in Euclidean space. Then the nerve of F and the union of the sets in F have the same homotopy type.*

Although the Čech complex can be used to accurately respresent the homotopy type of the offsets $\mathcal{X}^r$, in practice one must consider all subcollections of the points in $\mathcal{X}$. This can be expensive for large point clouds. A simplification of the Čech complex can be found by forming the **Vietoris-Rips complex**, denoted $\text{VR}_{\mathcal{X}}(r)$. This construction allows us to simply consider pairs of points. Explictly, we define

$$\text{VR}_{\mathcal{X}}(r) = \{\sigma \subseteq \mathcal{X} : \operatorname{diam} \sigma \leq 2r\}$$

where

$$\operatorname{diam} \sigma = \max_{p,q \in \sigma} \|p - q\|$$

Given a simplex $\sigma \in \check{\text{C}}\text{ech}_{\mathcal{X}}(r)$, we must have $B_p(r) \cap B_q(r) \neq \emptyset$ for every $p, q \in \sigma$ and in particular, this means $\|p - q\| \leq 2r$ for every $p, q \in \sigma$. Therefore, we have $\check{\text{C}}\text{ech}_{\mathcal{X}}(r) \subseteq \text{VR}_{\mathcal{X}}(r)$. Furthermore, we can establish a containment in the other direction if we increase the radius of the Čech complex enough, as proved in [26].

Lemma 2.2. *Let $\mathcal{X}$ be a point cloud and let $r \geq 0$. Then*

$$\text{VR}_{\mathcal{X}}(r) \subseteq \check{\text{C}}\text{ech}_{\mathcal{X}}(\sqrt{2}r)$$

We can use the Čech complex and the Vietoris-Rips complex to construct a simplicial complex from a point cloud $\mathcal{X}$. However, for densly populated regions, we will get very high dimensional simplices. One way to avoid such high dimensions is to consider the ***d*-skeleton**, K_d, of a simplicial complex K, which is simply the restriction of K to only include simplices of dimension d or less. Alternatively, we can avoid high dimensional simplices altogether.

For a point cloud $\mathcal{X}$, the **Voronoi diagram** of $\mathcal{X}$ is defined to be the collection $\mathcal{V}^1(\mathcal{X}) = \{V_x : x \in \mathcal{X}\}$ of sets V_x where a point $y \in \mathbb{R}^N$ is contained in V_x if and only if

$$\|x - y\| \leq \|z - y\| \qquad \forall z \in \mathcal{X}, z \neq x$$

That is, a Voronoi region V_x corresponding to the *site* x is the set of all points in $\mathbb{R}^N$ closer to x than any other point of $\mathcal{X}$. From this definition, it is clear that the boundary of these regions is the medial axis as introduced in Section 1.2. There are several algorithms to compute the Voronoi diagram including Fortune's algorithm [27], Lloyd's algorithm [36], and the Bower-Watson algorithm [55]. By uniformly sampling a region of Euclidean space and constructing the Voronoi diagram, one can construct a fairly uniform decomposition of the space.

From the definition of the Voronoi diagram, every point of $\mathbb{R}^N$ is either in a Voronoi region V_x for some $x \in \mathcal{X}$, or it is in the boundary of several Voronoi regions. In the latter case, it is clear that the point actually lies on the medial axis of the point cloud. Therefore, the sets in $\mathcal{V}^1(\mathcal{X})$ form a covering of $\mathbb{R}^N$. The nerve of this covering is known as the **Delaunay triangulation**, $\mathrm{Del}(\mathcal{X})$, of $\mathcal{X}$ [20]. This triangulation can be seen as the dual of the Voronoi diagram. Note that because we assume our point cloud $\mathcal{X}$ is in general position, the maximum dimension of any simplex in the Delaunay triangulation is N, the dimension of $\mathcal{X}$. The geometric realization of the Delaunay triangulation is used for a variety of of applications including automatic mesh construction, terrain modelling, and analyzing sampling density. Later, we will discuss a generalization of the Voronoi diagram, the k-th order Voronoi diagram, and with it, a generalized Delaunay triangulation.

Although the Delaunay triangulation is useful for many point clouds, the unbounded nature of the Voronoi regions can cause problems for some data sets. For example, suppose we sampled two lines in $\mathbb{R}^2$ given by $\ell_1 = \{(x, -1) : x \in \mathbb{R}\}$ and $\ell_2 = \{(x, 1) : x \in \mathbb{R}\}$. In the Delaunay triangulation, many edges will exist between points sampled from ℓ_1 and ℓ_2. This is because the medial axis of $\ell_1 \cup \ell_2$ is given by the line $\ell = \{(x, 0) : x \in \mathbb{R}\}$, and so Voronoi regions generated from points in ℓ_1 will terminate adjacent to regions generated from points in ℓ_2. This would yield a fully connected simplicial complex, which clearly does not reflect the topology of the set $\ell_1 \cup \ell_2$ from which the points were sampled.

The above example suggests we should bound the Voronoi regions so that we avoid such issues. Therefore, define $R_x(r) = B_x(r) \cap V_x$. The **α-complex** at resolution r, as defined in [26], is given by

$$\text{Alpha}_{\mathcal{X}}(r) = \{\sigma \subseteq \mathcal{X} | \bigcap_{u \in \sigma} R_u(r) \neq \emptyset\}$$

In the previous example, which emphasized the problem with using the Delauny triangulation, by setting $\alpha = 1/2$, the α-complex of a point cloud sampled from $\ell_1 \cup \ell_2$ will properly reflect the topology of the generating lines since no points from ℓ_1 will be close enough to points from ℓ_2 to cause an edge to be formed. Note that the nerve of $\text{Alpha}_{\mathcal{X}}(r)$ is isomorphic to a subset of the Delaunay triangulation since $R_u(r) \subseteq V_u$ for every $u \in \sigma$ and every $\sigma \subseteq \mathcal{X}$. Additionally, we have the desired property that for $r_1 \leq r_2$, we have $\text{Alpha}_{\mathcal{X}}(r_1) \subseteq \text{Alpha}_{\mathcal{X}}(r_2)$.

For all of the above complexes, we have a single parameter, r, which controls the scale of the simplicial complex. As we send r to zero, eventually the complexes simply consist of 0-simplices formed by the original points in the point cloud $\mathcal{X}$. Increasing the value of r, we were able to form chains of simplicial complexes. In particular, for $r_1 \leq r_2$, we saw $\check{\text{C}}\text{ech}_{\mathcal{X}}(r_1) \subseteq \check{\text{C}}\text{ech}_{\mathcal{X}}(r_2)$, $\text{VR}_{\mathcal{X}}(r_1) \subseteq \text{VR}_{\mathcal{X}}(r_2)$, and $\text{Alpha}_{\mathcal{X}}(r_1) \subseteq \text{Alpha}_{\mathcal{X}}(r_2)$. When we specify a sequence $0 = r_0 \leq r_1 \leq r_2 \leq ... \leq r_m$, then the induced inclusions on the simplicial complexes form what is known as a **filtration** of the simplicial complex. That is, if we let K_i denote one of the simplicial complexes discussed in this section with $r = r_i$, then we say

that the sequence

$$\emptyset = K_0 \subseteq K_1 \subseteq K_2 \subseteq ... \subseteq K_m$$

forms a filtration of K_m. Since there are only finitely many simplices in all the discussed simplicial complexes, there are only a finite number of parameter values at which the simplicial complex changes. After discussing homology in the next section, we will use filtrations in Section 2.3 when we discuss persistent homology, which tracks how the topology of a simplicial complex generated from a point cloud changes as we increase the scale parameter r.

2.2 Homology

Let $\mathbb{X}$ be a topological space. When investigating the topology of $\mathbb{X}$, two tools are often employed, homotopy and homology theory. While homotopy offers more information about the topology of $\mathbb{X}$, homology yields faster algorithms. In fact, some problems in homotopy theory, such as determining whether two triangulated 4-manifolds are homotopic, are undecidable. Despite offering less information than homotopy theory, the fact that homology provides computationally tractable algorithms makes homological considerations far more appealing. In this section, we lay out the basics of homological algebra for the case of a simplicial complex. A more thorough introduction can be found in the second part of Munkres' topology book [43] and an even more thorough treatment can be found in Rotman's text on homological algebra [48]

A **triangulation** of $\mathbb{X}$ is a simplicial complex, K, together with a homeomorphism, h, from the simplicial complex K to the topological space $\mathbb{X}$. Due to the homeomorphism, studying the topology of K is equivalent to studying the topology of $\mathbb{X}$. In this section, we will explicitly construct homology on the simplicial complex K, and all of our notation will reflect this dependence. However, because the triangulation comes equipped with a homeomorphism, it is natural to consider the homology groups of a topological space $\mathbb{X}$ without regard to the specific simplicial approximation K. Therefore, in other sections, we

will often use the notation $H_p(\mathbb{X})$ to refer to the homology groups of $\mathbb{X}$ despite the fact that the homology groups are computed on the simplicial complex K instead of the space $\mathbb{X}$.

Given two p-simplices σ_1 and σ_2 in K, we can formally define the sum of σ_1 and σ_2 (over a field of coefficients F), which we denote $a\sigma_1 + b\sigma_2$ where $a, b \in F$. We can extend these sums linearly when considering more than two p-simplices. Such formal sums are referred to as ***p*-chains**. For the current application, we will only consider sums with coefficients 0 or 1, that is we choose our field of coefficients F to be $\mathbb{Z}_2 = \mathbb{Z}/2\mathbb{Z}$.

Let K_p denote the set of all p-simplices of K. Then given two p-chains

$$c_1 = \sum_{i=1}^{|K_p|} a_i\sigma_i, \qquad c_2 = \sum_{i=1}^{|K_p|} b_i\sigma_i$$

with $a_i, b_i \in \mathbb{Z}_2$, we define addition component-wise,

$$c_1 + c_2 = \sum_{i=1}^{|K_p|} (a_i + b_i)\sigma_i$$

The set of all p-chains with addition as defined above forms an abelian group, known as the ***p*-th chain group** of K, denoted $C_p(K)$. When discussing homology, one must specify the chain groups, and homomorphisms between the chain groups. For our situation, we must define maps which take us from p-chains to $(p-1)$-chains. Since the boundary of a p-simplex is in fact a $(p-1)$-chain, it seems natural to use this property when forming the **boundary homomorphism**, $\partial_p : C_p \to C_{p-1}$. The homomorphism ∂_p maps a p-chain c to the $(p-1)$-chain $\partial_p(c)$ defined as the formal sum of all codimension-one faces of the p-simplices of σ. That is, for a p-simplex $\sigma = (v_0, ..., v_p)$, which is itself a length one p-chain

and hence a member of C_p, we define

$$\partial_p(\sigma) = \partial_p((v_0, ..., v_p)) = \sum_{i=0}^{p} (v_0, ..., \hat{v}_i, ..., v_p)$$

where $\hat{v}_i$ denotes the absence of a vertex. Since we are working with $\mathbb{Z}_2$ coefficients, we do not need to worry about the orientation of a simplex, and hence are left with a simple formula for the boundary. We then define the boundary map acting on an arbitrary p-chain $c = \sigma_1 + ... + \sigma_k$ with $\sigma_i \in K_p$ for $1 \leq i \leq k$ as

$$\partial_p(\sigma_1 + ... + \sigma_k) = \partial_p(\sigma_1) + ... + \partial_p(\sigma_k)$$

Putting the chain groups and the boundary homomoprhisms into a sequence, we obtain what is known as a **chain complex**,

$$... \xrightarrow{\partial_{p+2}} C_{p+1}(K) \xrightarrow{\partial_{p+1}} C_p(K) \xrightarrow{\partial_p} C_{p-1}(K) \xrightarrow{\partial_{p-1}} ...$$

Note that at some point the complex will terminate since eventually we will reach the chain group C_0, consisting of 0-simplices. Additionally, since we are dealing with complexes of finite dimension, the sequence will terminate in the other direction as well, in the sense that the chain groups will become trivial when p exceeds the dimension of the complex.

Next, define $Z_p(K) = \ker \partial_p$ and $B_p(K) = \operatorname{im} \partial_{p+1}$. We refer to Z_p and B_p as the ***p*-th cycle group** and ***p*-th boundary group** of the simplicial complex, respectively. In order to form the so called homology groups of K, we must have for any $p > 0$ and any $c \in C_p(K)$,

$$\partial_p \circ \partial_{p+1}(c) = 0$$

To see that this is in fact the case, let $(v_0, ..., v_{p+1})$ be some simplex in K_{p+1} and note that

$$\partial_p \circ \partial_{p+1}(v_0, ..., v_{p+1}) = \partial_p \left(\sum_{i=0}^{p} (v_0, ..., \hat{v_i}, ..., v_{p+1}) \right) = \sum_{i=0}^{p+1} \sum_{j=0, i \neq j}^{p+1} (v_0, ..., \hat{v_i}, ..., \hat{v_j}, ..., v_p)$$

Each $(p-1)$-simplex in the final sum appears twice and no $(p-1)$-simplex appears only once. Since we are working with $F = \mathbb{Z}_2$ as our field of coefficients, the addition is carried out modulo 2 and therefore, $\partial_p \circ \partial_{p+1}(c) = 0$. Thus, if $c \in B_p(K)$, then by definition there exists $c' \in C_{p+1}$ such that $\partial_{p+1}(c') = c$. However, this implies $\partial_p \circ \partial_{p+1}(c') = \partial_p(c) = 0$ and so $c \in Z_p(K)$. Thus, we see that $B_p(K)$ is a subset of $Z_p(K)$ for every p. It is in fact a normal subgroup and so we can form the quotient group

$$H_p(K) = Z_p(K)/B_p(K)$$

known as the ***p*-th homology group** of K. The assumption that $F = \mathbb{Z}_2$ ensures $H_p(K)$ defines a vector space over $\mathbb{Z}_2$. The dimensions of these vector spaces are known as the **Betti numbers** of K. That is, the p-th Betti number, β_p, is defined as

$$\beta_p = \operatorname{rank} H_p(K)$$

The Betti numbers reflect the number of p-dimensional holes in the simplicial complex K, and by association, the number of p-dimensional holes in the topological space $\mathbb{X}$. For small dimensions, one can informally think of β_0 as the number of connected components of K, β_1 as the number of 1-dimensional holes in K, and β_2 as the number of voids in K. Interestingly, the Betti numbers are connected to the Euler-characteristic through the following formula,

$$\chi(K) = \sum_{p \geq 0} (-1)^p \beta_p$$

Connecting this with the simplicial complexes described in Section 2.1, given a simplicial complex K, of the type discussed, with a small enough scale parameter r, the zeroth Betti number β_0 will simply be the size of the generating point cloud $\mathcal{X}$ and the other Betti numbers will be zero. As we grow the scale parameter, some edges and higher dimensional simplices will be formed. Thus, as we grow r, the Betti numbers will change. We can track these changes using the notion of persistent homology, discussed in the next section.

2.3 Persistent Homology

The field of persistent homology has been around for almost 30 years, however it has gained substantial popularity over the past decade. Although homology allows us to rigorously investigate the topological structure of simplicial complexes, it only operates on a single simplicial complex. In the case of point clouds, this requires us to fix a scale parameter r and build a simplicial complex at this scale. However, since the point cloud consists of zero dimensional points in $\mathbb{R}^N$, it is not immediately clear what serves as an appropriate value for the scale parameter. Furthermore, the topological characteristics of the simplicial complexes change as we vary r. Noise in the point cloud may cause features to appear at one scale and then abruptly disappear at a slightly larger scale. The key point here is that while noisy features will come and go, more stable features of the data should persist over larger ranges of the scale parameter r. We make this notion rigorous in this section.

Consider a real valued continuous function $f : \mathbb{X} \to \mathbb{R}$ on a topological space $\mathbb{X}$. We define the sublevel set $\mathbb{X}^r = f^{-1}(-\infty, r]$ for any $r \in \mathbb{R}$ similar to our definition in previous sections. Clearly, the definition tells us that for any $r < s$ we have $\mathbb{X}^r \subseteq \mathbb{X}^s$. By specifying a sequence $0 = r_0 < r_1 < ... < r_{m-1} < r_m = \infty$, we can induce a filtration, $\mathcal{F}$, of $\mathbb{X}$ given by

$$\mathcal{F} := \mathbb{X}_0 \subseteq \mathbb{X}_1 \subseteq ... \subseteq \mathbb{X}_{m-1} \subseteq \mathbb{X}_m = \mathbb{X}$$

where for notational simplicity, $\mathbb{X}_i = \mathbb{X}^{r_i}$. The inclusions at each level of the filtration induce homomorphisms on the level of homology, $H_k(\mathbb{X}_i) \to H_k(\mathbb{X}_{i+1})$. Depending on our

choices for the sequence $\{r_i\}_{i=0}^m$, the topology of the sublevel sets may change as we move along the induced filtration $\mathcal{F}$. In particular, the homology groups may change. We capture this notion more precisely in the following definition from [29].

Definition 2.3. Given a topological space $\mathbb{X}$ and a function $f : \mathbb{X} \to \mathbb{R}$, the value $a \in \mathbb{R}$ is a **homological regular value** of the function f if there exists $\epsilon > 0$ such that for each pair $x, y \in (a - \epsilon, a + \epsilon)$ with $x < y$, the inclusion

$$f^{-1}(-\infty, x] \hookrightarrow f^{-1}(-\infty, y]$$

induces isomorphisms on all the homology groups, that is

$$H_p(f^{-1}(-\infty, x]) \cong H_p(f^{-1}(-\infty, y])$$

for all p and each acceptable pair x, y. A number $a \in \mathbb{R}$ is a **homological critical value** of f if it is not a homological regular value of f. A function f is called **tame** if it has a finite number of homological critical values and all the homology groups $H_p(\mathbb{X}^r)$, for every $r \in \mathbb{R}$, have finite rank.

Looking closer at the homomorphisms induced by the inclusions, for $r < s$ we have the map

$$h_p^{r,s} : H_p(\mathbb{X}^r) \to H_p(\mathbb{X}^s)$$

which relates the homology of the sublevel sets $\mathbb{X}^r$ and $\mathbb{X}^s$. In particular, any homology class appearing in the kernel of the homomorphism $h_p^{r,s}$ exists in the homology group of $\mathbb{X}^r$ and ceases to exist in the homology group of $\mathbb{X}^s$. On the other hand, classes in the image of $h^{r,s}$ persist as we increase the scale parameter from r to s. This notion of persistence, allows us to define the persistent homology groups, which track the topological features which presist as we vary the scale parameter r.

Definition 2.4. Let $h_p^{r,s}$ denote the homomorphism between the p-th homology groups induced by the inclusion map, as defined above. The ***p*-dimensional persistent homology group** of f is defined as

$$H_p^{r,s} = \operatorname{im} h_p^{r,s}$$

The corresponding ***p*-th persistent Betti numbers** are the ranks of the persistence groups,

$$\beta_p^{r,s} = \operatorname{rank} H_p^{r,s}$$

When one considers a filtration of the topological space $\mathbb{X}$ as described above, we often write the persistent homology groups and the persistent Betti numbers using indices instead of the real values r, s. That is, we will denote the p-dimensional persistent homology group between $\mathbb{X}_i$ and $\mathbb{X}_j$ by $H_p^{i,j}$ and the p-th persistent Betti numbers will be denoted by $\beta_p^{i,j}$.

Given a tame function f, we can enumerate the homological critical values as $a_1 < a_2 < ... < a_m$ along with $a_0 = -\infty$ and $a_{m+1} = \infty$. We then pick intermediate values, $b_i \in (a_i, a_{i+1})$ and induce a filtration from the sequence $b_0 < b_1 < ... < b_m$. For the rest of this section, we will let $\mathcal{F} := \mathbb{X}_0 \subseteq \mathbb{X}_1 \subseteq ... \subseteq \mathbb{X}_m = \mathbb{X}$ be the induced filtration. By the definition of homological critical values, we know that the homology of the sublevel sets $\mathbb{X}^r$ does not change over the interval (a_i, a_{i+1}) for any $0 \leq i \leq m$. Therefore, this filtration captures the changes in homology for all the sublevel sets.

We can be more concrete about the persistence of individual homology classes. Let $\sigma \in H_p(\mathbb{X}_i)$ be a homology class present in the filtration at step i. We say that σ is **born** at $\mathbb{X}_i$ if $\sigma \notin H_p^{i-1,i}$. That is, a homology class is born at step i of a filtration if it exists in $H_p(\mathbb{X}_i)$, but is not the image of some class in $H_p(\mathbb{X}_{i-1})$. Similarly, for a class σ born at $\mathbb{X}_i$, we say that σ **dies** at $\mathbb{X}_j$ if $h_p^{i,j-1}(\sigma) \notin H_p^{i-1,j-1}$ but $h_p^{i,j}(\sigma) \in H_p^{i-1,j}$. That is, σ is born at step i and dies at step j if the class σ merges with an older homology class going from $j-1$ to j. The difference, $r_j - r_i$, between the birth (r_i) and death (r_j) of a homology class σ is called the **persistence** of σ. When a homology class σ is born but never dies, we say that

the class persists forever and has infinite persistence.

Since we can associate a birth and (possibly infinite) death to every homology class of the space $\mathbb{X}$ under a given filtration, we can form what is known as the **persistence diagram**, denoted $\text{Dgm}_p(f)$, of $\mathbb{X}$ with respect to the function f. The persistence diagram of a topological space $\mathbb{X}$ with an associated filtration $\mathcal{F}$, is a multiset of points in $\mathbb{R} \times (\mathbb{R} \cup \{\infty\})$. In particular, for every homology class σ which is born at some step of the filtration, we associate σ with the point (b, d) in $\mathbb{R} \times (\mathbb{R} \cup \{\infty\})$ where σ is born at $b \in \mathbb{R}$ and dies at $d \in \mathbb{R} \cup \{\infty\}$. Under this construction, the vertical distance from (b, d) to the diagonal gives the **persistence** of σ. We take the collection of all such points (one for each homological class in $H_p(\mathbb{X})$, allowing for multiplicity), and denote these points by the set $P = \{(b_1, d_1), ..., (b_m, d_m)\}$ where m is the number of homology classes in $H_p(\mathbb{X})$. Now let $b_* = \max_i b_i$ and $d_* = \max_i d_i$. Let the diagonal of $\mathbb{R}^2$ be denoted by $\Delta = \{(x, x) : x \in \mathbb{R}\}$ and let $\tilde{\Delta}$ represent the diagonal Δ intersected with some bounded domain $[0, r_1] \times [0, r_2]$ where $r_1 \gg b_*$ and $r_2 \gg d_*$. We use the bounded diagonal instead of the full diagonal because of compactness issues (in practice this assumption is inconsequential). Then the collection $P \cup \tilde{\Delta}$ is called the persistence diagram, $\text{Dgm}_p(\mathbb{X})$, of $\mathbb{X}$. We will sometimes use the notation $\text{Dgm}_p(\mathcal{F})$ to denote the persistence diagram of $\mathbb{X}$ when we wish to make the dependence on the filtration specific. The reason for including all points along the bounded diagonal is technical and will be made clear in the next section. For now, we justify their inclusion by noting that these points have zero persistence and are therefore topologically trivial.

We can be more precise about the multiplicity of a point in the persistence diagram. If we let $\mu_p^{i,j}$ denote the number of independent p-dimensional homology classes which are born at $\mathbb{X}_i$ and die at $\mathbb{X}_j$, then we have the following useful formula for the multiplicity, which is shown in [26],

$$\mu_p^{i,j} = (\beta_p^{i,j-1} - \beta_p^{i,j}) - (\beta_p^{i-1,j-1} - \beta_p^{i-1,j})$$

Using this formula, one can prove the following lemma, stating that the persistence diagram encodes all the information about the homology groups. It is worth noting that Edelsbrunner and Harer refer to this result as the **Fundamental Lemma of Persistent Homology** in [26].

Lemma 2.5. *Let $\emptyset = K_0 \subseteq K_1 \subseteq ... \subseteq K_m = K$ be a filtration of K. For every pair of indices $0 \leq k \leq l \leq m$, and every dimension p, we have the following equivalence,*

$$\beta_p^{k,l} = \sum_{i \leq k} \sum_{j > l} \mu_p^{i,j}$$

2.4 Stability of Persistence Diagrams

In practice, point clouds are often obtained through noisy processes. One of the key advantages of considering the topology of the point cloud over studying its geometry is that many topological properties are robust to small perturbations of the data. Considering the fact that persistence diagrams capture topological properties of the point cloud, it should come as no surprise that persistence diagrams exhibit stability as well. In this section, we discuss the stability of persistence diagrams. Using the stability result, we can show that persistence diagrams evolve continuously under continuous gradient flows. How these diagrams evolve is captured in what is known as the vineyard, introduced in [18], of the flowing point cloud.

To begin, we will define two notions of distance which will be useful throughout the remainder of this dissertation. Recall that for two sets X, Y of points in $\mathbb{R}^N$, the **Hausdorff distance** between X and Y, denoted $d_H(X, Y)$ is defined as follows,

$$d_H(X,Y) = \max \left\{ \sup_{x \in X} \inf_{y \in Y} \|x - y\|_\infty, \sup_{y \in Y} \inf_{x \in X} \|x - y\|_\infty \right\}$$

Of course, the Hausdorff distance between two sets can be infinite for unbounded sets.

Additionally, when X and Y are not closed, we may run into problems where $d_H(X,Y) = 0$ even though $X \neq Y$. For example, if $X = (0,1)$ and $Y = [0,1]$, then $d_H(X,Y) = 0$, yet clearly $X \neq Y$. However, on the space of non-empty compact subsets of $\mathbb{R}^N$, the Hausdorff distance is in fact a metric and thus we can use the Hausdorff distance to define a metric space on this set of sets. Next, we recall the related notion of the **bottleneck distance** $d_B(X,Y)$ between X and Y.

$$d_B(X,Y) = \inf_{\gamma} \sup_{x \in X} \|x - \gamma(x)\|_\infty$$

where the infimum is over all bijections $\gamma : X \to Y$ and $\|\cdot\|_\infty$ is the ℓ_∞ norm on $\mathbb{R}^N$. From the definitions above, it is clear that $d_H(X,Y) \leq d_B(X,Y)$.

Of course, persistence diagrams are multisets consisting of pairs $(b,d) \in \mathbb{R}^2$ along with the bounded diagonal $\tilde{\Delta}$. Since finite point sets and the bounded diagonal $\tilde{\Delta}$ are compact, their union will be compact. It is obviously also non-empty since $\tilde{\Delta} \neq \emptyset$. The choice of including the bounded diagonal $\tilde{\Delta}$ in the definition of the persistence diagram should now be clear. We bounded Δ to ensure compactness and we included $\tilde{\Delta}$ so that we can use the bottleneck distance on persistence diagrams $X = \mathrm{Dgm}_p(f_1)$ and $Y = \mathrm{Dgm}_p(f_2)$ where X does not have the same number of homology classes as Y (recall we must form bijections between X and Y). Given these considerations, we can use the bottleneck and Haussdorff distances as metrics on the space of persistence diagrams.

As it turns out, when two functions on a triangulable space are close (in the L^∞ sense), the resulting persistence diagrams will be close as well. This is critical as it demonstrates that persistence diagrams are robust to small perturbations in the underlying data. To see this, we turn to the key stability theorem from [17].

Theorem 2.6. *(Stability Theorem) Let $\mathbb{X}$ be a triangulable space and let $f, g : \mathbb{X} \to \mathbb{R}$ be two continuous and tame functions. For any dimension $p \geq 0$, the bottleneck distance between the persistence diagrams $\mathrm{Dgm}_p(f)$, $\mathrm{Dgm}_p(g)$ is bounded above by the L^∞ distance*

between f and g, that is

$$d_B(\mathrm{Dgm}_p(f), \mathrm{Dgm}_p(g)) \leq \|f - g\|_{L^\infty(\mathbb{R}^N)}$$

Since $d_H(X, Y) \leq d_B(X, Y)$ for every pair of multisets X, Y, the result above holds for Hausdorff distances as well. Of particular interest to the present work, if we evolve a point cloud using a gradient flow, the persistence diagrams will evolve continuously. That is, given a point cloud $\mathcal{X} = \{x_1, ..., x_n\}$ and a smooth function $E : \mathbb{R}^N \to \mathbb{R}$, we can define a gradient flow using the following system

$$\begin{cases} \dfrac{du_i}{dt} = -\nabla E(u_i) \\ \\ u_i(0) = x_i \end{cases}$$

for $1 \leq i \leq n$. This gives us a collection of point clouds $\mathcal{F} = \{\mathcal{X}_t : t \geq 0\}$ where $\mathcal{X}_t = \{u_i(t) : 1 \leq i \leq n\}$. For each time $t \geq 0$, we can compute the persistence diagram $\mathrm{Dgm}_p(f_t)$ where $f_t : \mathbb{R}^N \to \mathbb{R}$ is defined by $f_t(y) = d(y, \mathcal{X}_t)$. This yields a collection of persistence diagrams $\mathrm{Dgm}_p(\mathcal{F}) = \{\mathrm{Dgm}_p(f_t) : t \geq 0\}$. The stability theorem tells us that since $\mathcal{X}_t$ evolves continuously, the function $t \mapsto \mathrm{Dgm}_p(f_t)$ is continuous as well. In particular,

$$d_B(\mathrm{Dgm}_p(f_t), \mathrm{Dgm}_p(f_s)) \leq \|f_t - f_s\|_\infty$$

Since each persistence diagram is a multiset in $\mathbb{R}^2$, and these multisets are close in the sense above, we can treat the persistence diagrams from $\mathcal{F}$ as being embedded in $\mathbb{R}^3$ where the third dimension corresponds to $t \in \mathbb{R}$. Since the multisets evolve continuously, when stacked, the points form so-called **vines**. Taken together, the stacked persistence diagrams form what is known as a **vineyard**.

From the stability result above, if we can get bounds on $\|f_t - f_s\|_{L^\infty(\mathbb{R}^N)}$, we can provide an upper bound on the distance between the persistence diagrams at time t and at time s. Since E was assumed to be smooth, will will have $\|f_t - f_s\|_{L^\infty(\mathbb{R}^N)} \to 0$ as $t \to s$ and therefore we will have $d_B(\mathrm{Dgm}_p(f_t), \mathrm{Dgm}_p(f_s)) \to 0$ as $t \to s$. This confirms that under continuous gradient flows, the resulting persistence diagrams will evolve continuously.

In this dissertation, we will be interested in the difference between two point clouds in a topological sense. That is, given two point clouds $\mathcal{X}$ and $\mathcal{Y}$, we wish to measure the difference between $\mathcal{X}$ and $\mathcal{Y}$ using persistence diagrams. To this end, we will let $\mathbb{X}_r$ and $\mathbb{Y}_r$ be the α-complexs induced by $\mathcal{X}$ and $\mathcal{Y}$, respectively, at resolution r. That is,

$$\mathbb{X}_r = \mathrm{Alpha}_{\mathcal{X}}(r) \qquad \mathbb{Y}_r = \mathrm{Alpha}_{\mathcal{Y}}(r)$$

Then choosing values $0 = r_1 < r_2 < ... < r_{m-1} < r_m = \infty$ and $0 = q_1 < q_2 < ... < q_{k-1} < q_k = \infty$ corresponding to midpoints of the homological critical values of $\mathcal{X}$ and $\mathcal{Y}$ respectively (as described in Section 2.3), we can induce the filtrations

$$\mathcal{F}_{\mathcal{X}} := \mathbb{X}_{r_0} \subseteq \mathbb{X}_{r_1} \subseteq ... \subseteq \mathbb{X}_{r_m}$$

and

$$\mathcal{F}_{\mathcal{Y}} := \mathbb{Y}_{q_0} \subseteq \mathbb{Y}_{q_1} \subseteq ... \subseteq \mathbb{Y}_{q_k}$$

Next, we take the corresponding p-dimensional persistence diagrams, $\mathrm{Dgm}_p(\mathcal{F}_{\mathcal{X}})$ and $\mathrm{Dgm}_p(\mathcal{F}_{\mathcal{Y}})$, and use the bottleneck distance between these persistence diagrams to describe the ***p*-dimensional topological error**, $E_T^p(\mathcal{X}, \mathcal{Y})$, between $\mathcal{X}$ and $\mathcal{Y}$. That is, we set

$$E_T^p(\mathcal{X}, \mathcal{Y}) = d_B(\mathrm{Dgm}_p(\mathcal{F}_{\mathcal{X}}), \mathrm{Dgm}_p(\mathcal{F}_{\mathcal{Y}}))$$

where d_B is the bottleneck distance. This error function will come up in Chapter 5 when we explore numerical results related to the gradient flows discussed in this dissertation.

Chapter 3: k-Nearest Neighbor Smoothing

In this chapter, we present several methods of smoothing high dimensional point clouds using gradient flows designed to reduce the noise present in point clouds sampled with noise. The technique we develop is modeled after work done by Chazal et al in [14], where the authors introduce a gradient flow based on the distance-to-measure function. However, unlike the adhoc approach taken by Chazal, we present this gradient flow in a systematic way and provide a theoretical framework for its study. Using this framework, we are able to fully characterize the dynamics of the system. Furthermore, we introduce novel modifications to the flow which improve the smoothing behavior of the system. Through the use of the gradient flow based smoothing algorithm and the modifications we present, we aim to get better reconstruction results.

Smoothing point clouds is a common preprocessing step when analyzing point clouds. Often, if left unchecked, the noise present in a point cloud will corrupt the ensuing analysis. The benefits of smoothing using the gradient flows we discuss in this chapter are numerous. For starters, they are relatively easy to compute. Additionally, since we are inducing a flow, the level of smoothing can be naturally defined using the run time of the gradient flow. That is, the longer we let the flow evolve, the more smoothing we impose on the point cloud. The primary motivation behind the techniques presented in this section is the desire to reduce the overall distance-to-measure exhibited by the point cloud. In particular, we will evolve the points to reduce their distance-to-measure.

As we will show, when we take the measure μ to be the empirical measure of a point cloud, the distance-to-measure function $d_{m,\mu}$ is simply a quadratic k-nearest neighbors energy function. Since we can define this function everywhere in a continuous fashion, it is natural to induce a gradient flow on the point cloud and move the points according to this flow, thereby reducing their distance to the measure μ. This gradient flow will be referred to

as the k-nearest neighbors gradient flow for reason which will become obvious. A straightforward advantage of this approach over other smoothing algorithms is that computing the gradient and flowing along the gradient are inexpensive operations. The most expensive part of this process is computing the nearest neighbors, however this is an unavoidable problem for most smoothing algorithms. Since this operation is the most costly, in the next chapter we will discuss ways of improving the complexity of the algorithm by using space partitioning algorithms when determing the nearest neighbor sets. The result is a fast algorithm for reducing the noise of a point cloud.

This method is not without its flaws. As we will see, the induced gradient flow causes points to cluster together when the points are allowed to evolve for a long time. In this chapter, we will discuss a simple method to help alleviate this issue. Later, in the next chapter, we will build a more sophisticaed approach which approximates the normal space of a high dimensional manifold in a method similar to the moving least squares method discussed in Section 1.4. We will then project the computed gradient onto this normal space. This technique provides improved smoothing performance over the method presented in this chapter. However, before we can build the more advanced technique, we must understand the k-nearest neighbors gradient flow.

To provide a systematic theoretical framework for the k-nearest neighbors gradient flow, we must first introduce the higher order generalization of the Voronoi diagram. These higher order Voronoi diagrams provide a natural setting for investigating k-nearest neighbor problems. From this construction, we will be able to characterize many qualitative aspects of the gradient flow. In particular, we will identify conditions for all the sinks of the flow, we will find some positively invariant sets, and we will show there are no periodic orbits induced by the flow. Additionally, we will construct a *flow diagram* which fully describes the qualitative dynamics of the k-nearest neighbors gradient flow. Unfortunately, this construction will be computationally expensive and we will therefore be required to restrict the examples we present to two dimensions. However, the results will be valid and stated for arbitrary dimensions.

3.1 k-order Voronoi Regions

Voronoi diagrams have found applications in a wide variety of areas such as astronomy, robot navigation, and machine learning (see [45] for specific examples). While the traditional Voronoi diagram considers the nearest neighbor of a point, our problem will require us to consider the k-nearest neighbors of a point, for $k \in \mathbb{N}$ with $k > 1$. Applications of higher order Voronoi diagrams are nearly as numerous as the single nearest neighbor version. For example, k-nearest neighbors classification in machine learning becomes trivial once the higher order Voronoi diagram is constructed. Additionally, with the construction of a third order Voronoi diagram, determining the three closest radio towers from a given point becomes trivial as well.

Given a point cloud $\mathcal{X}$ and a point $x \in \mathbb{R}^N$, let $\mathrm{NN}^k_{\mathcal{X}}(x)$ denote the set of k-nearest neighbors of x, where $k < n = |\mathcal{X}|$. That is, we order the points $x_1, ..., x_n \in \mathcal{X}$ such that $\|x - x_1\| \leq ... \leq \|x - x_n\|$ and set $r(x) = \|x - x_k\|$. Then we define

$$\mathrm{NN}^k_{\mathcal{X}}(x) = \{x_i \in \mathcal{X} : \|x - x_i\| \leq r(x)\}$$

Note that the k-nearest neighbor set, as defined above, can contain more than k points of $\mathcal{X}$. These points will be of interest to us throughout the dissertation. From this definition of $\mathrm{NN}^k_{\mathcal{X}}(x)$, generalizing the definition of the Voronoi diagram to the k^{th}-order Voronoi diagram is straightforward, as seen in the following definition.

Definition 3.1. Let $\mathcal{X} \subset \mathbb{R}^N$ be a point cloud. For $u \in \mathbb{R}^N$, let $\mathrm{NN}^k_{\mathcal{X}}(u)$ denote the set of k-nearest neighbors of u, as defined above. If we have $|\mathrm{NN}^k_{\mathcal{X}}(u)| = k$, then we define

$$V^k(u) = \mathrm{cls}\left(\{v \in \mathbb{R}^N : \mathrm{NN}^k_{\mathcal{X}}(v) = \mathrm{NN}^k_{\mathcal{X}}(u)\}\right)$$

where $\mathrm{cls}(A)$ is the closure of a set A. For $u \in \mathbb{R}^N$, we call $V^k(u)$ the **k^{th}-order Voronoi region** centered at u. The points $u \in \mathbb{R}^N$ for which $|\mathrm{NN}^k_{\mathcal{X}}(u)| \neq k$ lie on the boundary of

a k^{th}-order Voronoi region. We call the set

$$V^k(\mathcal{X}) = \{V^k(x) : x \in \mathbb{R}^N \text{ such that } |\operatorname{NN}^k_{\mathcal{X}}(x)| = k\}$$

the $\boldsymbol{k^{\text{th}}}$**-order Voronoi diagram** of $\mathcal{X}$. Given a k^{th}-order Voronoi region V and a point $x \in \operatorname{int}(V)$, we call $\operatorname{NN}^k_{\mathcal{X}}(x)$ the **generators** of V, denoted $\operatorname{gen}(V)$. Finally, since for $y \in V^k(x)$ with $|\operatorname{NN}^k_{\mathcal{X}}(y)| = k$ we have $V^k(x) = V^k(y)$, we must take this set to be defined without multiplicity.

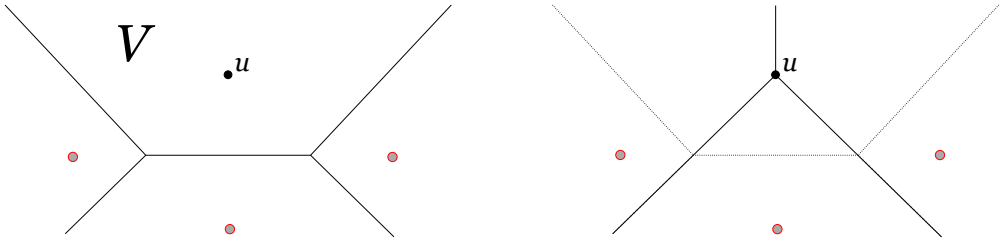
By the definition of a k-order Voronoi region $V \in V^k(\mathcal{X})$, we see that the definition of $\operatorname{gen}(V)$ does not depend on the particular choice of $x \in \operatorname{int}(V)$. This is because for all $x, y \in \operatorname{int}(V)$, we have $\operatorname{NN}^k_{\mathcal{X}}(x) = \operatorname{NN}^k_{\mathcal{X}}(y)$ by definition. Thus, $\operatorname{gen}(V)$ is well defined. For points $u \in \mathbb{R}^N$ such that $|\operatorname{NN}^k_{\mathcal{X}}(u)| \neq k$, we know that for any k^{th}-order Voronoi region V with $\operatorname{gen}(V) \subseteq \operatorname{NN}^k_{\mathcal{X}}(u)$, the point u will lie on the boundary of V, that is, $u \in \partial V$. We define the $\boldsymbol{k^{\text{th}}}$**-order Voronoi skeleton** of $\mathcal{X}$ to be

$$\partial V^k(\mathcal{X}) = \bigcup_{V \in V^k(\mathcal{X})} \partial V \tag{3.1}$$

Clearly for $k = 1$, the definition of $V^k(\mathcal{X})$ reduces to that of the traditional Voronoi diagram. From the definition, there are obviously at most

$$\binom{|\mathcal{X}|}{k}$$

k^{th}-order Voronoi regions, however in practice most of these will be empty sets. Also, if the point cloud $\mathcal{X}$ is finite, the k^{th}-order Voronoi diagram must contain a finite number of regions in $\mathbb{R}^N$ (since the k^{th}-order Voronoi regions are convex, as we will soon see). Thus, since we only consider finite point clouds in this dissertation, the intersection in Equation



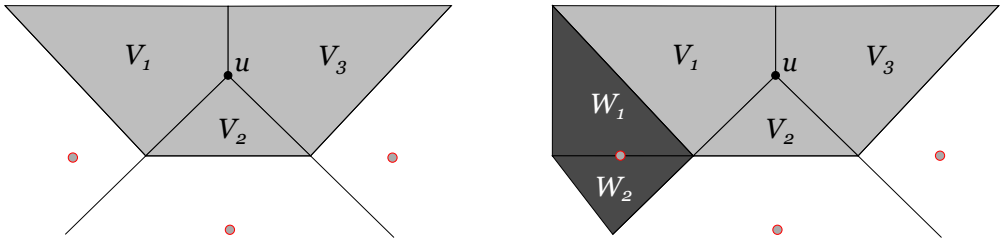
(a) $V^1(\mathcal{X})$ (b) Decomposition of V

Figure 3.1: Going from $k = 1$ to $k = 2$

3.1 will be of a finite number of closed sets. Hence, $\partial V^k(\mathcal{X})$ will be closed as well.

Note that the k^{th}-order Voronoi regions can be constructed recursively. Specifically, given a set $\mathcal{X} = \{x_1, ..., x_n\}$, suppose we can calculate the $(k-1)^{\text{th}}$-order Voronoi diagram, that is $V^{(k-1)}(\mathcal{X}) = \{V_1, ..., V_m\}$ where m is the number of Voronoi regions in $V^{(k-1)}(\mathcal{X})$. For a given $1 \leq i \leq m$, recall that any point $v \in \text{int}(V_i)$ has the same $k-1$ nearest neighbors set, $\text{NN}_{k-1}(\mathcal{X}) = \text{gen}(V_i)$. To begin constructing $V^k(\mathcal{X})$, we compute the 1^{st}-order Voronoi diagram on the set $\mathcal{X} \setminus \text{gen}(V_i)$. Suppose the resulting 1^{st}-order regions are denoted $W_1^i, ..., W_{m_i}^i$ for some $m_i \in \mathbb{N}$. By intersecting each W_j^i with V_i, we can decompose V_i into subregions, where for each region, every point in the region has the same k-nearest neighbors. To see this, let $x, y \in \text{int}(W_j^i \cap V_i)$. Then $x, y \in \text{int}(V_i)$ and so they must have the same $k-1$ nearest neighbors, specifically $\text{gen}(V_i)$. However, we also have $x, y \in \text{int}(W_j^i)$ and so the nearest neighbor of x and y in the set $\mathcal{X} \setminus \text{gen}(V_i)$ must be the single element of $\text{gen}(W_j^i)$. Therefore, both x and y have the same nearest k neighbors.

This process is illustrated in Figure 3.1, where the left hand figure shows the first order Voronoi diagram for a set of 4 points. We wish to decompose the region V, generated by the black point u, into subregions corresponding to the $k = 2$ nearest neighbors of u. The edges of the first order Voronoi diagram are shown as black solid lines. To decompose V, we first compute the first order Voronoi diagram on the point cloud $\mathcal{X} \setminus \{u\}$, as shown by the black solid lines in Figure 3.1b. In this figure, we have made the edges of the $k = 1$



(a) $V^1(\mathcal{X})$ (b) Decomposition of V

Figure 3.2: Going from $k = 1$ to $k = 2$

order Voronoi diagram dashed so it is clear that the two diagrams are overlayed.

We then intersect this diagram with the region V. The result of this operation is shown in Figure 3.2a. After the intersection, the subregions V_1, V_2, and V_3 are formed. Each of these regions will consist of points sharing the same two nearest neighbors. Of course, we have only decomposed the region V and we will have to repeat this procedure for every region in the first order Voronoi diagram.

Following the procedure outlined above for each $V_i \in V^k(\mathcal{X})$, we have a set of regions in $\mathbb{R}^N$ denoted

$$A = \{V_i \cap W_j^i : 1 \leq j \leq m_i, 1 \leq i \leq m\}$$

where $V^1(\mathcal{X} \setminus \text{gen}(V_i)) = \{W_j^i : 1 \leq j \leq m_i\}$ is the 1^{st}-order Voronoi decomposition of $\mathbb{R}^N$ using the points in $\mathcal{X} \setminus \text{gen}(V_i)$. Note that this decomposition is not exactly $V^k(\mathcal{X})$ since adjacent regions in $V^{(k-1)}(\mathcal{X})$ may be decomposed in such a way that they have adjacent subregions with the same k neighbors. For example, suppose $V_1, V_2 \in V^{(k-1)}(\mathcal{X})$ such that $\text{gen}(V_1) = \{x_1, ..., x_{k-1}\}$ and $\text{gen}(V_2) = \{x_2, ..., x_k\}$ are adjacent Voronoi regions in $V^{(k-1)}(\mathcal{X})$ with two disagreeing generators (i.e. $x_1 \in V_1$ but $x_1 \notin V_2$ and $x_k \in V_2$ but $x_k \notin V_1$). When the two regions V_1 and V_2 are intersected with the first order Voronoi diagrams of $\mathcal{X} \setminus \text{gen}(V_1)$ and $\mathcal{X} \setminus \text{gen}(V_2)$ respectively, the resulting decompositions of V_1 and V_2 may contain adjacent subsets where the k-nearest neighbors are $\{x_1, ..., x_k\}$ in both

subsets. Therefore, to finish the construction, one needs to search the adjacent regions in A and perform a union whenever the k-nearest neighbor sets of the adjacent regions are in agreement. This issue is illustrated in Figure 3.2b. Here, we see that the regions V_1 and W_1 will actually contain the same $k = 2$ nearest neighbors. Thus, we would need to take the union of these two regions. Through the above discussion, we have established the following.

Theorem 3.2. *The k^{th}-order Voronoi diagram of a point cloud $\mathcal{X}$ can be constructed recursively from the lower order Voronoi diagrams of $\mathcal{X}$.*

Note that while the above recursive algorithm will produce a higher order Voronoi diagram, it is by no means an efficient approach. Obviously, to construct a k^{th}-order Voronoi diagram following this approach, one must first construct all the lower order diagrams. Certainly, a fast first order Voronoi diagram algorithm would be desired to implement this algorithm, however a method which skips this recursive construction would be desireable. Furthermore, at each step of the recursion, we need to compute a first order Voronoi diagram for *every* Voronoi diagram already constructed. Therefore, the computational complexity of this algorithm will be exceptionally high. There are in fact much more efficient methods for constructing higher order Voronoi diagrams. Examples of such algorithms are given in [16], [6], and [35].

Now that we know of a few ways of constructing the higher order Voronoi diagrams, we put aside their construction and begin discussing their properties. We first need to recall a few definitions for convex polytopes.

Definition 3.3. A subset $C \subset \mathbb{R}^N$ is a **bounded convex polytope** if it is the convex hull of a finite set of points in $\mathbb{R}^N$.

As it turns out, Voronoi diagrams are collections of bounded and unbounded convex polytopes. When discussing convex polytopes, two representations are employed in the literature, the *vertex representation* and the *intersection of half spaces representation.* The vertex representation only applies for bounded convex polytopes. Since Voronoi diagrams

contain unbounded polytopes, we will need the half space definition as well. Recall that a half space in $\mathbb{R}^N$ can be represented using linear inequalities. For example, all points $x = (x_1, ..., x_N) \in \mathbb{R}^N$ such that

$$a_1x_1 + a_2x_2 + ... + a_Nx_N \leq c$$

for constants $c, a_i \in \mathbb{R}$ $(1 \leq i \leq N)$. Of course, one such inequality produces a convex set and so the intersection of two of these half spaces will also be convex. Doing this a finite number of times produces a convex polytope under the half space representation.

Definition 3.4. Let $A = (a_{ij}) \in \mathbb{R}^{m \times N}$ be a matrix and let $b \in \mathbb{R}^m$. Then a (closed) **convex polytope** is the set of solutions to the system of linear inequalities

$$Ax \leq b$$

Of course, unlike the vertex representation, this definition can result in unbounded convex polytopes. Although this work is not concerned with substantial convex analysis, we will need one additional tool for some later results.

Definition 3.5. Given a convex set $C \subset \mathbb{R}^N$, the **recession cone** of C, denoted $\operatorname{recc} C$, is the set of all directions $d \in \mathbb{R}^N$ such that all rays eminating from points in C in the direction d remain in C. That is

$$\operatorname{recc} C = \{d \in \mathbb{R}^N : \forall x \in C, \forall \lambda \geq 0 : x + \lambda d \in C\}$$

Obviously for bounded convex sets the recession cone is trivial, i.e. $\operatorname{recc}(C) = \{0\}$. However, not all unbounded convex sets have non-trivial recession cones. As an example, take the sets

$$A = [0, 1] \times [0, \infty)$$

$$B = [0, 1) \times [1, \infty) \cup [0, 1] \times [0, 1]$$

Here, we have $\text{recc}(A) = \{(0, y) : y \geq 0\}$ while $\text{recc}(B) = \{0\}$. As noted by Alexander Schrijver in [51], the recession cone has the following equivalent definition for a convex polytope C satisfying $Ax \leq b$,

$$\text{recc}(C) = \{x \in \mathbb{R}^N : Ax \leq 0\}$$

The following lemma is also from [51] and will be useful in a later section.

Lemma 3.6. *A closed, convex polytope C is bounded if and only if* $\text{recc}(C) = \{0\}$.

Of course, the set B defined above is not a counter example to this lemma since it is not closed.

We are now ready to turn our attention back to Voronoi diagrams and prove that the first order Voronoi diagram consists of convex polytopes. We will then extend this result to higher order Voronoi diagrams.

Theorem 3.7. *Let $\mathcal{X} \subset \mathbb{R}^N$ be a point cloud. Then $V^1(\mathcal{X})$ is a set of convex polytopes.*

Proof. Let $\mathcal{X} = \{x_1, ..., x_n\}$ and let $x_i \in \mathcal{X}$. For each $x_j \in \mathcal{X}$ with $i \neq j$, consider the Voronoi diagram induced by the set $\{x_i, x_j\}$, denoted V_{ij}. Let $H_j^i = \{u \in \mathbb{R}^N : d(u, x_i) \leq d(u, x_j)\}$ be the region of this diagram corresponding to points closer to x_i than x_j. Clearly, V_{ij} bisects the domain with a hyper-plane, creating two half spaces H_j^i and $\mathbb{R}^N \backslash H_j^i$. Now notice that the set $H_1^i \cap H_2^i \cap ... \cap \hat{H}_i^i \cap ... \cap H_n^i$ is in fact the Voronoi region, $V^1(x_i)$, about x_i (here, $\hat{H}_i^i$ denotes the absence of H_i^i from the intersection). To see this, suppose $u \in V^1(x_i)$. Then u is closer to x_i than any point in $\{x_1, ..., \hat{x}_i, ..., x_n\}$. That is, $d(u, x_i) \leq d(u, x_j)$ for all $j \neq i$. Therefore, u is in H_j^i for all j, and so $u \in H_1^i \cap ... \cap \hat{H}_i^i \cap ... \cap H_n^i$. Conversely, suppose $u \in H_1 \cap ... \cap \hat{H}_i^i \cap ... \cap H_n$. Then $d(u, x_i) \leq d(u, x_j)$ for all $j \neq i$ and so $u \in V^1(x_i)$. Therefore, $V^1(x_i) = H_1 \cap ... \cap \hat{H}_i^i \cap ... \cap H_n$. Finally, since $V^1(x_i)$ is an intersection of a finite number of half-spaces, $V^1(x_i)$ is a convex polytope. Since i was chosen arbitrarily, the result holds for all the Voronoi regions. □

This convexity result gives us much more than simply convexity in the case of 1^{st}-order Voronoi diagrams, since the recursive algorithm can be employed to construct higher order diagrams. Here, we establish the critical result that Voronoi regions in higher dimensional Voronoi diagrams are convex polytopes. This will be important when analyzing the forthcoming gradient flow since trajectories under this flow will be shown to be piecewise straight lines in the Voronoi regions.

Theorem 3.8. *Let $\mathcal{X} \subset \mathbb{R}^N$ be a point cloud. Then the k-order Voronoi diagram of $\mathcal{X}$ is a set of convex polytopes.*

Proof. Let $V^k(\mathcal{X})$ be the k-order Voronoi diagram of a point cloud $\mathcal{X}$ and consider a k-order Voronoi region $V \in V^k(\mathcal{X})$ with generators $x_1, ..., x_k \in \mathcal{X}$. Any point $x \in V$ is closer to the points $x_1, ..., x_k$ than any other generator $y \in \mathcal{X}$. Consider the subsets $\mathcal{X}_i \subset \mathcal{X}$ for $1 \leq i \leq k$, where $\mathcal{X}_i = \mathcal{X} \backslash \{x_1, ..., \hat{x}_i, ..., x_k\}$ and $\hat{x}_i$ denotes the absence of x_i. Then for the first order Voronoi diagram $V^1(\mathcal{X}_i)$, the Voronoi region V_i, with generator x_i, is convex by Theorem 3.7. Additionally, the region V_i consists of all points whose distance to x_i is less than or equal to the distance to any other point in $\mathcal{X} \backslash \{x_1, ..., x_k\}$. Doing this for every $1 \leq i \leq k$ yields a set of convex polytopes $\{V_1, ..., V_k\}$. Since each V_i is convex, the set $V_1 \cap ... \cap V_k$ is also convex. We claim $V_1 \cap ... \cap V_k = V$, and so V is convex.

To see this, let $x \in V_1 \cap ... \cap V_k$. Then since $x \in V_i$ for each $1 \leq i \leq k$, we know the distance from x to x_i is less than or equal to the distance between x and any point in $\mathcal{X} \backslash \{x_1, ..., x_k\}$. Since this holds for every $1 \leq i \leq k$, we know

$$\max_{1 \leq i \leq k} d(x, x_i) \leq \min_{y \in \mathcal{X} \backslash \{x_1, .., x_k\}} d(x, y)$$

Thus, we must have $x \in V$ and so $V_1 \cap ... \cap V_k \subseteq V$. Conversely, if $x \in V$, then by the definition of a Voronoi region, for any $1 \leq i \leq k$, we know the distance from x to x_i is less than or equal to the distance from x to any point in $\mathcal{X} \backslash \{x_1, ..., x_k\}$. Hence, we have $x \in V_i$ for all $1 \leq i \leq k$. Since this holds for any $1 \leq i \leq k$, we see that $x \in V_i$ for every $1 \leq i \leq k$.

Thus, $x \in V_1 \cap ... \cap V_k$ and so $V \subseteq V_1 \cap ... \cap V_k$. Therefore, we have $V = V_1 \cap ... \cap V_k$. Since V was an arbitrary k-order Voronoi region in the Voronoi diagram of $\mathcal{X}$, we see that all Voronoi regions in the diagram must be convex. □

Note that unlike the 1^{st}-order Voronoi regions, a k^{th}-order Voronoi region with $k > 1$ can be empty. This will happen when the regions $\{V_i\}_{i=1}^k$, as defined in the proof, do not intersect. The following result is not surprising, but worth stating explicitly as it characterizes when a higher order Voronoi region can be empty.

Lemma 3.9. *Let $\mathcal{X} \subset \mathbb{R}^N$ be a point cloud. A k^{th}-order Voronoi region V generated by $\text{gen}(V) = \{x_1, ..., x_k\} \subset \mathcal{X}$ has nonempty interior if and only if there exists a point $x \in \mathbb{R}^N$ and $r > 0$ such that the open ball $B = B(x, r)$ centered at x of radius r contains $\text{gen}(V)$, but $B \cap (\mathcal{X} \setminus \text{gen}(V)) = \emptyset$.*

Proof. First suppose there exists $x \in \mathbb{R}^N$ and $r > 0$ such that $\text{gen}(V) \subseteq B(x, r) = B$ but $B \cap (\mathcal{X} \setminus \text{gen}(V)) = \emptyset$, then the nearest neighbors of x are precisely the points in $\text{gen}(V)$, i.e. $\text{NN}_{\mathcal{X}}^k(x) = \text{gen}(V)$. Thus, $x \in V$ and so V is nonempty. Furthermore, we must have $x \notin \partial V$ since if x is in ∂V then it lies on some Voronoi face. Since any point on a Voronoi face of $V^k(\mathcal{X})$ has an ambiguous nearest neighbor set, we know that x must lie within the interior of V and hence V has non-empty interior.

Conversely, pick a point x in the interior of V such that $u \notin \text{gen}(V)$. Then $\text{NN}_{\mathcal{X}}^k(x) = \text{gen}(V)$. Define

$$R_I = \max_{y \in \text{gen}(V)} \|x - y\| \quad \text{and} \quad R_E = \min_{u \in \mathcal{X} \setminus \text{gen}(V)} \|x - u'\|$$

Note that $R_I > 0$ since $x \notin \text{gen}(V)$. Also note that $R_I < R_E$ since $\text{gen}(V) = \text{NN}_{\mathcal{X}}^k(x)$ and $|\,\text{NN}_{\mathcal{X}}^k(x)| = k$ (since $x \in \text{int}(V)$). Then pick

$$r \in (R_I, R_E)$$

Since the set $\mathrm{NN}^k_{\mathcal{X}}(x)$ is well defined (i.e. $|\mathrm{NN}^k_{\mathcal{X}}(x)| = k$), we know $\|x - z\| > r$ for any $z \in \mathcal{X} \backslash \mathrm{gen}(V)$. Thus, $B(x,r) \cap (\mathcal{X} \backslash \mathrm{gen}(V)) = \emptyset$. On the other hand, $\mathrm{gen}(V) \subseteq B(x,r)$ by our choice of r (since $r > R_I$). Hence, $B(x,r)$ satisfies the requirements of the lemma and thus the lemma is proved. □

Every Voronoi region V in the Voronoi diagram is a convex polytope as shown in Theorem 3.8. Using this fact, we know that the regions are simply connected and in particular, we can compute the barycenter of the polytope, V. However, since each polytope has a set of generators, we can define another notion of the barycenter of V, by taking the barycenter of the generators. The distinction between these two types of barycenters will be important later on, so we state some definitions now and give an example to emphasize how the two notions of barycenters differ.

Definition 3.10. For a k-order Voronoi region V generated by $\{x_1, ..., x_k\}$, let $\mathrm{Bar}_G(V)$, called the **generator barycenter**, be the barycenter of $\{x_1, ..., x_k\}$. That is,

$$\mathrm{Bar}_G(V) = \frac{1}{k} \sum_{i=1}^{k} x_i$$

On the other hand, we will denote the barycenter of the convex polytope V, if it exists, by $\mathrm{Bar}_P(V)$ and refer to this as the **polytope generator** of V. That is,

$$\mathrm{Bar}_P(V) = \frac{\int_{\mathbb{R}^N} x I_V(x) dx}{\int_{\mathbb{R}^N} I_V(x) dx}$$

where I_V is the characteristic function of V and the integrals are taken coordinate-wise. This integral will not converge for unbounded polytopes and hence $\mathrm{Bar}_P(V)$ will not always exist.

It is important to note that while the polytope barycenter of a k^{th}-order Voronoi region V is always contained in V, the generator barycenter of a k^{th}-order Voronoi region V need

not belong to V when $k > 1$. This is obviously quite different than 1^{st}-order Voronoi diagrams, where the single generator (and hence also the generator barycenter) is contained well within the Voronoi regions. This condition will be very important in what follows, when we find fixed points of the k-nearest neighbors flow. The next example highlights how this condition can arise.

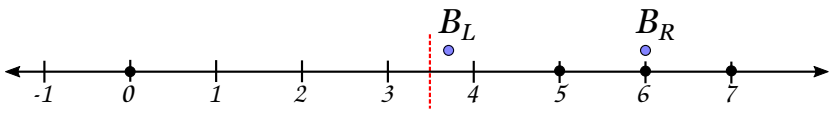


Figure 3.3: Example with $\text{Bar}_G(V) \notin V$

Example 3.1. Consider Figure 3.3 and let $\mathcal{X} = \{0, 5, 6, 7\} \subset \mathbb{R}$ with $k = 3$. When $k = |\mathcal{X}| - 1$, the k-order Voronoi diagram is known as the **furthest point Voronoi diagram**. This is because we are really partitioning the space into regions where every point in a region has the same furthest point. Using this fact, we know there will be two 3-order Voronoi regions V_1, V_2 induced by $\text{gen}(V_1) = \{0, 5, 6\}$ and $\text{gen}(V_2) = \{5, 6, 7\}$ respectively. We can compute the generator barycenters

$$\text{Bar}_G(V_1) = \frac{1}{3}(0 + 5 + 6) = \frac{11}{3}$$

$$\text{Bar}_G(V_2) = \frac{1}{3}(5 + 6 + 7) = \frac{18}{3} = 6$$

In the figure we let $B_L = \text{Bar}_G(V_1)$ and $B_R = \text{Bar}_G(V_2)$. We can find the point of separation between V_1 and V_2 by noticing that if $x \geq 3.5$, then x will be closer to 7 than it will to 0. Hence, we have

$$V_1 = \left\{ x \in \mathbb{R} : x \leq \frac{7}{2} \right\} \qquad \text{and} \qquad V_2 = \left\{ x \in \mathbb{R} : x \geq \frac{7}{2} \right\}$$

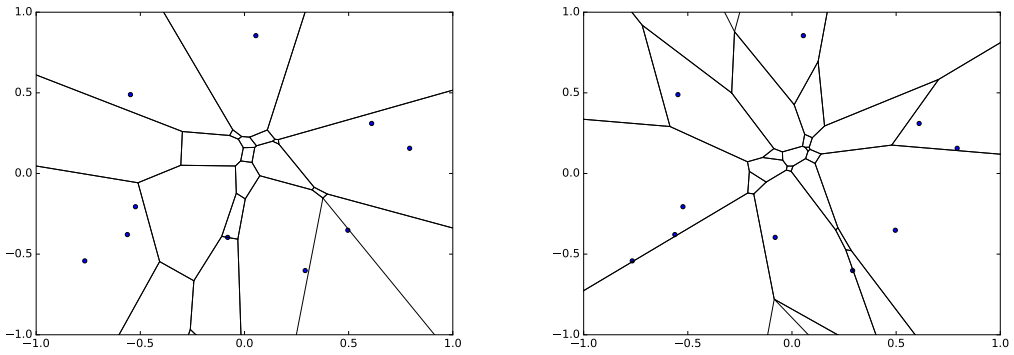
and so $\text{Bar}_G(V_1) \notin V_1$ while $\text{Bar}_G(V_2) \in V_2$. In this case, since the two regions V_1 and V_2 are unbounded, neither has a polytope generator.

It should be clear from the definitions that the k^{th}-order Voronoi diagram depends heavily on the value of k. Different values of k can produce wildly different higher order Voronoi diagrams. As we will see in this dissertation, the choice of k requires careful thought as the gradient flow we induce later in this chapter depends on the particular value of k. In the next example, we show how two different values of k produce two quite different diagrams.

Example 3.2. Here we will present the first part of an ongoing example. We have generated a point cloud $\mathcal{X}$ of 10 points sampled from $[0,1] \times [0,1]$ at random, we then computed the third and fifth order Voronoi diagrams. The diagrams $V^3(\mathcal{X})$ and $V^5(\mathcal{X})$ are shown in Figure 3.4. In these diagrams, the original points of $\mathcal{X}$ are shown as black dots and the lines correspond to the Voronoi edges. Note the difference between the two diagrams. Also, notice that there are many regions which do not contain any of the original points. There are also areas of the diagram with many small, densely packed regions. As we will see, these regions result in very jittery and seemingly chaotic trajectories, however their behavior can be completely determined.

3.2 The k-Nearest Neighbor Gradient Flow

In this section, we will introduce the k-nearest neighbor flow and investigate some of its properties. The goal of introducing this flow is to reduce the noise in a point cloud and to reduce the influence of outliers on manifold reconstruction techniques. While many smoothing techniques simply remove outliers, our technique will move the outliers to less noisy positions. Therefore, we will not reduce the number of points in the sample and will use every point, as best we can. Additionally, since the k-nearest neighbor flow is a gradient descent based algorithm, the complexity of the algorithm will be tied to the complexity of computing the gradient, which only depends linearly on the dimensionality and size of



(a) $k = 3$ (b) $k = 5$

Figure 3.4: k^{th}-order Voronoi diagrams for samples drawn from $[0, 1]^2$

the neighborhood (i.e. k). Of course, computing the neighborhood itself is a quadratic operation when performed naïvely since it requires the computation of the distance matrix. Thus, this step dominates the computational complexity of the gradient flow procedure. To address this, we will discuss the use of space paritioning algorithms to decrease the overall complexity of the gradient flow. However, we leave this modification to a later chapter.

3.2.1 Local k-Nearest Neighbors Gradient Flow

Let $\mathcal{X}$ be a finite point set with $|\mathcal{X}| = n$. We begin by observing that the definition of the distance-to-measure function under the empirical measure is equivalent to a k-nearest neighbors energy function $E^k_{\mathcal{X}} : \mathbb{R}^N \to \mathbb{R}$ defined as follows,

$$E^k_{\mathcal{X}}(x) = \frac{1}{k} \sum_{y \in \text{NN}^k_{\mathcal{X}}(x)} \|x - y\|^2 \tag{3.2}$$

To see this, we set $\mu = \mu_{\mathcal{X}}$ in Definition 1.8 of the distance-to-measure function d_{μ,m_0}, where $m_0 \in \mathbb{R}$ is some positive mass parameter and $\mu_{\mathcal{X}}$ is the empirical measure . That is,

for any Borel set $B \subseteq \mathbb{R}^N$, we have

$$\mu_{\mathcal{X}}(B) = \frac{1}{n}|B \cap \mathcal{X}|$$

As the definition of the distance-to-measure function relies on the pseudo distance function $\delta_{\mu_{\mathcal{X}},m}(\cdot)$ we first turn to Definition 1.7. Using the empirical measure $\mu_{\mathcal{X}}$, we have

$$\begin{aligned}
\delta_{\mu_{\mathcal{X}},m_0}(x) &= \inf\left\{r > 0 : \mu_{\mathcal{X}}(\bar{B}(x,r)) > m_0\right\} \\
&= \inf\left\{r > 0 : \frac{1}{n}|\bar{B}(x,r) \cap \mathcal{X}| > m_0\right\} \\
&= \inf\left\{r > 0 : |\bar{B}(x,r) \cap \mathcal{X}| > m_0 n\right\}
\end{aligned}$$

Suppose for some r, $|\bar{B}(x,r) \cap \mathcal{X}| > m_0 n$. Then there exist points $x_1, x_2, ..., x_j \in \mathcal{X}$, for some $j > m_0 n$, such that $x_i \in \bar{B}(x,r)$ for $i = 1, ..., j$. Hence, if we set $j = \lceil m_0 n \rceil$, then the pseudo distance-to-measure function simplifies to a j^{th}-nearest neighbor query on $\mathcal{X}$. That is, $\delta_{\mu_{\mathcal{X}},k/n}(x)$ is the distance to the j^{th}-nearest neighbor of x. Also, since $|\bar{B}(x,r) \cap \mathcal{X}| \in \mathbb{N}$, we know the map $m_0 \mapsto \delta_{\mu_{\mathcal{X}},m_0}$ is constant for all $m_0 \in \left(\frac{j}{n}, \frac{j+1}{n}\right)$. Therefore,

$$d^2_{\mu,m_0}(x) = \frac{1}{m_0}\int_0^{m_0} \delta_{\mu,m}(x)^2 dm = \frac{1}{m_0 n}\sum_{j=1}^{m_0 n} \delta_{\mu,j/n}(x)^2 = \frac{1}{k}\sum_{y \in \mathrm{NN}^k_{\mathcal{X}}(x)} \|y - x\|^2$$

where $k = \lceil m_0 n \rceil$. Thus, we have $d^2_{\mu,m_0}(x) = E^k_{\mathcal{X}}(x)$ as desired.

For the remainder of this section, we will fix k and forget about the parameter m_0 since specifying m_0 yields a value for k. Additionally, in our notation for $E^k_{\mathcal{X}}$, we will suppress the explicit dependence on k and simply write $E_{\mathcal{X}}$.

For a point $u \in \mathbb{R}^N$, we can calculate the distance-to-measure under the empirical

measure $\mu_{\mathcal{X}}$ using the k-nearest neighbor formula above. If we wanted to move the point u so that we reduce its distance-to-measure $\mu_{\mathcal{X}}$, then we would want to move opposite the gradient of $E_{\mathcal{X}}$. This is the idea behind the k-nearest neighbors gradient flow.

First, note that the gradient is given by

$$\nabla E_{\mathcal{X}}(u) = \frac{2}{k} \sum_{y \in \mathrm{NN}_{\mathcal{X}}^k(u)} (u - y) \tag{3.3}$$

where $\mathrm{NN}_{\mathcal{X}}^k(u)$ is the set of k-nearest neighbors to u. This can be computed directly from the definition of the distance-to-measure under the empirical measure. However, it can also be obtained from the gradient in Corollary 1.14. Note that $\nabla E_{\mathcal{X}}$ is discontinuous across the k-order Voronoi faces in general, since the neighbors determining the gradient will change. It is undefined on the faces (since $\mathrm{NN}_{\mathcal{X}}^k$ is undefined on the faces). We will discuss the flow's behavior along the boundary later. Additionally, we will only consider trajectories intersecting with codimension one Voronoi faces since lower dimensional Voronoi face intersections occur with measure zero and will therefore only occur with synthetic and pathological point clouds. Note that although the co-dimension one Voronoi faces have measure zero in the ambient space $\mathbb{R}^N$, the set of initial points whose trajectories intersect these faces does not have measure zero in the space of trajectories. For now we will only consider the behavior of trajectories within higher order Voronoi regions, leaving the definition of gradient flow across the Voronoi skeleton for later.

Definition 3.11. Given a k^{th}-order Voronoi region $V \in V^k(\mathcal{X})$, we induce a gradient flow on V as follows. For every $x \in \mathrm{int}(V)$, we define $u : [0, T_x) \to \mathbb{R}^N$ for some $T_x > 0$, such that

$$\begin{cases} \dfrac{du}{dt} = -\nabla E_{\mathcal{X}}(u) \\ \\ u(0) = x \end{cases} \tag{3.4}$$

We then set

$$\varphi_\ell(x,t) = u(t) \qquad \forall (x,t) \in V \times [0, T_x) \tag{3.5}$$

and call the resulting system, the **local *k*-nearest neighbor flow**. We choose T_x such that $\varphi_\ell(x,t) \in V$ for all $t \leq T_x$. That is, once the trajectory $u(t)$ intersects the boundary ∂V, we terminate the flow.

Note that using the above system, we can induce a single trajectory for each point in $\mathcal{X}$ and track its evolution under the local flow. In particular, we set

$$\mathcal{X}_t^\ell = \{\varphi_\ell(x,t) : x \in \mathcal{X}\}$$

However, for the purpose of computing nearest neighbors, we will always use the original points in the point cloud $\mathcal{X}$. That is, when computing $\nabla E_{\mathcal{X}}(u(t))$ for some time $t \geq 0$, we use the nearest neighbors from the point cloud $\mathcal{X}_0^\ell = \mathcal{X}$ instead of the point cloud $\mathcal{X}_t^\ell$. While it is certainly possible to compute nearest neighbors using $\mathcal{X}_t^\ell$, doing so results in strong clustering behavior as the points continually flow toward one another. As we will see, using $\mathcal{X}_0^\ell$ to compute nearest neighbors for the flow still induces some clustering, however using $\mathcal{X}_t^\ell$ makes the effect far more pronounced. Furthermore, it is obviously more computationally expensive to update the nearest neighbors at every iteration since any spatial indexing scheme would require updating the index at every step.

3.2.2 Defining the Flow on the Boundary

In this section, we address the behavior of the flow along the Voronoi faces. In the previous definitions of the k-nearest neighbor flow, we have only described the flow locally, constrained to a single k-order Voronoi region. Here, we will extend the flow across regions by investigating the behavior of the gradients across the boundary.

Given a k-order Voronoi diagram, $V^k(\mathcal{X})$, recall that $\partial V^k(\mathcal{X})$ denotes the **skeleton** of the diagram. The issue with defining the flow along $\partial V^k(\mathcal{X})$ is the discontinuity of the

gradient across these boundaries. This is a consequence of the nearest neighbor set changing as we cross from one region to another. As we will now see, extending the gradient flow system of Equation 3.5 across the skeleton of the k-order Voronoi diagram naturally results in a Filippov type system. With this in mind, we define the gradient along $\partial V^k(\mathcal{X})$ in a manner consistent with Fillipov systems.

We will follow the nomenclature of M. di Bernardo et al. and their discussion of piecewise-smooth dynamical systems (see [24]). First, note that the local gradient flow system described by Equation 3.5 comes close to satisfying the definition of a piecewise-smooth flow as defined in [24], and reproduced below.

Definition 3.12. A **piecewise-smooth flow** is given by a finite set of ordinary differential equations defined on sets $S_i \subset \mathbb{R}^N$,

$$\dot{x} = F_i(x, \mu), \qquad \text{for } x \in S_i$$

where $\cup_i S_i \subseteq \mathbb{R}^N$ and each S_i has a non-empty interior. The intersection $\Sigma_{ij} := \bar{S}_i \cap \bar{S}_j$ is either an $\mathbb{R}^{N-1}$-dimensional manifold included in the boundaries ∂S_j and ∂S_i, or is the empty set. Each vector field F_i is smooth in both the state x and the parameter μ, and defines a smooth flow $\Phi_i(x, t)$ within any open set $U \supset S_i$. In particular, each flow Φ_i is well defined on both sides of the boundary ∂S_i.

To see the connection with our gradient flow, note that we can take the sets $\{S_i\}$ to be the finitely many k-order Voronoi regions $V \in V^k(\mathcal{X})$. For this section, we let $m = |V^k(\mathcal{X})|$ and index the k-order Voronoi regions so that for each $V \in V^k(\mathcal{X})$, there exists $i \in \{1, ..., m\}$ such that $V_i = V$, to better match the notation in [24]. Returning to the definition, our system is certainly smooth within any k-order Voronoi region V and if we extend the gradient to any open set $U \supset V$, it will clearly remain smooth, afterall the gradient is simply a vector directed toward $\text{Bar}_G(V)$. Unlike the above definition, our system does not utilize any parameters μ, however this does not change the conclusion that our gradient

system almost defines a piecewise-smooth gradient flow.

A non-empty boundary Σ_{ij} between two k-order Voronoi regions $V_i, V_j \in V^k(\mathcal{X})$ will be referred to as a **switching manifold**. Of course, the switching manifolds will be piecewise-smooth as well, given they are the faces of convex polytopes. In what follows, as in the reference work [24], we will only consider codimension one switching manifolds. This is justified since the set of points whose trajectories intersect switching manifolds with codimension greater than one is a null set.

To characterize the smoothness of a piecewise-smooth dynamical system, M. di Bernardo et al. use the concept of *degree of smoothness*. In the following definition, reproduced from [24], and the next, we will continue to use the notation of M. di Bernardo (i.e. the notation from Definition 3.12). After presenting the definitions, we will switch back to our notation.

Definition 3.13. The **degree of smoothness** at a point x_0 in a switching set Σ_{ij} of a piecewise-smooth ordinary differential equation is the highest order r such that the Taylor series expansions of $\Phi_i(x_0, t)$ and $\Phi_j(x_0, t)$ with respect to t, evaluated at $t = 0$, agree up to terms of $O(t^{r-1})$. That is, the first non-zero partial derivative with respect to t of the difference $[\Phi_i(x_0, t) - \Phi_j(x_0, t)]|_{t=0}$ is of order r.

The above definition describes the smoothness of the system at a single point. Since we are concerned with defining the gradient flow vector along all of $\partial V^k(\mathcal{X})$, we need another definition. To further characterize the smoothness, we can use the following definition from [24].

Definition 3.14. A switching set Σ_{ij} is said to be **uniformly discontinuous** in some domain $\mathcal{D}$ if the degree of smoothness of the system is the same for all points $x \in \Sigma_{ij} \cap \mathcal{D}$. We say that the discontinuity is **uniform with degree m** if the first non-zero partial derivative of $F_i - F_j$ evaluated on Σ_{ij} is of order $m - 1$. Furthermore, the degree of smoothness is one if $F_i(x) - F_j(x) \neq 0$ for $x \in \Sigma_{ij} \cap \mathcal{D}$.

From the above two definitions, we see that our switching sets $\Sigma_{ij} = V_i \cap V_j$ have degree of smoothness one. This is due to the fact that, in general, the gradients ∇E_i and ∇E_j

corresponding to V_i and V_j respectively, do not agree on Σ_{ij}. When dealing with systems having switching sets with degree of smoothness one, care must be taken when defining the gradient along the switching sets. This is because the gradients on either side of a switching set may be inducing flow in opposing directions. Piecewise-smooth flows exhibiting degree of smoothness one are called **Fillipov systems**.

Defining the gradient along the switching manifolds becomes important in Filippov systems. In this work, we follow Filippov's convex method. To utilize Filippov's convex method, we must define the **sliding region**, $\hat{\Sigma}_{ij}$ of the switching manifold $\Sigma_{ij} = \partial V_i \cap \partial V_j$. First, let η_x denote the normal vector of the codimension one hyperplane $\partial V_i \cap \partial V_j$. In particular, we choose the normal vector which points into V_i. Let $E_i(x)$ and $E_j(x)$ be the gradient $E^k_{\mathcal{X}}(x)$ as defined for V_i and V_j respectively (i.e. induced by $\text{gen}(V_i)$ and $\text{gen}(V_j)$). Then we can define the sliding region, $\hat{\Sigma}_{ij}$ of the switching manifold Σ_{ij} to be

$$\hat{\Sigma}_{ij} = \{x \in \Sigma_{ij} : \langle \eta_x, \nabla E_i(x) \rangle \cdot \langle \eta_x, \nabla E_j(x) \rangle < 0\}$$

Thus, points in the sliding region $\hat{\Sigma}_{ij}$ consist of points where the gradients on either side of Σ_{ij} point into different k-order Voronoi regions.

For points $x \in \Sigma_{ij} \backslash \hat{\Sigma}_{ij}$, the definition of the gradient is straightforward since both gradients point into the same k-order Voronoi region. In particular, we will just follow the flow and if a trajectory leaves V_1 and enters V_2, then we define the gradient on $\partial V_1 \cap \partial V_2$ to be the gradient induced by the generators of V_2 (i.e. $\text{gen}(V_2)$). However, on $\hat{\Sigma}_{ij}$, we define the gradient $\nabla E^k_{\mathcal{X}}(X)$ to be a convex combination of the gradients on either side of $\partial V_i \cap \partial V_j$. Thus, for a point $x \in \hat{\Sigma}_{ij}$, we set

$$\nabla E^k_{\mathcal{X}}(x) = (1 - \alpha(x))\nabla E_i(x) + \alpha(x)\nabla E_j(x) \quad \text{for all } x \in \Sigma_{ij} \tag{3.6}$$

We then define $\alpha : \mathbb{R}^N \to \mathbb{R}$ as in [24], setting

$$\alpha(x) = \frac{\langle \eta_x, \nabla E_i(x) \rangle}{\langle \eta_x, \nabla (E_i - E_j)(x) \rangle}$$

This choice of $\alpha(x)$ causes the gradient $\nabla E^k_{\mathcal{X}}(x)$ to lie orthogonal to η_x. This is illustrated in Figure 3.5, which shows the resulting gradient (red arrow) for two regions V_i and V_j which exhibit attractive sliding along the interface.

In [24], Bernardo et al. show that the above definition of $\hat{\Sigma}_{ij}$ is identical to the following,

$$\hat{\Sigma}_{ij} = \{x \in \Sigma_{ij} : 0 \leq \alpha \leq 1\}$$

There are two modes of sliding which can occur in the sliding region. Recall that in the region V_i (resp. V_j), the induced flow follows the vector field given by $-\nabla E_i$ (resp. $-\nabla E_j$). Note that if $\langle \eta_x, -\nabla E_i \rangle < 0$ and $\langle \eta_x, -\nabla E_j \rangle > 0$, then the the system exhibits *attractive sliding* at the point x, since η_x was chosen to point into V_i. Conversely, if $\langle \eta_x, -\nabla E_i \rangle > 0$ and $\langle \eta_x, -\nabla E_j \rangle < 0$, then the system exhibits *repulsive sliding* at x.

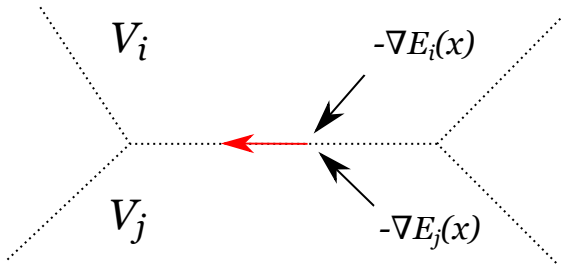


Figure 3.5: Attractive sliding along the interface $\partial V_i \cap \partial V_j$

Repulsive sliding does not pose any problems evolving the point cloud in forward time since these interfaces cannot be reached in forward time and the probability that a point in the point cloud starts on the interface between two k-order Voronoi regions is zero. On the

other hand, attractive sliding is important since regardless of whether a point starts on the interface, it may be drawn to the interface, at which point it will begin sliding. If points can slide along an interface, then two points can enter the sliding regions at different points and exit the sliding region at the same point. Thus, trajectories are not well defined in reverse time. However, as we will see in the next section, attractive sliding does not occur for any point cloud undergoing this flow. Repulsive sliding, on the other hand, does occur. In fact, repulsive sliding regions provide the unstable regions of the gradient flow system. In particular, an repulsive sliding region acts as a separatrix, with points on opposing sides following entirely different trajectories.

Now that we have all we need to define the gradient flow system almost everywhere in $\mathbb{R}^N$, we put off analyzing the sliding region dynamics in favor of updating our definitions and notation.

Definition 3.15. To define the trajectory $u(t)$ for a point $x \in \mathbb{R}^N$, we solve the following differential equation.

$$\begin{cases} \dfrac{du}{dt}(t) = -\nabla E_{\mathcal{X}}(u(t)) & t > 0 \\ \\ u(t) = x & t = 0 \end{cases} \tag{3.7}$$

where $\nabla E_{\mathcal{X}}$ follows the form given in Equation 3.3 if $x \notin \Sigma$ and the form given in Equation 3.6 when $x \in \Sigma$. Next, we let

$$\phi(x, t) = u(t) \tag{3.8}$$

where u is the trajectory defined above for a given $x \in \mathbb{R}^N$. Given a point cloud X, we define

$$X_t = \{\phi(x, t) : x \in X\} \tag{3.9}$$

for all $t \geq 0$.

As we increase t, every point in the point cloud will move in the direction which minimizes its distance-to-measure. For the remainder of the dissertation, we will drop the

notation ϕ_ℓ and X_t^ℓ in favor of ϕ and X_t respectively. The local gradient flow has served its purpose and we are now interesed in analyze the entire flow as defined on $\mathbb{R}^N$.

3.3 Properties of the k-Nearest Neighbor Flow

The gradient flow system in Equation 3.8 will be the bedrock of the rest of this work. In what follows, we will introduce many novel modifications of this gradient system which have not been applied in this context previously. In doing so, we will improve the overall performance of the algorithm. Despite these modifications, the underlying gradient based flow will remain largely the same. We will therefore spend some effort analyzing this system. In the remainder of this chapter, we will find the solutions of the above equations and describe a flow diagram which completely encapsulates the dynamics of the system.

To start, we will determine the fixed points of the k-nearest neighbor flow. As it turns out, these points are easy to determine using the k-order Voronoi diagram. However, without the diagram, the process becomes computationally expensive. Therefore, computing these fixed points in higher dimensions is intractable. However, we will present a simple theoretical procedure to do so. We can easily conclude a new result giving necessary and sufficient conditions for a Voronoi region to be a positively invariant set of the k-nearest neighbor flow given in Definition 3.15.

Proposition 3.16. *Let $V^k(\mathcal{X})$ be the k-order Voronoi diagram for a finite point cloud $\mathcal{X} \subset \mathbb{R}^N$. Under the gradient flow induced by Equation 3.7, a region $V \in V^k(\mathcal{X})$ is a positively invariant set if and only if its interior contains the barycenter of its generators. That is*

$$\varphi(x,t) \in V \quad \forall x \in V, t \geq 0 \qquad \Leftrightarrow \qquad \mathrm{Bar}_G(V) \in \mathrm{int}(V)$$

Proof. Let $\mathrm{gen}(V) = \{x_1, ..., x_k\}$ be the k generators of V. Since all the points of V have the same k-nearest neighbors, we can solve the gradient flow exactly for all $x \in V$, until the

trajectory leaves V of course. In particular, for any $x \in V$ we have

$$\frac{du_x}{dt} = -\frac{2}{k}\sum_{i=1}^{k}(u_x - x_i) = -2u_x + \frac{2}{k}\sum_{i=1}^{k} x_i = -2u_x + 2\,\mathrm{Bar}_G(V)$$

which has the solution

$$u(t) = \mathrm{Bar}_G(V) + c_1 e^{-2t}$$

Thus, if $u(0) = x \in V$, we have $x = \mathrm{Bar}_G(V) + c_1$ and so $c_1 = x - \mathrm{Bar}_G(V)$. This yields the solution

$$\begin{aligned} u(t) &= \mathrm{Bar}_G(V) + (x - \mathrm{Bar}_G(V))e^{-2t} \\ &= xe^{-2t} + (1 - e^{-2t})\,\mathrm{Bar}_G(V) \\ &= x\lambda(t) + (1 - \lambda(t))\,\mathrm{Bar}_G(V) \end{aligned} \tag{3.10}$$

where $\lambda(t) = e^{-2t}$. This solution is valid so long as $u(t)$ stays in V. Note that $\lambda(t) \in (0, 1]$ for $t \in [0, \infty)$. Therefore, $\{u(t)\}_{t\geq 0}$ is simply a line segment from x to $\mathrm{Bar}_G(V)$. Since both x and $\mathrm{Bar}_G(V)$ are in V by assumption, and V is convex by Corollary 3.8, the region V must contain the entire line segment and so we have $u(t) \in V$ for all $t \geq 0$. Thus, the region V is a positively invariant set of the flow.

The converse is straightforward. Suppose that a k-order Voronoi region $V \in V^k(\mathcal{X})$ is a positively invariant set. Then for any point $x \in V$, we have $\varphi(x, t) \in V$ for all $t \geq 0$. We know the trajectory of x must begin at x and follow a straight line (by the above derivation of a trajectory's solution), ending at $\mathrm{Bar}_G(V)$. Since $\varphi(x, t) \in V$ for all $t \geq 0$, we know the line stays within V. Finally, since V is closed, we must have the end points of the trajectory line are contained in V as well. Thus, $\mathrm{Bar}_G(V) \in V$ as desired. □

In what follows, we will often refer to the condition $\mathrm{Bar}_G(V) \in V$ as the **barycentric sink condition**. Using this language, a generator barycenter is a sink of the gradient flow

system if and only if it satisfies the barycentric sink condition. We refer to such barycenters as **barycentric sinks**. Note that we have also proved something more general about trajectories under the gradient flow defined in Equation 3.7: a point in $\mathbb{R}^N$ flows towards the generator barycenter of the Voronoi region containing the point. This of course happens until the point encounters the boundary of the Voronoi region. We will soon address the flow's behavior on the boundary of a Voronoi region, but first we state a lemma which follows immediately from the previous proof and the fact that φ is smooth within the interior of the Voronoi regions.

Lemma 3.17. *Given a point $x \in \text{int}(V)$, for some Voronoi region $V \in V^k(\mathcal{X})$, there exists $\tau > 0$ such that $\varphi(x,t) \in V$ for all $t < \tau$. Furthermore, for any $x \in V$, the set $\{\varphi(x,t) : t < \tau\}$ is a line segment from x towards $\text{Bar}_G V$.*

The previous results immediately lead to another new theorem which characterizes the sinks of the gradient flow system. Furthermore, we can use the barycentric sink condition to determine some positively invariant sets of the flow. Recall, a set M is **positively invariant** under the flow $\phi : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}$ if for all $x \in M$, we have $\phi(x,t) \in M$ for any $t \geq 0$.

Theorem 3.18. *Let $\mathcal{X} \subset \mathbb{R}^N$ be a finite point cloud and let $V^k(\mathcal{X})$ be the corresponding k-order Voronoi diagram. Let*

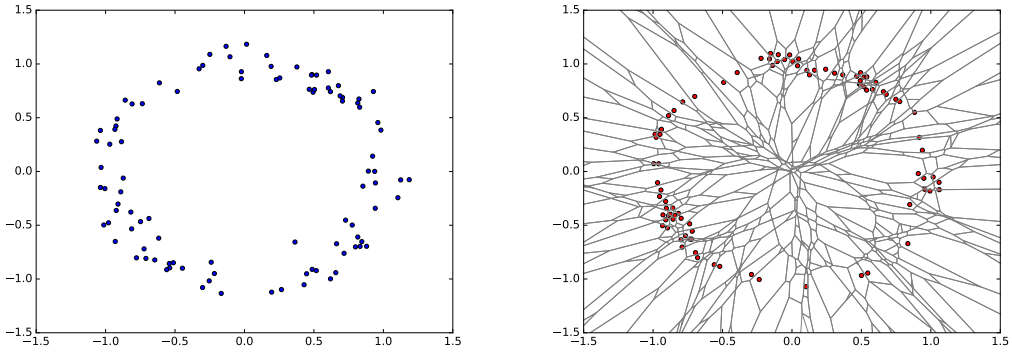
$$\mathcal{S}(\mathcal{X}) = \{V \in V^k(\mathcal{X}) : \text{Bar}_G(V) \in V\}$$

and let

$$\mathcal{B}(\mathcal{X}) = \{\text{Bar}_G(V) : V \in \mathcal{S}\}$$

Then all sets in $\mathcal{S}$ are positively invariant sets of the gradient flow defined in Equation 3.7 and $\mathcal{B}$ is the ω-limit set of the gradient flow system given by Equation 3.8.

In Figure 3.6, we show a sample point cloud drawn noisily from the unit circle S^1. The generator barycenters which act as sinks of the k-nearest neighbor gradient flow induced on



(a) Original Point Cloud (b) Barycentric Sinks

Figure 3.6: The barycentric sinks of a point cloud drawn from S^1

this point cloud are shown in Figure 3.6b. In this figure, the fifth-order Voronoi diagram is shown in gray. The points of $\mathcal{B}(\mathcal{X})$ are shown in red. Notice that there are some areas of $\mathbb{R}^2$ where the barycentric sinks are quite dense. This will lead to clustering when the point cloud evolves according to the k-nearest neighbor gradient flow.

While the above theorem provides us with some positively invariant sets, the system contains much larger positively invariant sets. To fully determine these sets, we will need to build a flow diagram in the next section. From this, we will be able to determine the regions of stability for each accumulation point in $\mathcal{B}$. It is important to note that this theorem makes clear that the final positions of all points in $\mathcal{X}$ under the gradient flow defined by Equation 3.7, are contained in $\mathcal{B}$. Of course, not every point in $\mathcal{B}$ will be approached by a point in $\mathcal{X}$, however these barycenters can be of interest themselves as we will see later.

Next, we introduce the concept of the **regional travel time**, $R(x)$, for a point x in a Voronoi region V, defined to be the time it takes the point x to escape V. That is

$$R(x) = \sup\{\tau > 0 : \varphi(x,t) \in V, \forall t < \tau\} \tag{3.11}$$

Furthermore, we can define the regional travel time $R(V)$ for a Voronoi region $V \in V^k(\mathcal{X})$,

to be

$$R(V) = \sup\{R(x) : x \in V\}$$

Of course, for a k-order Voronoi region V satisfying the barycentric sink condition, i.e. $\text{Bar}_G(V) \in V$, we will have $R(x) = \infty$ for all $x \in V$. On the other hand, if $\text{Bar}_G(V) \notin V$, then $R(x) < \infty$ for all $x \in V$.

3.3.1 Sliding Motion

As previously discussed, we are primarly interested in the attractive sliding motion case. For a point x on a sliding set Σ_{ij}, we know that the gradient from V_i pushes x toward $\text{Bar}_G(V_i)$ and the gradient from V_j pushes x toward $\text{Bar}_G(V_j)$. For attracting motion to occur, the vector from x to $\text{Bar}_G(V_i)$ must be in an opposing direction to the vector from x to $\text{Bar}_G(V_j)$. Note that since both V_i and V_j are convex polytopes, the intersection of their boundary must be a subset of some hyperplane L. Thus, if we let H_i (resp. H_j) denote the half space of $\mathbb{R}^N$ separated by L and containing $\text{int}(V_i)$ (resp. $\text{int}(V_j)$), then for attractive sliding motion to occur, we must have $\text{Bar}_G(V_i) \in L_j$ and $\text{Bar}_G(V_j) \in L_i$. Note that if $\text{Bar}_G(V_i) \in L_i$ and $\text{Bar}_G(V_j) \in L_j$, then we have repulsive sliding motion at x and if the barycenters are in the same half-space (that is H_i or H_j), then there is no sliding motion at all. We can now prove the following important theorem.

Theorem 3.19. *The gradient flow system described by Equation 3.8 does not exhibit any attractive sliding motion.*

Proof. For sake of contradiction, suppose that the interface between $V_i \in V^k(\mathcal{X})$ and $V_j \in V^k(\mathcal{X})$ exhibits attractive sliding motion. Since the interface $\partial V_i \cap \partial V_j$ between two k-order Voronoi regions is a codimension one hyperplane (V_i and V_j are convex polytopes), we can extend the interface to decompose $\mathbb{R}^N$ into two regions H_i and H_j, such that $V_i \subseteq H_i$ and $V_j \subseteq H_j$. Following our procedure in the previous section, we define η_x to be the normal vector to $\partial V_i \cap \partial V_j$ at the point $x \in \partial V_i \cap \partial V_j$ pointing into V_i, and thus into H_i as well. Of

course, the normal direction of $\partial V_i \cap \partial V_j$ is constant along $\partial V_i \cap \partial V_j$ (due to $\partial V_i \cap \partial V_j$ being a subset of a hyperplane), therefore, we will simply write η when referring to η_x. Given a point x on an $(N-1)$-dimensional subset of $\partial V_i \cap \partial V_j$, we have

$$r_i = \max_{y \in \text{gen}(V_i)} \|x - y\| = \max_{z \in \text{gen}(V_j)} \|x - z\| = r_j$$

To see why this is true, suppose without loss of genreality that $r_i < r_j$. Then there exists $z \in \text{gen}(V_j)$ such that $\|x - y\| < \|x - z\|$ for all $y \in \text{gen}(V_i)$. Since there are k points in $\text{gen}(V_i)$, the point z cannot be one of the k-nearest neighbors of x, contradicting the fact that $x \in V_j$. Thus, we can let

$$G_i = \{y \in \text{gen}(V_i) : \|x - y\| = r_i\}$$

$$G_j = \{y \in \text{gen}(V_j) : \|x - y\| = r_j\}$$

Note that since x was chosen to be on an $(N-1)$-dimensional subset of $\partial V_i \cap \partial V_j$, it must be on the relative interior of $\partial V_i \cap \partial V_j$. Additionally, we must have $|G_i| = |G_j| = 1$ and $G_i \cap G_j = \emptyset$. This can be seen immediately from the recursive construction of the k-order Voronoi diagram. During its construction, all Voronoi faces in the k-order Voronoi diagram arise as bisecting hyperplanes (between two points). This is due to taking the intersection of a lower order Voronoi region with a first order Voronoi diagram, which itself consists of bisecting hyperplanes. Hence, all Voronoi faces in $V^k(\mathcal{X})$ arise as bisecting hyperplanes. Finally, since x lies on an $(N-1)$-dimensional subset of $\partial V_i \cap \partial V_j$, it cannot lie on the intersection of two or more bisecting hyperplanes. Thus, the sets G_i and G_j must consist of a single point each, in particular the points for which $\partial V_i \cap V_j$ act as a bisecting hyperplane.

For the system to exhibit attractive sliding motion at the point $x \in \partial V_i \cap \partial V_j$, we must have

$$\langle \eta, -\nabla E_i(x) \rangle < 0 \qquad \text{and} \qquad \langle \eta, -\nabla E_j(x) \rangle > 0$$

Since G_i and G_j each consist of a single element, we can let $G_i = \{y\}$ and $G_j = \{z\}$ with $y \neq z$. By definition, since every point other of $\text{gen}(V_i)$ and $\text{gen}(V_j)$ are closer to x than the points y and z are to x, we must have that the sets $\text{gen}(V_i)$ and $\text{gen}(V_j)$ intersect in every element except y and z. That is,

$$\text{gen}(V_i) \cap \text{gen}(V_j) = \{c_1, ..., c_{k-1}\}$$

where $c_i \in \mathcal{X} \backslash \{y, z\}$ for every $1 \leq i \leq k-1$. Now define

$$B = \frac{1}{k-1} \sum_{i=1}^{k-1} c_i$$

and notice that

$$\text{Bar}_G(V_i) = \left(\frac{k-1}{k}\right) B + \frac{1}{k} y$$

$$\text{Bar}_G(V_j) = \left(\frac{k-1}{k}\right) B + \frac{1}{k} z$$

That is, $\text{Bar}_G(V_i)$ and $\text{Bar}_G(V_j)$ are both convex combinations of B with y and z, respectively. Now by the assumption that Σ_{ij} exhibits sliding motion, we must have $\text{Bar}_G(V_i) \in H_j$ and $\text{Bar}_G(V_j) \in H_i$. We also know $y \in H_i$ since points of V_i are closer to y than they are to z. Then since $\text{Bar}_G(V_i) \in H_j$ is a convex combination of B and $y \in H_i$, we must have $B \in H_j$ following from the fact that H_i is also convex. However, we also have $z \in H_j$ and since $\text{Bar}_G(V_j) \in H_i$ is a convex combination of B and $z \in H_j$, we must similarly have $B \in H_i$. The only way this can occur is if $B \in H_i \cap H_j$. But then if $B \in H_i \cap H_j$, any convex combination of B with a point from $\text{int}(H_i)$ will fall in H_i and not in H_j. Thus we have come to a contradiction since we assumed $\text{Bar}_G(V_i) \in H_j$. Therefore, we cannot have attractive sliding motion. □

This is an important result for the numerical implementation of the k-nearest neighbor

flow. Suppose that this result did not hold and that the system did exhibit attractive sliding motion. Then once a trajectory encountered a sliding set exhibiting attractive sliding, we would need to calculate $\alpha(x)$. However, this requires we know the normal vector η which in turn requires that we have computed the k^{th}-order Voronoi diagram. This is extremely computationally expensive in high dimensions. Therefore, any accurate numerical implementation would have poor computational performance in high dimensions. As it turns out, attractive sliding does not occur and repulsive sliding is only relevant for points which start on the boundary of two k^{th}-order Voronoi regions since no other points will flow onto a switching set exhibiting respulsive sliding. Since the skeleton of the k^{th}-order Voronoi diagram has measure zero, these points will occur with probability zero when using data sampled with noise.

3.3.2 Periodic Orbits

For a smooth function $F : \mathbb{R}^N \to \mathbb{R}$, the gradient flow induced by F is given by

$$\frac{dx}{dt} = -\nabla F(x)$$

and it is a well known fact that this system does not have any periodic orbits. One can see this through simple computation. If such a periodic orbit did exist, then we would have $x(0) = x(T)$ for some $T > 0$ and so $F(x(0)) = F(x(T))$. Thus,

$$0 = F(x(T)) - F(x(0)) = \int_0^T \frac{dF(x)}{dt} dt = \int_0^T \nabla F \cdot \frac{dx}{dt} dt = -\int_0^T \left\| \frac{dx}{dt} \right\|^2 dt < 0$$

which is a contradiction and hence such an orbit cannot exist.

For our problem, while the function $E_{\mathcal{X}}$ is continuous, it is not a smooth function since it has a discontinuous gradient along the Voronoi faces. However, we can still establish the following.

Theorem 3.20. *The k-nearest neighbors flow defined by Equation 3.8 does not have any periodic orbits.*

Proof. Let $x \in \mathbb{R}^N$ and let $\phi(x, \cdot) : \mathbb{R} \to \mathbb{R}^N$ denote the solution to the gradient flow system of Equation 3.8 with initial point x. Suppose for sake of contradiction there exists $s > 0$ such that

$$\phi(x, s) = \phi(x, 0) = x$$

As we increase t from 0 to s, the trajectory $\phi(x, t)$ may intersect Voronoi faces of the k-order Voronoi diagram. Suppose the trajectory $\phi(x, \cdot)$ intersects m Voronoi faces, $F_1, ..., F_m$, of the k-order Voronoi diagram at times $0 < t_1 < ... < t_m < s$, i.e. we assume $\phi(x, t_i) \in F_i$ for each $i \in \{1, ..., m\}$. For simplicity, let $t_{m+1} = s$. Note that in the intervals $I_i = (t_i, t_{i+1})$, the function $E_{\mathcal{X}}(\cdot)$ is smooth. Thus, we have the following

$$\begin{aligned} 0 &= E_{\mathcal{X}}(x) - E_{\mathcal{X}}(x) = E_{\mathcal{X}}(\phi(x, s)) - E_{\mathcal{X}}(\phi(x, 0)) \\ &= \sum_{i=1}^{m} E_{\mathcal{X}}(\phi(x, t_{i+1})) - E_{\mathcal{X}}(\phi(x, t_i)) = \sum_{i=1}^{m} \int_{t_i}^{t_{i+1}} \frac{dE_{\mathcal{X}}}{dt}(\phi(x, t))dt \end{aligned} \tag{3.12}$$

However, letting $u(t) = \phi(x, t)$, we have

$$\frac{dE_{\mathcal{X}}}{dt}(\phi(x, t)) = \nabla E_{\mathcal{X}}(u(t)) \cdot \frac{\partial u}{\partial t}(t) = -\nabla E_{\mathcal{X}}(u(t)) \cdot \nabla E_{\mathcal{X}}(u(t)) = -\left\|\nabla E_{\mathcal{X}}(\, u(t))\right\|^2 < 0$$

for all $t \in I_i$ and each $i \in \{1, ..., m\}$. Thus, we see that the right hand side of Equation 3.12 must be strictly less than 0. This contradicts the assumption that $E_{\mathcal{X}}(\phi(x, s)) - E_{\mathcal{X}}(\phi(x, 0)) = 0$ and hence we cannot have $\phi(x, s) = \phi(x, 0)$. Therefore, there cannot be any periodic orbits arising from the gradient flow system defined by Equation 3.8. □

Theorem 3.20 shows the gradient flow system defined by Equation 3.8 contains no periodic orbits. Having this theoretical result in hand means it will come as no surprise that the

flow diagrams we produce later in Section 3.5 also exhibit no periodic orbits. This is also an important result given our stated goal of reducing noise in point clouds. If the gradient flow system of Equation 3.8 did exhibt periodic orbits, then we would have a difficult time determining optimal stopping times since the algorithm would not converge.

3.3.3 Stability of Flow in the Bottleneck Distance

Inducing a gradient flow on a point cloud moves the points around the ambient space and will therefore change the topological nature of any complexes constructed from the evolving point cloud. We would like to be able to track these changes. Of course, we can do this through the use of persistence diagrams. As the point cloud evolves, so too do the persistence diagrams. Stacking up the persistence diagrams yields a vineyard. Since the gradient flow induces a continuous motion of the points in the cloud, it seems natural that the persistence diagrams will flow continuosly as well.

We would like to apply Theorem 2.6 to prove the stability of the persistence diagrams. Of course, to apply this result to the distance function, we first need to know that the distance function is in fact tame. The following lemma results from the fact that $\mathcal{X}$ is a finite point cloud. It is established in [42] by Elizabeth Munch.

Lemma 3.21. *The distance function $d_{\mathcal{X}}$ is tame for any finite point cloud $\mathcal{X} \subset \mathbb{R}^N$.*

Thus, we see that the persistence diagrams generated by two distance functions are close if the distance functions themselves are close, in the L^∞ sense. Putting these results together, we can prove a new result establishing the stability of the persistence diagrams generated by the distance function during the point cloud flow. The following theorem follows immediately from Theorem 2.6.

Theorem 3.22. *Let $E(u)$ denote the k-nearest neighbor function. Define $\phi(x,t)$ as in Equation 3.8. Let $\mathcal{X}_t = \{\phi(x,t) : x \in \mathcal{X}\}$ for all $t \geq 0$ and let $\mathrm{Dgm}_p(\mathcal{X}_t)$ denote the*

p-dimensional persistence diagram induced by the distance function $d_{\mathcal{X}_t}$. Finally, let

$$K_t^s = \max_{x \in \mathcal{X}} \|\phi(x,t) - \phi(x,s)\|$$

Then for any $t < s$, we have

$$d_B(\mathrm{Dgm}_p(\mathcal{X}_t), \mathrm{Dgm}_p(\mathcal{X}_s)) \leq K_t^s$$

Proof. First note that both distance functions $d_{\mathcal{X}_t}$ and $d_{\mathcal{X}_s}$ are tame by Lemma 3.21. Thus, they both satisfy Theorem 2.6 and so

$$d_B(\mathrm{Dgm}_p(\mathcal{X}_t), \mathrm{Dgm}_p(\mathcal{X}_s)) \leq \|d_{\mathcal{X}_t} - d_{\mathcal{X}_s}\|_\infty$$

Therefore, we only need to prove that $\|d_{\mathcal{X}_t} - d_{\mathcal{X}_s}\|_\infty \leq K_t^s$. To see this, fix a point $u \in \mathbb{R}^N$ and let

$$x_1 = \arg\min_{x \in \mathcal{X}} \|\phi(x,t) - u\|$$

Similarly, let

$$x_2 = \arg\min_{x \in \mathcal{X}} \|\phi(x,s) - u\|$$

Then we know $\|\phi(x_1,t) - u\| \leq \|\phi(x_2,t) - u\|$ and so we have

$$d_{\mathcal{X}_t}(u) - d_{\mathcal{X}_s}(u) = \|\phi(x_1,t) - u\| - \|\phi(x_2,s) - u\| \leq \|\phi(x_2,t) - u\| - \|\phi(x_2,s) - u\|$$

Similarly, we know $\|\phi(x_2,s) - u\| \leq \|\phi(x_2,t) - u\|$. Hence, we have

$$d_{\mathcal{X}_s}(u) - d_{\mathcal{X}_t}(u) = \|\phi(x_2,s) - u\| - \|\phi(x_1,t) - u\| \leq \|\phi(x_1,s) - u\| - \|\phi(x_1,t) - u\|$$

Now suppose $d_{\mathcal{X}_t}(u) \geq d_{\mathcal{X}_s}(u)$. Then we have

$$|d_{\mathcal{X}_t}(u) - d_{\mathcal{X}_s}(u)| \leq \Big| \|\phi(x_2, t) - u\| - \|\phi(x_2, s) - u\| \Big| \leq \|\phi(x_2, t) - \phi(x_2, s)\|$$

by the reverse triangle inequality. On the other hand, if $d_{\mathcal{X}_t}(u) \leq d_{\mathcal{X}_s}(u)$, then we have

$$|d_{\mathcal{X}_s}(u) - d_{\mathcal{X}_s}(u)| \leq \Big| \|\phi(x_1, t) - u\| - \|\phi(x_1, s) - u\| \Big| \leq \|\phi(x_1, t) - \phi(x_1, s)\|$$

again by the reverse triangle inequality. Therefore, if we set

$$M(x) = \max\{\|\phi(x_1, t) - \phi(x_1, s)\|, \|\phi(x_2, t) - \phi(x_2, s)\|\}$$

we have

$$|d_{\mathcal{X}_t}(u) - d_{\mathcal{X}_s}(u)| \leq M(x)$$

But of course $M(x) \leq K_t^s$. Therefore, we have

$$|d_{\mathcal{X}_t}(u) - d_{\mathcal{X}_s}(u)| \leq K_t^s$$

Since this holds for every $u \in \mathbb{R}^N$, we must have

$$\|d_{\mathcal{X}_t} - d_{\mathcal{X}_s}\| \leq K_t^s$$

Hence, by the Stability Theorem, we have the desired result, i.e.

$$d_B(\mathrm{Dgm}_p(\mathcal{X}_t), \mathrm{Dgm}_p(\mathcal{X}_s)) \leq \|d_{\mathcal{X}_t} - d_{\mathcal{X}_s}\|_\infty \leq K_t^s$$

□

This proof tells us that as we evolve a point cloud following the k-nearest neighbor

gradient flow, the peristence diagrams associated with $\mathcal{X}_t$ at each time $t > 0$ will evolve continuously. Thus, the vineyard of the point clouds $\mathcal{X}_t$ will evolve continuously.

3.4 Diffusive Flow

One issue that arises from the gradient system described by Equation 3.8 is that points will tend to cluster together under the flow. This happens when points in $\mathcal{X}$ flow toward the same barycenter. One obvious way to lessen this effect is to terminate the flow before the points get too clustered together. Another approach is to add some diffusion to the system. In particular, we would like to push points away from each other when they get too near. We can do this by adding a diffusive term to our gradient system. That is, we modify the system to become

$$\begin{cases} \dfrac{du}{dt} = -\nabla E_{\mathcal{X}_0}(u) + \lambda \nabla E_{\mathcal{X}_t}(u) & t > 0 \\ u(0) = x & t = 0 \end{cases} \tag{3.13}$$

where $\lambda \in [0, 1]$ and

$$\phi_\lambda(x, t) = u(t)$$

where u is the solution to Equation 3.13 with initial condition x. We can then set

$$\mathcal{X}_t = \{\phi_\lambda(x, t) : x \in \mathcal{X}\}$$

The trick here rests in computing the second gradient (i.e. $\nabla E_{\mathcal{X}_t}(u)$) using the current nearest neighbors (i.e. $\mathrm{NN}^k_{\mathcal{X}_t}(u)$, using points from $\mathcal{X}_t$) instead of the original neighbors from $\mathcal{X}_0$. As the points approach one another and begin clustering, the distance-to-measure function induced by the empirical measure on the evolved point cloud $\mathcal{X}_t$, that is $\mu_{\mathcal{X}_t}$, will increase. Thus, by introducing the gradient of $E_{\mathcal{X}_t}$ with opposite sign of the first gradient term, points will push against their nearests neighbors. The repulsivity parameter λ is used to control how much repulsion is exhibited by the points.

Although this does prevent the points from clustering, there is a more sophisticated, efficient, and effective method, presented in the next chapter, which is based on the surface reconstruction work presented in Section 1.4. In this technique, we will approximate the normal bundle of the underlying manifold. In the process, we also approximate the tangent bundle. While we use the normal bundle to reduce the noise and perform the smoothing, we can imagine applying this diffusive flow in the tangential directions obtained via the tangent bundle approximation. This would cause points to spread out along the surface of the inferred manifold. Often, for example with 3d line scanners, this behavior can be strongly desired.

3.5 Flow Diagram

From the analysis conducted so far, we have been able to say quite a lot about the gradient flow system introduced in Equation 3.8. We have determined the trajectories within Voronoi regions, found some positively invariant sets, discovered all the fixed points, and shown there are no periodic orbits. Using this information, we would like to encapsulate all of these dynamics into a single, finite structure. The result will be a directed graph upon which we can perform traditional graph analytics. To accomplish this, we will introduce the notion of the k-order Delaunay triangulation $D^k(\mathcal{X})$ of a set $\mathcal{X}$. We will then find a subgraph of $D^k(\mathcal{X})$ and orient the edges so that the k-nearest neighbor gradient flow is qualitatively described. All constructions and results concerning the flow diagram are novel, as applied to this gradient flow system. First, we start with a well known generalization of the Delaunay triangulation.

Definition 3.23. Let $V^k(\mathcal{X})$ be the k-order Voronoi diagram of a finite point cloud $\mathcal{X} \subset \mathbb{R}^N$. We define the ***k*-order Delaunay triangulation** $D^k(\mathcal{X})$ to be the graph whose vertices correspond to Voronoi regions in $V^k(\mathcal{X})$, and where an edge exists between two regions $V_1, V_2 \in V^k(\mathcal{X})$ whenever $\overline{V}_1 \cap \overline{V}_2 \neq \emptyset$.

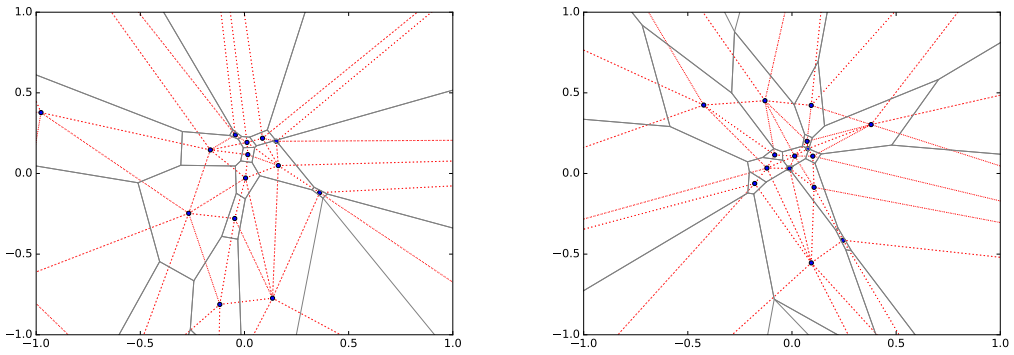
Therefore, the k^{th}-order Delaunay triangulation is the dual graph of the k^{th}-order

Voronoi diagram. It is important to note that this definition of the k^{th}-order Delaunay triangulation differs from the definition sometimes given in the literature, for example by Gudmundsson et al in [30]. In that work, the authors define the higher order Delaunay triangulations to be a class of triangulations. In particular, they relax the notion of the Delaunay triangulation by requiring that for each triangle in the triangulation, the circle passing through the vertices of the triangle contains at most k points of the generating point cloud $\mathcal{X}$. Therefore, there is an entire class of higher order Delaunay triangulations of any given order. In contrast, our definition extends naturally from the duality of the first order Voronoi diagram and the traditional Delaunay triangulation.

Example 3.3. We now expand upon Example 3.2 where we computed the third and fifth order Voronoi diagrams of a point cloud $\mathcal{X}$ sampled randomly from $[0,1]^2$. In particular, we compute the third and fifth order Delaunay triangulations of $\mathcal{X}$. We show the result in Figure 3.7. Here, we still show the Voronoi diagram using black lines, but also show edges in the higher order Delaunay triangulations as red, dotted lines. Instead of showing the original points of $\mathcal{X}$, we are now showing the vertices of the Delaunay triangulations. For this, we take the vertex of a region $V \in V^k(\mathcal{X})$ to be the generator barycenter if $\text{Bar}_G(V) \in V$ and we use the polytope barycenter, $\text{Bar}_P(V)$, otherwise. It is important to note that the bounds of the image were chosen to highlight the behavior near the sampled region $[0,1]^2$. Many of these higher order Voronoi regions are in fact unbounded.

As we noted earlier, if a Voronoi region V satisfies the barycentric sink condition, that is if $\text{Bar}_G(V) \in V$, then any trajectory begining in V will forever remain in V. However, if a region does not satisfy this condition, all trajectories will eventually exit V. Although every trajectory in V will flow towards the generator barycenter of V, $\text{Bar}_G(V)$, the trajectories may not move to the same region once departing V. To determine where the trajectories go upon exiting the region V, we need to concept of a polyhedral pyramid.

Definition 3.24. Given a point $x \in \mathbb{R}^N$ and an N-dimensional polytope σ, we define the **polyhedral pyramid**, $C_\sigma(x)$, from x to σ to be the union of all line segments from x to



(a) $k = 3$ (b) $k = 5$

Figure 3.7: k-order Delaunay triangulation for samples drawn from $[0, 1]^2$

points in σ. That is,

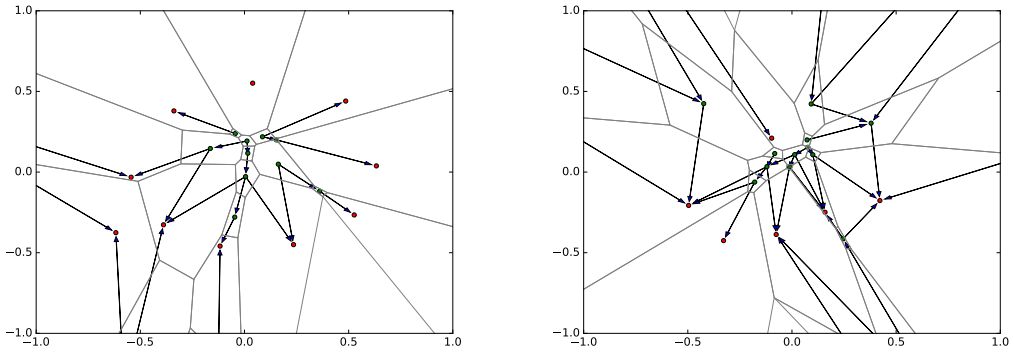
$$C_\sigma(x) = \{\lambda y + (1 - \lambda)x : \lambda \in [0, 1], y \in \sigma\}$$

Since the k^{th}-order Delaunay triangulation connects all adjacent k^{th}-order Voronoi regions, we would like to subset this graph and only maintain edges upon which trajectories will actually flow. Furthermore, we would like to orient the edges to respect the directionality of this flow. For any region $V \in V^k(\mathcal{X})$, either we have $\text{Bar}_G(V) \in V$ or $\text{Bar}_G(V) \notin V$. In the former case, Proposition 3.16 tells us we should not have any flow lines leaving V since all trajectories are trapped within V. On the other hand, if $\text{Bar}_G(V) \notin V$, then as shown in the proof of Proposition 3.16, any point $x \in \text{int}(V)$ will flow towards $\text{Bar}_G(V)$ under the gradient flow for some positive time $R(x)$. The point will follow this trajectory until it reaches a boundary and enters a new region. Therefore, for Voronoi regions V with $\text{Bar}_G(V) \notin V$, we must determine into which adjacent regions the points in V will flow. We can determine this information using the polyhedral pyramid from $\text{Bar}_G(V)$ to V, that is $C_V(\text{Bar}_G(V))$.

Definition 3.25. The ***k*-nearest neighbor flow diagram**, $F^k(\mathcal{X})$, is a directed graph

with vertices given by regions in $V^k(\mathcal{X})$ and edges defined as follows: For any edge (V, W) in $D^k(\mathcal{X})$ where $V, W \in V^k(\mathcal{X})$ and $\text{Bar}_G(V) \notin V$, we include a directed edge from V to W in the graph $F^k(\mathcal{X})$ if $W \cap C_V(\text{Bar}_G(V)) \neq \emptyset$.

From this directed multigraph, what we refer to as a flow diagram, it is possible to capture much of the dynamics of the gradient flow system defined in Equation 3.8. Given any $x \in \mathbb{R}^N$, one finds the corresponding Voronoi region containing x, and then follows the flow lines in $F^k(\mathcal{X})$ to the final sink at which point the trajectory falls forever toward that sink. Of course, any k^{th}-order Voronoi region may contain multiple flow lines eminating from the region. Therefore, the flow lines constitute a multivalued map. Once we have a finite graph structure, we can perform a variety of graph based analytics to further investigate the system. For example, we can compute the number of cycles, which we know will be zero based on the results of Theorem 3.20. We can also compute the number of connected components and determine the possible regions of attraction.



(a) $k = 3$ (b) $k = 5$

Figure 3.8: k-order flow diagram for samples drawn from $[0, 1]^2$

Example 3.4. Once again, we continue the example given in Examples 3.2 and 3.3. In this example, we show the flow diagram for the point cloud $\mathcal{X}$ sampled from $[0, 1]^2$. The results are shown in Figure 3.8. In this figure, the vertices of the Delaunay triangulation are shown in either red or green, with the red vertices corresponding to sinks. The arrows

indicate the direction of flow from green vertices to either green or red vertices. Note that no flow lines emanate from the red vertices (sinks). It is important to note that the bounds of the image were chosen to highlight the behavior near the sampled region $[0, 1]^2$ and that some flow lines are not visible because of this restriction.

Chapter 4: Extensions

In this chapter, we present several novel modifications to the distance-to-measure gradient flow. These modifications are motivated by issues that are often encounted when smoothing using the system described in Chapter 3. To begin, we discuss the *normal bundle flow* in Section 4.1. This technique involves approximating the normal bundle of the manifold from which a point cloud was sampled. Then, when smoothing the point cloud, the k-nearest neighbors gradient vector is projected into the approximated normal space. This has the effect of reducing the amount of clustering present in the evolving point clouds.

In the process of approximating the normal space around a point, we are also approximating the tangent space. This has the benefit of allowing us to induce diffusion of the point cloud along the tangent space, further reducing clustering of the points in the point cloud. This modification is also useful for point clouds obtained from line scanners, which scan an object in lines. The process leaves visible streaks where points were sampled, with regions void of any samples in between the scanned line. We present this diffusive technique in Section 4.2.

One of the assumptions we have made throughout our discussion thus far is that a noisy point cloud requires the same level of smoothing throughout the point cloud. Of course, for many real world examples, this assumption does not hold. Thus, we require a method for adapting the gradient flow based on the local geometry of the point cloud. To this end, in Section 4.3, we design a novel technique for adding adaptivity to the k-nearest neighbors gradient flow. This method relies on approximating the local curvature around a point and adjusting the value of k to reflect the degree of curvature. In particular, for high curvature regions, we lower the value of k in an effort to preserve the high curvature feature. This technique builds on our normal and tangent bundle approximations.

Although the normal bundle flow arose from the gradient of the distance-to-measure

function, it actually shares an interesting relationship with another gradient flow, this time induced by the Mahalanobis distance. The Mahalanobis distance is a function often used in statistics to measure the distance between a point and a probability distribution. Since the distance-to-measure function also provides a method for describing the distance between a point and a probability distribution, it comes as no surprise that these two functions induce related gradient flows. In Section 4.4, we discuss this relationship and explicitly show how these two flows differ.

Finally, we close the chapter with a short discussion on the method we employ for finding the k-nearest neighbors in an efficient manner. Since the nearest neighbor computation dominates the computational complexity of our algorithms, it is important that we employ a fast and scalable technique for computing the nearest neighbors. In Section 4.5, we discuss our use of the $k-d$ tree data structure and suggest other approaches which would provide additional speed increases, at the cost of accuracy.

4.1 Normal Bundle Flow

As we saw in the previous chapter, while the k-nearest neighbor flow described by Equation 3.8 evolves a point cloud toward the underlying sampled manifold according to the distance-to-measure function, the flow tends to cause many points to cluster together. To alleviate these concerns, we discussed adding a diffusive term to the gradient flow in Section 3.4. Another approach which avoids point clustering is to approximate the normal bundle of the underlying manifold and then project the k-nearest neighbor flow onto this normal bundle. Therefore, we only allow points to move toward the manifold in the direction of the approximated normals.

The work in this section is based on the ideas presented in Section 1.4. However, in the reference paper for that section (i.e. [4]), the authors only discuss approximating normal vectors and tangent planes for a two-dimensional manifold in $\mathbb{R}^3$. Here, we generalize their approach to arbitrary dimensions and arbitrary dimensional manifolds. Thus, instead of approximating a normal direction and a tangent plane, we approximate the normal

space and tangent space. Since we compute this approximation at every point of the point cloud, we are effectively approximating the normal and tangent bundles. In this section, we incorporate the normal bundle approximation method with the gradient flow system. This provides a novel improvement to the existing gradient flow.

Thus far, we have not required many assumptions concerning the underlying probability distribution from which we sample the point cloud $\mathcal{X}$. However, in this chapter, we will require that the underlying distribution is supported on a manifold. This requirement comes about due to the additional structure implied by the normal and tangent bundles. On the other hand, no such structure was required for the k-nearest neighbor flow of Chapter 3. To be precise, we assume a uniform measure ν_M whose support is on some d-dimensional compact manifold M. We then convolve this measure with a noise measure ζ, often taken to be an N dimensional Gaussian distribution. Recall that we defined this sampling procedure more rigorously in Section 1.2. Under this convolution assumption, one can consider implementing a deconvolution procedure, as the authors did in [10], to reduce the influence of the noise distribution ζ. Here, we will instead use the point cloud sampled directly from the convolved measure and attempt to reduce the noise present in the point cloud through a modified version of the k-nearest neighbor flow.

Since we are assuming an underlying manifold structure, we can utilize the concept of the normal and tangent bundles of the manifold. Consider a point $x \in M$ sampled directly on the manifold (i.e. with no noise). If we were to move this point, a perturbation in any of the tangential directions at x would cause the distance between the point and the manifold to only increase slightly, if at all (the point may simply be relocated to another position on the manifold). However, if we perturb the sample in a normal direction, then we introduce a great deal of noise compared to the tangential perturbation. Therefore, our primary concern will be reducing the noise present in the normal directions. Moving points along the tangent bundle basically relocates the points along the manifold and adjusts the sampling density on the manifold, whereas moving points in the normal bundle drastically impacts the overall noise in the system. We use this insight as motivation for what follows.

First, we recall the following definitions of the tangent and normal bundles of a manifold from Guillemin and Pollack in [31].

Definition 4.1. Let $X \subset \mathbb{R}^N$ be a d-dimensional manifold and let $\phi : U \to X$ be a local parameterization around $x \in X$, where $U \subset \mathbb{R}^d$ is open. For simplicity, suppose $\phi(0) = x$. Then the map

$$u \to \phi(0) + d\phi_0(u) = x + d\phi_0(u)$$

is the best linear approximation to ϕ at 0. We define the **tangent space**, $T_x(X)$, of X at $x \in X$ to be the image of $d\phi_0 : \mathbb{R}^d \to \mathbb{R}^N$. That is,

$$T_x(X) = \operatorname{im} d\phi_0$$

Then the **tangent bundle**, TX, of the manifold X is given by

$$TX = \{(x, v) : x \in X, v \in T_x(X)\}$$

Similarly, we define the **normal space**, $N_x(X)$, of X at $x \in X$ to be the vectors in $\mathbb{R}^{N-d}$ which are orthogonal to $T_x(X)$. That is

$$N_x(X) = \{v \in \mathbb{R}^{N-d} : (v, w) = 0, \forall w \in T_x(X)\}$$

and the **normal bundle**, NX, of the manifold X is given by

$$NX = \{(x, v) : x \in X, v \in N_x(X)\}$$

Note that while the tangent bundle of a manifold is an intrinsic object associated with the manifold, the normal bundle depends on the embedding of the manifold. In particular, the dimension of the normal bundle depends on the dimensionality of the ambient space $\mathbb{R}^N$. However, since we are considering a fixed ambient space, this dependence does not pose any

complications. Additionally, it is important to note that while the definition of the tangent space uses a particular choice of parameterization ϕ_0, any other local parameterization will produce the same tangent space as shown in [31].

The normal bundle of a manifold carries with it a great deal of geometric information about the manifold. Therefore, approximating this normal bundle is a natural way of approximating the geometry of the manifold. To this end, we construct a vector space for each point x in our point cloud $\mathcal{X}$, which approximates the normal space at the point $P_M(x)$, where $P_M(x)$ is the projection of x onto the manifold M. That is,

$$P_M(x) = \arg\min_{y \in M} \|x - y\|$$

In doing so, we will also be defining the tangent space at $P_M(x)$ and therefore we will be approximating a local frame for the manifold.

In particular, we follow the normal bundle approximation procedure outlined in [46], modified for our focus on arbitrary dimensions. For simplicity, we will outline the procedure assuming a codimension 1 manifold. The extension to higher codimensional manifolds will be made explicit, but is quite simple. Since we are only considering codimension one manifolds at the moment, for a point $u \in \mathbb{R}^N$, we will be seeking a single normal vector $n \in \mathbb{R}^N$, which defines the tangent plane $T(u)$ near u. Here, we use the Hesse normal form of a hyperplane. That is, the tangent plane is defined as

$$T : y_i \cdot n - d = 0$$

where d is the distance from the origin, 0, to the plane T and y_i are a set of k points which lie on the plane T. From this definition of a hyperplane, one can write an equation for a hyperplane containing the points y_i. In our case, we would like to find a hyperplane containing the points $p_i \in \mathrm{NN}^k_{\mathcal{X}}(u)$, however we cannot hope these points all lie on a

hyperplane. Therefore, we will need to approximate a hyperplane which best fits the k-nearest neighbor points. Since finding a hyperplane containing all the points is unlikely, we instead seek a plane T which minimizes the following sum of weighted squares,

$$\sum_{p \in \mathrm{NN}^k_{\mathcal{X}}(u)} (p \cdot n - d)^2 \phi_p(u)$$

such that $\|n\| = 1$ and the weights $\phi_p(u)$ are defined as $\phi_p(u) = \phi(\|p - u\|, u)$ where

$$\phi(\ell, u) = e^{-\ell^2/(r(u)^2 \sigma^2)} \tag{4.1}$$

for $r(u) = \max_{p \in \mathrm{NN}^k_{\mathcal{X}}(u)} \|p - u\|$ and a noise parameter $\sigma \in \mathbb{R}$ which controls the influence of the neighbors of u. We use weights in the above formulation to give greater influence to neighbors close to x than those distant from x. This allows local features to have greater influence, helping to preserve them. Additionally, since the value of $r(u)$ depends on the size of the minimum enclosing sphere of $\mathrm{NN}^k_{\mathcal{X}}(u)$, the weights naturally adapt to the local sampling density. Through the use of Lagrange multipliers, the above minimization problem is found to be equivalent to the following eigenvector problem,

$$C(u) \cdot v = \lambda v \tag{4.2}$$

where the matrix $C(u)$ is chosen to be the sample covariance matrix whose form is given by

$$C(u) = \frac{1}{N-1} \begin{bmatrix} p_1 - \tilde{p}(u) \\ \vdots \\ p_k - \tilde{p}(u) \end{bmatrix}^T \begin{bmatrix} p_1 - \tilde{p}(u) \\ \vdots \\ p_k - \tilde{p}(u) \end{bmatrix} \tag{4.3}$$

and we define $\tilde{p}(u)$ using the k-nearest neighbor set $\mathrm{NN}^k_{\mathcal{X}}(u) = \{p_1, p_2, ..., p_k\}$ so that

$$\tilde{p}(u) = \frac{1}{S} \sum_{p \in \mathrm{NN}^k_{\mathcal{X}}(u)} p\phi_p(u) \tag{4.4}$$

where

$$S(u) = \sum_{p \in \mathrm{NN}^k_{\mathcal{X}}(u)} \phi_p(u)$$

This is the sample covariance matrix of the set $\mathrm{NN}^k_{\mathcal{X}}(u)$ where the expected value is given by $\tilde{p}(u)$. Thus, considering the points of $\mathrm{NN}^k_{\mathcal{X}}(u)$ probabalistically, we have assigned probabilities of $\phi_p(u)$ to each point $p \in \mathrm{NN}^k_{\mathcal{X}}(u)$. Alternatively, we could have set $\tilde{p}(u) = \mathrm{Bar}_G(V)$ where $u \in V$.

Finding the eigenvectors of Equation 4.2 will allow us to approximate the normal and tangent spaces at the point $u \in \mathbb{R}^N$ in a manner similar to that of [46]. Since $A^T A$ is positive-semidefinite for any matrix A with real entries, we know C is positive-semidefinite as well. Furthermore, since the matrix $C(u)$ is real and symmetric, it will have real, orthogonal eigenvectors. Finally, since it is derived from noisy data, $C(u)$ will be a full rank matrix in general. Therefore, the eigenvectors can be used to approximate a coordinate frame around u.

The minimizing normal is then found to be the eigenvector v_i with smallest associated eigenvalue λ_i. We can easily generalize this situation for arbitrary codimension m manifolds (i.e. the dimension of the manifold is $d = N - m$) by taking the smallest m such eigenvectors. Suppose we order the eigenvectors such that $v_1 \leq v_2 \leq ... \leq v_N$ where the ordering is induced by the associated eigenvalues, i.e. $0 \leq \lambda_1 \leq \lambda_2 \leq .. \leq \lambda_N$ (where we know the eigenvalues are all non-negative since C is positive-semidefinite). Then the first m such eigenvectors give us the local coordinate frame for the normal space and the remaining $N - m$ eigenvectors span the approximated tangent space. When taking eigenvectors, we will always use unit length eigenvectors. We denote the normal and tangent vector

approximations by $N(u)$ and $T(u)$, respectively. That is,

$$\begin{aligned} \tilde{N}(u) &= \{v_1, ..., v_m\} \\ \tilde{T}(u) &= \{v_{m+1}, ..., v_N\} \end{aligned} \tag{4.5}$$

We can now induce a gradient flow along the approximated normal directions. To accomplish this, we define the new gradient flow to be

$$\begin{cases} \dfrac{du}{dt} = -g_N(u) & t > 0 \\ u(0) = x & t = 0 \end{cases} \tag{4.6}$$

where, letting $E(u)$ denote the k-nearest neighbor energy function from Equation 3.2, we define

$$g_N(u) = \sum_{i=1}^{m} \langle \nabla E(u), v_i \rangle v_i \tag{4.7}$$

By restricting the flow to the approximated normal directions, we aim to avoid the clustering we saw under the unrestricted gradient flow. When $\tilde{p}(u)$ is defined as in Equation 4.4, we call the gradient flow in Equation 4.6 the **normal bundle flow**. If we choose $\tilde{p}(u) = \text{Bar}_G(V)$ for $u \in V$, then we call the gradient flow the **unweighted normal bundle flow**.

In Figure 4.1 we illustrate how the normal bundle flow works for a one dimensional curve in $\mathbb{R}^2$. In this figure, the curve is shown as the thick black line and the point cloud is shown by points that are both filled and unfilled. The focal point of this figure is the point u, which has two emanating arrows. The unfilled points are the nearest neighbors of the point u. In this example, we take $k = 5$. The dashed arrow emanating from u is the negative distance-to-measure gradient. The solid arrow on the other hand is the negative of the projected normal bundle gradient, i.e. $-g_N(u)$. Notice that $-g_N(u)$ is perpedicular

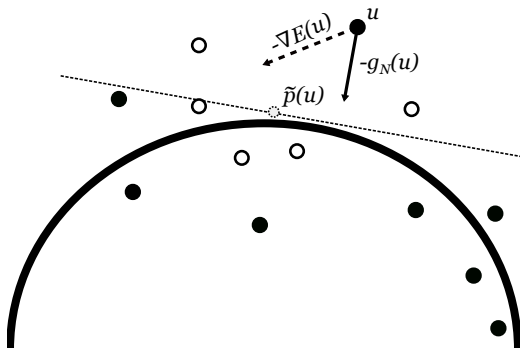


Figure 4.1: Normal Bundle Flow

to the approximated tangent plane, shown here as the long dashed line. The point with the dashed boundary represents the weighted barycenter $\tilde{p}(u)$.

Using the normal bundle modification of the k-nearest neighbor flow provides some robustness against the clustering behavior produced by the unmodified flow. As we saw, including this projection into the approximated normal directions requires one to compute a sample covariance matrix $C(u)$ as well as compute the eigenvalues and eigenvectors of this matrix. Thus, the normal bundle flow will be slower than the k-nearest neighbor flow. However, when we compare the k-nearest neighbor flow and the normal bundle flow in Chapter 5, we will see that the normal bundle flow outperforms the k-nearest neighbor flow in both a geometric and a topological sense. Futhermore, the normal bundle flow actually requires a lower value of k than the k-nearest neighbor flow for some interesting point clouds. Since the value of k plays a part in determining the computational expense of either algorithm, it is possible that the additional matrix computations inherent to the normal bundle flow are balanced out by using a lower value of k. The geometric and topological error reduction potential of these two algorithms will be further explored in Chapter 5.

4.2 Diffusive Flow

In this section, we build on the work in Section 3.4, where we explored adding a diffusive term to the k-nearest neighbor gradient flow with the goal of alleviating the clustering present in the original flow. In the previous section, we developed the machinery necessary to project the gradient $\nabla E_{\mathcal{X}}(u)$ into the approximated normal space around a point $u \in \mathbb{R}^N$. In the process of constructing the approximated normal space, we computed a basis for the approximated tangent space as well. This was given by $\tilde{T}(u)$ as defined in Equation 4.5. Previously, in Section 3.4, we hinted at restricting the diffusive term of Section 3.4 to the approximated tangent space of u. We can now make this precise.

For starters, let us recall the equation for the k-nearest neighbors gradient flow with the diffusive term added. This was given in Equation 3.13 and reproduced below.

$$\begin{cases} \dfrac{du}{dt} = -\nabla E_{\mathcal{X}_0}(u) + \lambda \nabla E_{\mathcal{X}_t}(u) & t > 0 \\ u(0) = x & t = 0 \end{cases}$$

Recall that λ was chosen between 0 and 1. This parameter controls the level of repulsivity points exhert. Furthermore, since the second gradient $\nabla E_{\mathcal{X}_t}(u)$ is taken on the set $\mathcal{X}_t$, the current position of the points is taken into account. That is, we compute nearest neighbors using $\mathrm{NN}^k_{\mathcal{X}_t}(u)$ instead of $\mathrm{NN}^k_{\mathcal{X}_0}(u)$. Thus, when points flow toward one another, they begin pushing on each other. The problem with this approach is that points can be pushed in directions normal to the sampled manifold, thus introducing more noise. We would therefore like to project this diffusive gradient into the approximate tangent space.

To do this, we follow the same procedure we used above when projecting the original gradient $-\nabla E_{\mathcal{X}_0}(u)$ into the normal space. In particular, we set

$$g_T(u) = \sum_{i=m+1}^{N} \langle \nabla E_{\mathcal{X}_t}, v_i \rangle v_i \tag{4.8}$$

where $\tilde{T}(u) = \{v_{m+1}, ..., v_N\}$ is as previously defined in Equation 4.5. Remember that the vectors in $\tilde{T}(u)$ are normalized so the projection formula in Equation 4.8 takes a simple form. We now induce a gradient flow system as follows.

$$\begin{cases} \dfrac{du}{dt} = -g_N(u) + \lambda g_T(u) & t > 0 \\ \\ u(0) = x & t = 0 \end{cases} \tag{4.9}$$

Once again, we set $\lambda \in [0, 1]$. Notice that we used the gradient $g_N(u)$ instead of $\nabla E_{\mathcal{X}_0}(u)$. Of course, we could have used the unprojected gradient, however as we will see in Chapter 5, the normal bundle flow outperforms the unprojected flow and so we will stick with $g_N(u)$. We will refer to this gradient flow as the **normal bundle flow with diffusivity**.

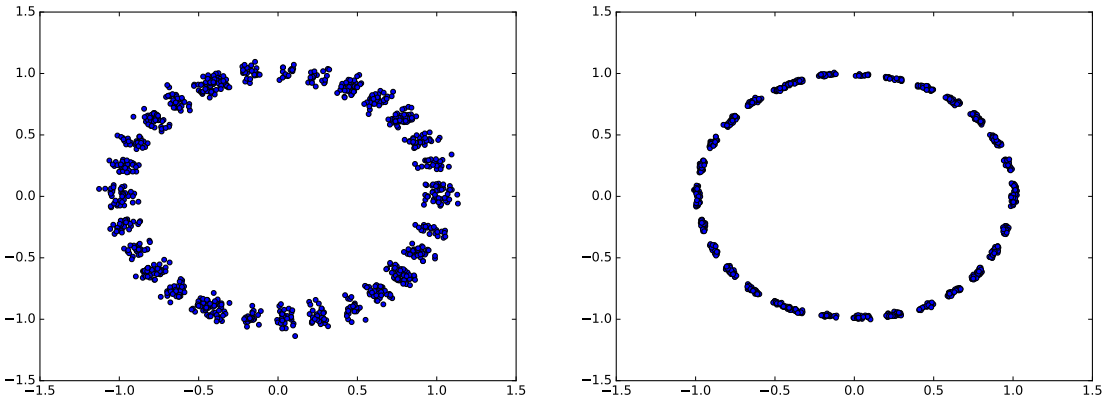
This modification can be useful for point clouds which were obtained using a line scanner. Since these scanners take readings in lines, these point clouds often exhibit streaks of sampled points with no points in between the streaks. This is simply an artifact of the sampling procedure and is not a true feature of the sampled surface. Thus, by inducing diffusive smoothing, one can fill out these empty regions of the surface by moving points sampled along the dense streaks into these barren regions. This is likely to improve topological reconstructions since simplicial complexes built on point clouds smoothed by the normal bundle flow with diffusivity would be able to connect across the unscanned regions more easily.

In practice, it makes sense to ramp up the diffusion as t increases. Thus, if we run the gradient flow until $T = t$, we use

$$\lambda(t) = \frac{t}{T}\lambda$$

as the diffusivity coeffient at time t. This has the effect of first smoothing the points, attempting to lower the distance-to-measure in the normal direction. Then, once the points are closer to the sampled manifold, letting the points push harder on one another and diffuse

across the surface of the manifold.



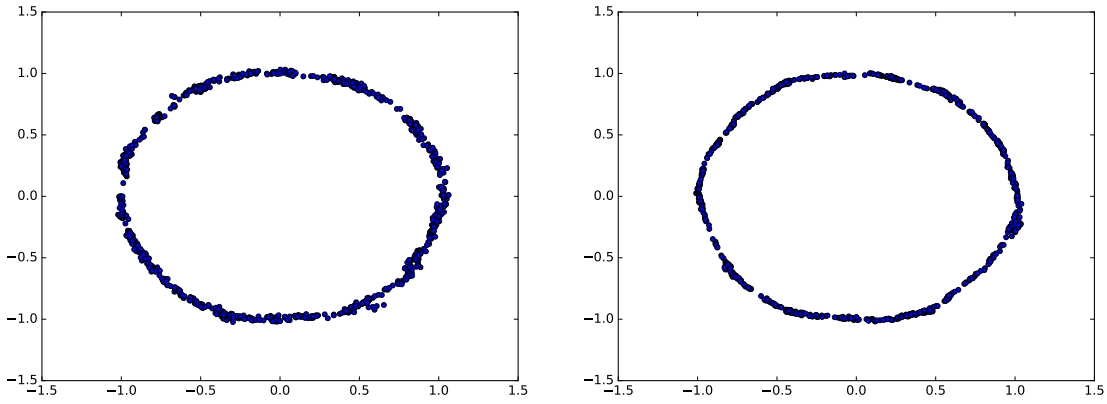
(a) Original Point Cloud (b) Normal Bundle Smoothing

Figure 4.2: Smoothing without diffusion

To demonstrate the use of the diffusive term, we show a point cloud in Figure 4.2 which was sampled from the unit circle S^1 in a manner similar to a line scanner. However, this example is in two dimensions to aid with visualization. Here, we see that only certain *bands* were scanned, resulting in strips void of any samples. As seen in Figure 4.2b, the normal bundle flow leaves these gaps in tact.

On the other hand, adding diffusivity to the gradient flow helps close the gaps by allowing flow in the tangential directions. The results of running the normal bundle flow with diffusivity are shown in Figure 4.3. The only difference between the smoothing performed in Figure 4.2 and Figure 4.3 was the addition of the diffusive term. It is clear from Figure 4.3a that the gaps have mostly disappeared. As a final step, we applied a single iteration of the normal bundle smoothing to the point cloud in Figure 4.3a. This has the effect of smoothing out some of the roughness that is still left after the initial diffusive flow. The results of this second pass are shown in Figure 4.3b.

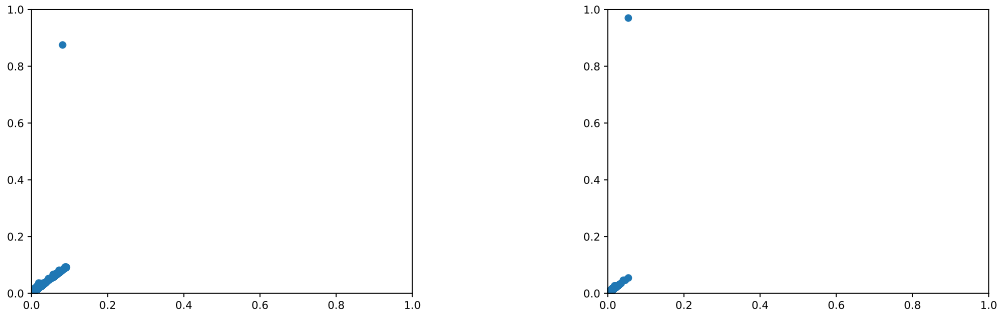
To get a sense of the difference between the two smoothed point clouds, we show the one-dimensional persistence diagrams associated with the point clouds smoothed with and



(a) Diffusion

(b) Diffusion with Second Pass

Figure 4.3: Smoothing with diffusion



(a) No Diffusion

(b) Diffusion

Figure 4.4: Persistence diagram of smoothed point clouds

without diffusion. These can be seen in Figure 4.4, where the left hand figure is the persistence diagram of the point cloud smoothed without diffusion (Figure 4.2b) and the right hand side is the point cloud with diffusion (Figure 4.3a). Notice that the most persistent component in the diffusion case is closer to $(0, 1)$ than the most persistent component in the non-diffusive point cloud. Since the circle contains a single hole, born at radius 0 and dying at radius 1, the optimal persistence diagram would contain a single point at $(0, 1)$. Thus, the fact that the persistence diagram of the point cloud obtained through the diffusive flow is closer to the optimal diagram than is the non-diffusive point cloud, indicates that the diffusive flow can help control the topological error.

Of course, the diffusive flow should only be used when there is some indication that the sampling is somewhat non-uniform. On the other hand, if the data is too non-uniform, then the coordinate frame approximations will be inaccurate and the flow along the tangent space will create problems. Additionally, since we need access to the nearest neighbors of the set $\mathcal{X}_t$ to a point $x \in \mathbb{R}^N$, the computational complexity of the algorithm increases. This is because the spatial indexing solution we utilize in Section 4.5 to efficiently compute the nearest neighbors of a point has to be reworked when using the diffusive term. Instead of computing the spatial index once, at the beginning of the flow, and using the same spatial index throughout, we must create a new spatial index at every time t when we need the current nearest neighbors. Thus, when determining whether to add a diffusive term, the computational expense must be considered.

4.3 Adaptive Flow

When we discussed the surface reconstruction problem in Section 1.4, we also discussed a method suggested by Tamal Dey and Jian Sun in [22] for adapting the surface approximation technique based on the local feature size of the point cloud. This technique has obvious advantages since not all point clouds require the same level of smoothing throughout the cloud. As a simple example, consider a point cloud consisting of two separated spheres

B_1 and B_2 of radius r_1 and r_2, respectively, with $r_1 \ll r_2$. In this example, we would benefit from choosing a different step size for the points sampled around B_1 than for those sampled around B_2. Using the same step size would require us to strike a balance between oversmoothing B_1 and undersmoothing B_2. If we were able to approximate the local feature size, as done in [22], we could specify feature dependent step sizes for B_1 and B_2 and smooth both spheres appropriately. Furthermore, suppose both spheres were sampled using the same sample density. Under this assumption, since B_2 is a much larger sphere than B_1, there would be many more points sampled from B_2 than from B_1. In this case, choosing the right number of neighbors to smooth points around B_1 and B_2 becomes difficult. An appropriate value of k for smooothing B_2 would be too large when smoothing B_1. Conversely, if we choose a smaller value of k to accomodate smoothing of B_1, we would not be considering enough neighbors for B_2 and our tangent plane approximations would be skewed by the noise in the sampling procedure.

Similar to the adaptive surface reconstruction technique, we would like to find a way to determine how much smoothing we should apply to the point cloud given local considerations. We may want to adjust the step size, flow time, or even the number of neighbors we consider as a result. To accomplish this, we need a notion akin to the local feature size. However, the local feature size relies on approximating the medial axis of the point cloud. This is accomplished using Voronoi diagrams in [23] and only for two dimensions. Since computing Voronoi diagrams in dimensions higher than two becomes extremely expensive, we cannot use this notion for smoothing higher dimensional point clouds. We need a fast and efficient technique that can fit in our smoothing framework. Instead of using the local feature size, we will attempt to approximate the local curvature of the underlying manifold from which the point cloud was sampled. This is motivated by the fact that for higher curvature regions of the manifold, we want to use fewer neighbors to approximate the tangent plane so that we don't oversmooth the region.

Instead of allowing the gradient flow to continuously adapt its parameters every iteration, we will compute *adaptivity coefficients* for each point in $\mathcal{X}$ at the beginning of the

gradient flow. This cuts down on the required number of computations since we do not need to update these coefficients. These adaptivity coefficients will be based on the local geometry around each point. We will then adapt the gradient flow parameters for each point in $\mathcal{X}$ based on these coefficients. In the remainder of this section, we will only consider modifying the value of k locally. However, adapting the step size or the length of time the gradient flow is allowed to evolve may also prove useful. Finally, since our normal bundle flow requires the calculation of an approximate coordinate frame, we would like to utilize this construction to avoid extra computation.

Ideally, we would have access to the sampled manifold. Then we would measure the curvature at a point using the second fundamental form. That is, given a hypersurface M with second fundamental form $I\!I(X, Y)$, if we fix a point $x \in M$ and have an orthonormal basis $\{X_1, ..., X_{n-1}\}$ for the tangent space $T_x(M)$, then the principal curvatures $\kappa_1, \kappa_2, ..., \kappa_{n-1}$ are the eigenvalues of the matrix

$$\begin{bmatrix} I\!I(X_1, X_1) & I\!I(X_1, X_2) & ... & I\!I(X_1, X_{n-1}) \\ I\!I(X_2, X_1) & I\!I(X_2, X_2) & ... & I\!I(X_2, X_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ I\!I(X_{n-1}, X_1) & I\!I(X_{n-1}, X_2) & ... & I\!I(X_{n-1}, X_{n-1}) \end{bmatrix} \tag{4.10}$$

We could then let the curvature be given by $\kappa = \kappa_1\kappa_2...\kappa_{n-1}$. For large values of κ, i.e. high curvature regions, we would modify the parameters in the gradient flow to reflect the fact that the manifold exhibits high curvature around this point. Of course, we do not have direct access to the manifold. We can only infer the curvature from the sampled point cloud. Therefore, we need an alternative measure of the curvature. For this, we return to the underlying idea behind curvature. Namely, how far does a surface deviate from a hyperplane?

Although we do not know the tangent space or the second fundamental form precisely, we do have approximations for the tangent and normal spaces. In particular, we have the

basis vectors given by the sets $\tilde{T}(x)$ and $\tilde{N}(x)$ in Equation 4.5. Thus, we can determine how far points in $\mathrm{NN}^k_{\mathcal{X}}(x)$ are from the approximated tangent space, i.e. $\mathrm{span}(\tilde{T}(x))$, of x. In fact, for any point $x \in \mathcal{X}$, we can compute the following two values for every neighbor $y \in \mathrm{NN}^k_{\mathcal{X}}(x)$.

$$d_T(x,y) = \left\| \sum_{v \in \tilde{T}(x)} \langle v, y - x \rangle v \right\|, \quad \text{and} \quad d_N(x,y) = \left\| \sum_{w \in \tilde{N}(x)} \langle w, y - x \rangle w \right\| \tag{4.11}$$

These simply reflect the distance between x and y in the tangent space ($d_T(x,y)$) and in the normal space ($d_N(x,y)$). We compute these values for every point $y \in \mathrm{NN}^k_{\mathcal{X}}(x)$ and use the tuples $(d_T(x,y), d_N(x,y))$ to form the set

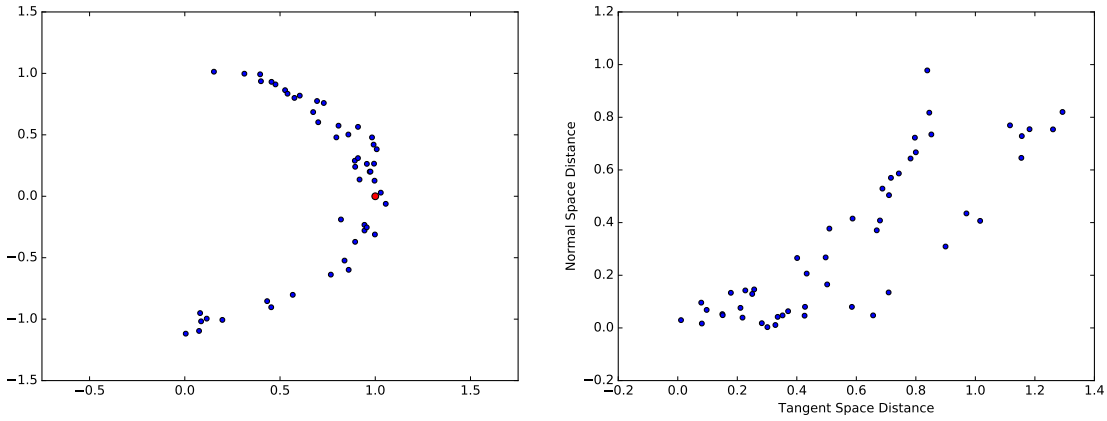
$$P(x) = \{(d_T(x,y), d_N(x,y)) : y \in \mathrm{NN}^k_{\mathcal{X}}(x)\}$$

This set gives us the distance from x to every point in $\mathrm{NN}^k_{\mathcal{X}}(x)$ as measured in the tangent space and the normal space. For example, consider the partial point cloud shown in Figure 4.5. We are interested in the red point, i.e. the point $x = (1, 0)$. We can see the results of computing the set $P(x)$ in Figure 4.5. The idea is that for high curvature regions, as $d_T(x,y)$ increases, the value of $d_N(x,y)$ will increase quickly. We can capture this by computing a linear regression on the set $P(x)$.

That is, we seek the line of best fit for the set $P(x)$. In particular, we employ Ordinary Least Squares estimation to find two parameters, the slope $\lambda \in \mathbb{R}$ and the intercept $b \in \mathbb{R}$, such that the line defined by

$$n = \lambda t + b \tag{4.12}$$

where n is the normal space distance (i.e. $d_N(x, \cdot)$) and t is the tangent space distance (i.e. $d_T(x, \cdot)$), minimizes the sum of the squared residuals. For the point clouds shown in Figure 4.5, we show the line of best fit in Figure 4.6. For this example, we have $\lambda \approx 0.71$ and



(a) Partial Point Cloud

(b) $P(x)$ for $x = (1, 0)$

Figure 4.5: Example Point Cloud and $P(x)$

$b \approx -0.08$.

After computing these regression lines for every point in the point cloud, we can associate a slope, $\lambda(x)$, to every point $x \in \mathcal{X}$. Since we really only care about the magnitude of the slope, we let $\lambda(x)$ actually be the absolute value of the previously computed slope. Then we let

$$m = \min_{x \in \mathcal{X}} \lambda(x) \qquad \text{and} \qquad M = \max_{x \in \mathcal{X}} \lambda(x)$$

and set the adaptivity coefficient $a(x)$ to be

$$a(x) = \frac{\lambda(x) - m}{M - m} \tag{4.13}$$

Thus, the point $x^* \in \mathcal{X}$ with the steepest line of best fit for $P(x^*)$ will receive an adaptivity score of $a(x^*) = 1$ and the point $x_* \in \mathcal{X}$ with the flattest line of best fit for $P(x_*)$ will receive a score of 0. This reflects the fact that points around x^* move away from the tangent space, $\text{span}(\tilde{T}(x^*))$, of x^* much faster than the points around x_* move away from $\text{span}(\tilde{T}(x_*))$.

Finally, we can use the value $a(x)$ to modify the value of k around x. In particular, we

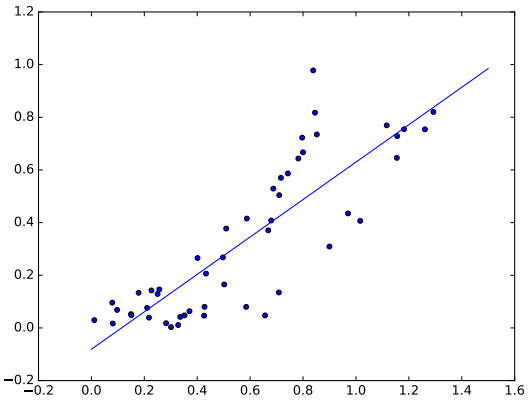


Figure 4.6: Regression line for $P(x)$

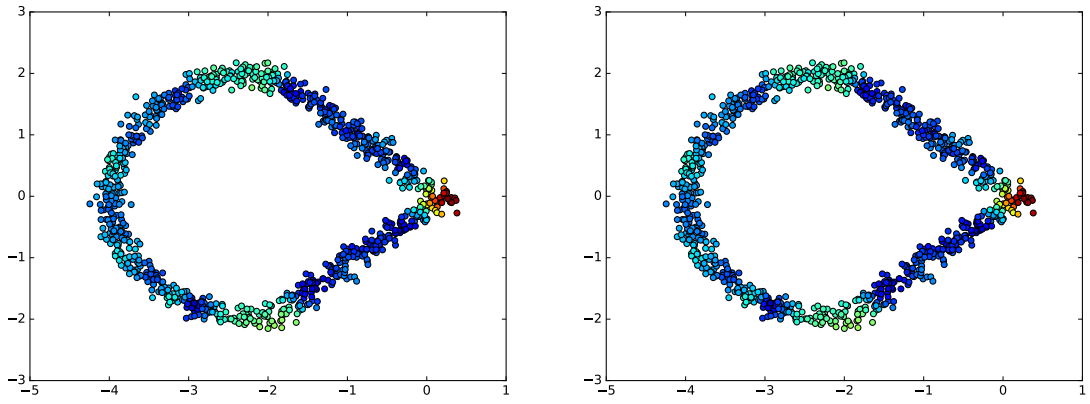
set

$$k(x) = \lfloor (k - k^*)(1 - a(x)) + k^* \rfloor \tag{4.14}$$

where $\lfloor z \rfloor$ is the largest integer n for which $n \leq z$ and $0 < k^* < k$ is chosen to be the minimum desired value for $k(x)$. Choosing k^* depends on the properties of the point cloud to be smoothed. In this section, we show results for $k^* = \lfloor k/2 \rfloor$. Notice that in Equation 4.14 we use the value $1 - a(x)$ to determine the value of $k(x)$. This results in lower values of $k(x)$ for larger values of $a(x)$. Thus, a point x with high curvature will have a lower value of $k(x)$.

Although we have chosen to use the line $n = \lambda t + b$ to fit the data $P(x)$, we could have alternatively chosen to use a quadtratic $n = \lambda_3 t^2 + \lambda_2 t + \lambda_1$ with $\lambda_i \in \mathbb{R}$. In this case, we would actually want to set $\lambda_2 = 0$ to ensure the minimum occurs at $x = 0$. Thus, we would fit the quadratic $n = \lambda_3 t^2 + \lambda_1$ to the data $P(x)$ and use the value λ_3 to determine the adaptivty coefficients. Using a quadratic over a line may allow for better fits to the data. However, due to noise in the data, a line can sometimes produce a better fit.

It should be noted that in practice, we often employ one more step in computing the adaptivity coefficients. In particular, after computing $a(x)$ for every point $x \in \mathcal{X}$ we *smooth*



(a) Linear Fit (b) Quadratic Fit

Figure 4.7: Adaptivity Coefficients $a(x)$

the values of a by taking the average of $a(x)$ over $\mathrm{NN}^s_{\mathcal{X}}(x)$ where $s \ll k$. That is, when determining $k(x)$, we actually use the adaptivity coefficients $\tilde{a}(x)$ defined by

$$\tilde{a}(x) = \frac{1}{s} \sum_{y \in \mathrm{NN}^s_{\mathcal{X}}(x)} a(y) \tag{4.15}$$

This is often a necessary step due to the noise in the point cloud. Although this process adds another parameter to our model, in particular s, this choice is often relatively benign. For our examples, we have taken $s = 5$.

We show an example point cloud in Figure 4.7a which has been color coded to show the value of $\tilde{a}(x)$ as defined in Equation 4.15 using a linear fit (i.e. Equation 4.12). On the other hand, in Figure 4.7b, we show the results under a quadratic fit. Here, we have $\tilde{a}(x) = 1$ show up in red while $\tilde{a}(x) = 0$ appears dark blue. It is interesting to note that not only was the sharp corner on the right of the point cloud properly identified as exhibiting high curvature, both the top and bottom curves were also identified as high curvature (though less so than the right hand corner) and the straight lines leading up to the corner were identified as exhibiting very low curvature. The adaptivity coefficients appear quite similar

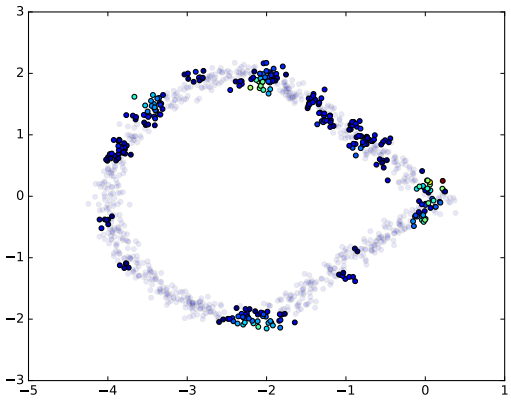
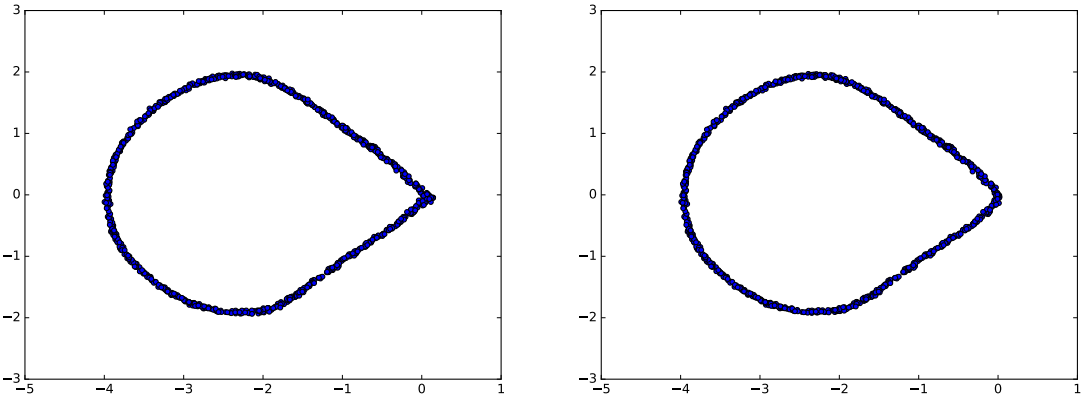


Figure 4.8: Linear and Quadratic Absolute Difference

under the linear and quadratic fits. To make the difference more pronounced, in Figure 4.8, we make highlight the points whose adaptivity coefficients $a(x)$ changed by more than 0.1 between the two methods. The color of these points indicates how much the value of $a(x)$ changed between the two methods, with light blue and yellow points indicating the greatest change. All the other points are made transparent. From this, we see that there are quite a few points impacted by this change.



(a) Adaptive Flow (b) Non-Adaptive Flow

Figure 4.9: Normal bundle flow with and without adaptivity

Smoothing the point cloud given in Figure 4.7 with and without adaptivity reveals how this modification can prove useful. We show the results of the smoothing this point cloud in Figure 4.9. In Figure 4.9a, which uses the adaptive flow we developed in this section, we see that the corner of the original sampling surface has been preserved. On the other hand, the corner has been rounded off in Figure 4.9b, where we did not use the adaptive flow. From this example we see that the adaptive flow helps preserve some of the finer features in this point cloud.

4.4 Mahalanobis Flow

The distance-to-measure function introduced in Section 1.5 provides a measure of the distance from a point to a probability distribution. To produce our smoothing algorithm, we computed the gradient of the distance-to-measure function and used it to induce a gradient flow on the point cloud. In doing so, we were able to move points *closer* to the sampling distribution. By "closer", we mean closer according to some distance function d, which in this case was the distance-to-measure function d^2_{μ,m_0}. A natural question then follows, what other functions can be used for this purpose?

In previous chapters, we mentioned several benefits to using the distance-to-measure function. These included its stability properties, error guarantees, and ease of computation. However, we also noticed some issues with the original algorithm, clustering in particular. Therefore we discussed projecting the gradients along the eigenvectors of the covariance matrix, similar to techniques used in surface reconstruction. This helps prohibit clustering by restricting the trajectories of the points onto the approximate normal directions of the sampled manifold. As it turns out, this modified gradient flow shares a close connection with the Mahalanobis distance, a quantity commonly used in statistics. In this section, we explore this connection.

The **Mahalanobis distance**, introduced by P.C. Mahalanobis in 1936 (see [37]), is a measure of the distance between a point and a distribution. Specifically, given a point

$x \in \mathbb{R}^N$ and a point cloud $\mathcal{X}$ with mean μ and covariance matrix S, the Mahalanobis distance between x and the sampled distribution is estimated to be

$$D_M(x) = \sqrt{(x-\mu)^T S^{-1}(x-\mu)} \tag{4.16}$$

Recall the definition of the covariance matrix S of the random vector $[X_1, ..., X_n]^T$ is the $n \times n$ matrix whose entries E_{ij} are given by

$$S_{ij} = E[X_i X_j] - \mu_i \mu_j$$

where $\mu_i = E[X_i]$ for $1 \leq i, j \leq n$. In what follows, we will be interested in the empirical distribution induced by $\mathcal{X}$ and so we set μ to be the barycenter of the point cloud $\mathcal{X}$. This is an obvious choice since under the emipirical distribution, the expected value of the empirical distribution is given by the barycenter of $\mathcal{X}$.

In the above equation, we use the inverse of the covariance matrix S. Since covariance matrices are only guaranteed to be positive semidefinite, the matrix S may not be invertible. In this case, it is common to use a pseudoinverse, such as the Moore-Penrose pseudoinverse. Let us recall the definition of this pseudoinverse.

Definition 4.2. Let A be an $m \times n$ matrix with entries taken from a field K. The **Moore-Penrose pseudoinverse** is the unique $n \times m$ matrix $A^\dagger$ satisfying the following conditions.

(a) $AA^\dagger A = A$ and $A^\dagger A A^\dagger = A$

(b) $(AA^\dagger)^T = AA^\dagger$ and $(A^\dagger A)^T = A^\dagger A$

For a further discussion of this generalized inverse, see the work by Penrose [47]. This reference contains the simple proof that $A^\dagger$ is unique along with many propeprties of the Moore-Penrose pseudoinverse.

Given the nice structure of S, the pseudoinverse is easy to compute . In particular, since S is a real, symmetric matrix, we can find the eigendecomposition $S = Q\Lambda Q^T$ where Q

consists of the eigenvectors of S and Λ is the diagonal matrix with the associated eigenvalues. Finding the Moore-Penrose inverse $S^\dagger$ is then simply a matter of taking the reciprocals of the diagonal non-zero entries of Λ. That is, for $\Lambda = (\Lambda_{ij})_{i,j=1}^N$ with $\Lambda_{ii} = \lambda_i$ for all i and 0 otherwise, we set $\Lambda^\dagger$ to be the diagonal matrix with $\Lambda_{ii}^\dagger = 1/\lambda_i$ for all i such that $\lambda_i \neq 0$ and $\Lambda_{ii} = 0$ if $\lambda_i = 0$. Then the pseudoinverse is simply $S^\dagger = Q\Lambda^\dagger Q^T$.

Note that since S is symmetric, $S^\dagger$ will also be symmetric. This is easy to see in the Definition 4.2. In particular, if A is symmetric and $A^\dagger$ is the Moore-Penrose inverse of A, then we can take the transpose of all the statements in Definition 4.2 and obtain

$$
\begin{aligned}
AA^\dagger A = A \quad &\Rightarrow \quad A^T(A^\dagger)^T A^T = A^T \quad \Rightarrow \quad A(A^\dagger)^T A = A \\
A^\dagger AA^\dagger = A \quad &\Rightarrow \quad (A^\dagger)^T A^T (A^\dagger)^T = A^T \quad \Rightarrow \quad (A^\dagger)^T A (A^\dagger)^T = A \\
(AA^\dagger)^T = AA^\dagger \quad &\Rightarrow \quad AA^\dagger = (AA^\dagger)^T \\
(A^\dagger A)^T = A^\dagger A \quad &\Rightarrow \quad A^\dagger A = (A^\dagger A)^T
\end{aligned}
$$

Thus, $(A^\dagger)^T$ satisifies all the conditions in Definition 4.2 and so $(A^\dagger)^T$ is a Moore-Penrose pseudoinverse of A. However, Moore-Penrose pseudoinverses are unique. Thus, we must have $A^\dagger = (A^\dagger)^T$ and so $A^\dagger$ is in fact symmetric.

Since $A^\dagger = A^{-1}$ for an invertable matrix A, in what follows we will stick to the $A^\dagger$ notation. Using the pseudoinverse in the Mahalanobis distance when the covariance matrix S proves to be singular is not uncommon. For example, in [32] the authors use the pseudoinverse when their imagery texture data produces singular matrices.

We would like to investigate what happens when we induce a gradient flow according to the Mahalanobis distance. Similar to the distance-to-measure function, the Mahalanobis distance achieves a minimum at the barycenter of the point cloud. To see this, note that $D_M(x) \geq 0$ since S^{-1} is positive semi-definite. Then by setting $x = \mu$, we obtain $D_M(\mu) = 0$

and hence, μ is a minimizer of D_M. This is to be expected since the Mahalanobis distance is an example of a Bregman divergence.

Definition 4.3. Let Ω be a closed, convex set and let $f : \Omega \to \mathbb{R}$ be a strictly convex, continuously differentiable function. The **Bregman divergence** $D_f : \Omega \times \Omega \to \mathbb{R}$ induced by f is defined as

$$D_f(x, y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle$$

The squared Mahalanobis distance is the Bregman divergence induced by the function $g(x) = x^T Ax$. The particular form of the Mahalanobis distance we use above is obtained by taking $D_M(x) = D_g(x, \mu)$ where μ is the barycenter of the point cloud $\mathcal{X}$. A simple property of Bregman divergences is that $D_f(x, y) \geq 0$ for all $x, y \in \Omega$ and $D_f(x, y) = 0$ if and only if $x = y$. Thus, since D_M is in fact a Bregman divergence, we see that $D_M(x) \geq 0$ and $D_M(x) = 0$ if and only if $x = \mu$. Thus, the minmium of the Mahalanobis distance is achieved at the barycenter of the point cloud.

Notice however that the Mahalanobis distance requires the entire point cloud to compute $D_M(x)$. This is in contrast to the distance-to-measure function, which is a local function and only considers a neighborhood of x. Since locality is a desirable property for a smoothing algorithm, we must modify the Mahalanobis distance so that it acts as a local distance measure. The trick is simple, we will only consider the k-nearest neighbors of the point x, instead of the entire point cloud $\mathcal{X}$. Thus, we define the **local Mahalanobis distance**, denoted $D_M^k(x)$ to be

$$D_M^k(x) = \sqrt{(x - \mathrm{Bar}_G(V))^T S_k^\dagger(x)(x - \mathrm{Bar}_G(V))} \tag{4.17}$$

where $V \in V^k(\mathcal{X})$ such that $x \in V$ and $S_k^\dagger(x)$ is the pseudoinverse of the covariance matrix of $\mathrm{NN}_{\mathcal{X}}^k(x)$ as defined in Equation 4.3 for the *unweighted* normal bundle flow. We then follow our previous procedure and induce a gradient flow on the point cloud by taking the gradient of the squared local Mahalanobis distance. This produces the following differential

system, referred to as the **Mahalanobis flow**.

$$\begin{cases} \frac{du}{dt} = -\nabla D_M^k(x)^2 & t > 0 \\ u(0) = x_0 & t = 0 \end{cases} \tag{4.18}$$

Similar to our treatment of the distance-to-measure flow, when computing the barycenter and the covariance matrix we use the original nearest neighbors to avoid clustering. Therefore, we can once again use the k-order Voronoi diagram $V^k(\mathcal{X})$ when discussing the Mahalanobis flow.

Note that from the definition of the local Mahalanobis distance (Equation 4.17), we recognize $D_M^k(x)^2 = (x - \text{Bar}_G(V))^T S_k^\dagger (x - \text{Bar}_G(V))$ as a quadratic form. Thus, if $S_k^\dagger(x)$ is constant in the k-order Voronoi region V for which $x \in V$, then we can compute the gradient using the simple equation for the gradient of a quadratic form ($\nabla_y(y^T A y) = 2Ay$). Unfortunately, the covariance matrix $S_k(x)$ used for the (weighted) normal bundle flow is not constant in V. This is because we use the weighted barycenter when computing the covariance, which depends on x. This is why we must instead use the covariance matrix from the unweighted normal bundle flow. In this case, $S_k^\dagger(x)$ is constant in x and so applying the formula for the gradient of a quadratic form, with $y = x - \text{Bar}_G(V)$ and $A = S_k^\dagger(x)$ being the real, symmetric, square matrix defined above, we obtain

$$\nabla D_M^k(x) = 2S_k^\dagger(x - \text{Bar}_G(V))$$

Inserting this into the above system, we recognize the Mahalanobis flow as follows,

$$\begin{cases} \frac{du}{dt} = -2S_k^\dagger(u)(u - \text{Bar}_G(V)) & t > 0 \\ u(0) = x_0 & t = 0 \end{cases} \tag{4.19}$$

From this, it is readily apparent that the only ways to obtain $\frac{du}{dt} = 0$ are for either $u(t) = \text{Bar}_G(V)$ or $S_k^\dagger(u)w = 0$ with $w = u - \text{Bar}_G(V) \neq 0$. The first case we already noted was a stable point, similar to the distance-to-measure function. The second case requires w to be in the null space of the matrix $S_k^\dagger(u)$. Therefore, $S_k^\dagger(u)$ is singular and w must be orthogonal to every eigenvector v associated with a non-zero eigenvalue λ. This can be seen below, recalling that $S_k^\dagger(u)$ is symmetric,

$$\langle v, w\rangle = \frac{1}{\lambda}\langle \lambda v, w\rangle = \frac{1}{\lambda}\langle S_k^\dagger(u)v, w\rangle = \frac{1}{\lambda}\langle v, S_k^\dagger(u)^T w\rangle = \frac{1}{\lambda}\langle v, S_k^\dagger(u)w\rangle = \frac{1}{\lambda}\langle v, 0\rangle = 0$$

Since the eigenvectors correspond to the covariance matrix, this result tells us there is no variance in the neighbor set $\text{NN}_{\mathcal{X}}^k(x)$ along the direction $w = u - \text{Bar}_G(V)$. In this case, we would not want to move the point u in the direction w since this would be introducing variance in a direction which originally had zero variance. Thus, the fact that this gradient is zero makes sense and is desirable. However, since we are interested in applying this gradient flow to noisy data, in general the matrix $S_k(x)$ will be full rank and so the fix points arising when $x - \text{Bar}_G(V)$ is in the null space of $S_k(x)$ occur with probability zero.

We now seek an explicit formula for the Mahalanobis flow which will let us directly compare the Mahalanobis flow and the normal bundle flow. For the following line of reasoning, we will fix x and simplify our notation by setting $S_k^\dagger(x) = S^\dagger$ and $B = \text{Bar}_G(V)$. Recall the simple structure of the pseudoinverse $S^\dagger = Q\Lambda^\dagger Q^T$. Note that since the matrix S is positive semi-definite, we are guaranteed that the eigenvalues of S will be non-negative. Let $n^* \leq n$ be the number of non-zero eigenvalues. Denote the non-zero eigenvalues of S as λ_i with $i \leq n^*$ and ordered according to magnitude, i.e. $0 < \lambda_1 \leq ... \leq \lambda_{n^*}$. Also, let $\lambda_i = 0$ for $i > n^*$. Finally, let e_i be the eigenvector associated with λ_i. Note that the eigendecomposition is $S^\dagger = Q\Lambda^\dagger Q^T = Q\Lambda^\dagger Q^{-1}$, and so we can take the eigenvectors to be normalized to length one since the product involves both Q and Q^{-1}. Thus, we assume

$\|e_i\| = 1$ for all $i \leq n^*$. Then we can decompose the gradient of $D_M^k(x)$ as follows.

$$\nabla D_M^k(x) = 2S^\dagger(x - B) = 2Q\Lambda^\dagger Q^T(x - B) = 2\sum_{i=1}^{n^*} \frac{1}{\lambda_i} e_i e_i^T (x - B) = 2\sum_{i=1}^{n^*} \frac{1}{\lambda_i} \langle x - B, e_i \rangle e_i \tag{4.20}$$

Now let us consider the normal bundle flow. Recall that for the projected k-nearest neighbor gradient, we must specify the codimension of the sampled manifold to approximate the normal space of the manifold. Let m denote this codimension. Thus, we use the gradient $g_N(x)$ which takes the following form.

$$g_N(x) = \sum_{i=1}^{m} \langle \nabla E(x), e_i \rangle e_i = \sum_{i=1}^{m} \left\langle \frac{2}{k} \sum_{y \in \mathrm{NN}_{\mathcal{X}}^k(x)} (x - y), e_i \right\rangle e_i = 2\sum_{i=1}^{m} \langle x - B, e_i \rangle e_i \tag{4.21}$$

Note that since we use the same covariance matrix for both flows, the eigenvectors v_i will be the same for both flows.

From Equations 4.20 and 4.21 the connection between the two flows becomes clear. Although the two flows share remarkably similar structure, there are a few key differences that cause the behavior of a point cloud under these two flows to be different.

First, the number of eigenvectors used in each equation is different. In particular, we will always have $m \leq n^*$ and so the number of eigenvectors in Equation 4.20 will always be greater than or equal to that used in Equation 4.21. An advantage of the Mahalanobis flow over the normal bundle approximation flow is that the dimensionality of the manifold does not need to be specified a priori. This can be of great benefit for high dimensional data sets where the intrinsic dimension is unknown.

The other major difference between the two flows concerns the fact that the scale factor used for each eigenvector in the sum varies dramatically. In the Mahalanobis flow, the scaling factor for each eigenvector is the reciprocal of the corresponding eigenvalue. Thus, for smaller eigenvalues, there will be more motion in the direction of the corresponding

eigenvector. Since in our setting we take eigenvectors with small eigenvalues to be approximations of the normal directions, this implies that the Mahalanobis flow boosts the flow along the "normal directions". The normal bundle flow does not follow this approach. Instead, the eigenvectors with the smallest eigenvalues are taken as approximate normal directions and then each of these eigenvectors is weighted equally in the flow. Although it might make sense to weight each eigenvector based on how likely it is to represent the normal direction, taking the reciprocal will likely produce far too much boosting in the given direction. In particular, for an eigenvector v with eigenvalue λ, the scaling factor, $1/\lambda$, grows to infinity as $\lambda \to 0$. Thus, the gradient may be dominated by eigenvectors with very small but positive eigenvalues.

Taking the similarity between the two equations as a cue, it seems natural to generalize the flow as follows. Let $W \in \mathbb{R}^{n^*}$ be a weight vector and set

$$g_W(x) = 2\sum_{i=1}^{n^*} W_i \langle x - B, e_i \rangle e_i \tag{4.22}$$

We can recover Equation 4.20 by settings $W_i = 1/\lambda_i$ and Equation 4.21 by setting $W_i = 1$ for $i \leq m$ and $W_i = 0$ for $m < i \leq n*$.

To find the best of both worlds, we can consider a weight vector W defined as follows. We set

$$\tau = \sum_{i=1}^{n^*} \frac{1}{\lambda_i}$$

and then set

$$W_i = \frac{1/\lambda_i}{\tau} \tag{4.23}$$

We refer to the resulting gradient flow as the **normalized Mahalanobis flow**. The benefit of this approach is that the scaling factor will always be bounded by one, but the flow will still be boosted along the eigenvectors with smallest eigenvalues (i.e. the approximate

normal directions). Additionally, the dimensionality of the manifold does not need to be specified a priori, unlike the normal bundle flow.

There are certainly many other choices for the weight vector in Equation 4.22. In this dissertation, we have investigated three different choices; a binary scheme corresponding to the normal bundle flow, the reciprocal of the eigenvalues corresponding to the Mahalanobis flow, and the normalized reciprocal eigenvalues corresponding to the normalized Mahalanobis flow. The two approaches given in Equation 4.20 and Equation 4.21 each have advantages and disadvantages. The normalized Mahalanobis flow defined by Equation 4.23 was designed to combine the advantages of both approaches and diminish the disadvantages.

4.5 Nearest Neighbor Calculations

We close this chapter with a quick discussion of the nearest neighbor calculation. While computing the k-nearest neighbor gradient (Equation 3.3) and even computing the normal directions (Equation 4.7) are relatively inexpensive operations, the computation of the k-nearest neighbor set is the overwhelming driver of the computational complexity of the algorithms described in this dissertation. This operation näively exhibits quadratic complexity since for a point cloud $\mathcal{X}$ with $|\mathcal{X}| = n$, we must compute $n(n-1)/2$ distances to find the k nearest neighbors of all the points. Thus, running the smoothing algorithms described in this section on large data sets becomes impractical.

We employ a common method of reducing the complexity of the nearest neighbor operation by using a space partitioning method. In particular, we will employ a k-d tree to obtain the complexity savings. It is important to note that the name k-d tree is conventional and that k, as used in this context, refers to the dimensionality of the data set upon which the k-d tree is constructed, it is not used in the same context we have been using in reference to the number of neighbors. A k-d tree provides a partitioning of ambient space such that searching for neighbors is made efficient. A tree is built from the data set so that the leaf nodes of the tree represent the data points and the non-leaf nodes represent hyper-planes which subdivide the space. To find a nearest neighbor of a sample point, one can simply

use the properties of the tree to climb up the tree and descend into adjacent regions looking for the nearest neighbor. Since we very rarely need to ascend the entire length of the tree, much of the search is eliminated, providing significant compuational savings. The specific k-d tree technique we use in our implementations of the algorithms described in this section can be found in the paper by Maneewongvatana and Mount [38].

Although k-d trees provide large savings for lower dimensional datasets, higher dimensional point clouds cause the method to becomes less and less effective. This is because as the dimensionality increases, the number of points which must be considered at each recursion of the tree grows larger. Thus, it is often recommended that for an N-dimensional point cloud $\mathcal{X}$, we need $|\mathcal{X}| \gg 2^N$. This, of course, is a consequence of the curse of dimensionality. However, by using *approximate nearest neighbors* instead of exact nearest neighbors, we can gain computational savings for high dimensional point clouds as well. Allowing some errors in the selection of a point's neighbors can drastically reduce the complexity of finding nearest neighbors in high dimensional spaces. Techniques for approximate nearest neighbors include *locality-sensitive hashing* and *best bin first* which is an adaptation of k-d trees.

Chapter 5: Applications

In this chapter, we will conduct several numerical experiments with the k-nearest neighbors gradient flow and the normal bundle flow. Our goal will be to determine optimal parameters for the two flows when applied to point clouds sampled from various manifolds. To begin, we will use simple geometric shapes as our sampling manifolds. We use these simple geometries since we know both the geometry and topology of these shapes. This allows us to measure both the geometric and topological error of a point cloud smoothed using the two flows. Next, we will apply the gradient flows to noisy point clouds drawn from the Stanford Bunny and the Stanford dragon. These pose new challenges for the gradient flows since they are much more complicated geometries. As we did with the simple geometries, we measure the geometric error of the point clouds obtained after smoothing the noisy clouds using the two flows. Finally, we will apply the gradient flows to LiDAR point clouds with hundreds of thousands of points. Since we have no knowledge of the ground truth for these point clouds, we use these experiments to analyze the computational cost of running the gradient flow algorithms.

Since we will be implementing these gradient flows on a computer to perform the experiments, we have two choices. First, we can compute the k-order Voronoi diagram and solve the gradient flow system exactly using the tools developed in Chapter 3. Of course, this will only allow us to solve the k-nearest neighbor flow, not the normal bundle flow. Furthermore, computing the k-order Voronoi diagram for large values of k and in three or more dimensions is computationally extremely expensive. Therefore, instead of following this approach, we follow the alternative approach, implementing a discretized gradient flow.

In particular, for a gradient $\nabla E(u)$, instead of using the continous gradient flow system

$$\begin{cases} \dfrac{du}{dt} = -\nabla E(u) \\ \\ u(0) = x \end{cases}$$

we use the discretized flow given by

$$u(t_{i+1}) = u(t_i) - \sigma \nabla E(u(t_i))$$

$$u(0) = x$$

where we discretize time to be $\{0, t_1, ..., t_T\}$ and use step size $\sigma > 0$. In this case, we say that the gradient flow system was run for T iterations. The step size is chosen to control the smoothness and accuracy of this discrete approximation. Of course, too small a step size would cause the discretized gradient flow to progress very smoothly and slowly, but would yield greater accuracy. On the other hand, a large step size would allow the gradient flow to move points further during each iterations, but this would come at a loss of accuracy. From the framework we build in Chapter 3, we know that if the step-size is chosen to be too large, a point may jump over an entire k-order Voronoi region when it moves along the gradient. Therefore, the size of the smallest k-order Voronoi region helps determine the optimal step size. However, for this work, we will simply choose the step size to be $\sigma = 0.01$ for all our experiments.

In all the experiments that follow, we use this discretized gradient flow instead of the continuous gradient flow. Thus, when we refer to the k-nearest neighbor flow or the normal bundle flow, we are really refering to the discretized versions of these gradient flows. Additionally, when we refer to time, we are talking about the number of iterations for which the gradient flow was run.

5.1 Simple Geometries

In this section, we will present numerical results obtained through systematic testing of the described algorithms on point clouds sampled from several simple geometric shapes embedded in Euclidean spaces of varying dimensions. In particular, we will see how the smoothing induced by the k-nearest neighbors gradient flow and the normal bundle flow reduce the error in the sampled point cloud. Since the geometric shapes from which we will sample are known a priori, we can construct error functions and inspect the point clouds to find which parameter values yield the minimum geometric error. Most of the shapes we will investigate exhibit many symmetries which diminish the need for using an adaptive flow. However, the capsule we study in Section 5.1.2 will benefit from the use of the adaptive flow. Therefore, in this section we will also study the performance of the normal bundle flow with adaptivity.

Our goal in this analysis will be to determine the optimal values of k and the optimal number of iterations T to run the gradient flows. Choosing too high a value of k will result in oversmoothing, destroying important features in the process. On the other hand, too low a value of k will fail to remove the noise in the point cloud. We can also induce over-smoothing or under-smoothing by our choice of the number of iterations. Of course, running the gradient flow indefinitely is not always ideal since many features may collapse during the flow. We report exactly when, for a range of values of k, the optimal stopping time is reached. Additionally, we can report the minimum error present during the flow, which corresponds to the error reached with the optimal value of k after the optimal number of iterations.

5.1.1 Circle

For our first experiment, we will sample a point cloud from the unit circle

$$S^1 = \{x \in \mathbb{R}^2 : \|x\| = 1\}$$

Since we wish to incorporate noise into our model, we will compute a point (x, y) by setting

$$x = r\cos(\theta) \qquad y = r\sin(\theta)$$

and sampling (r, θ) according the following probability distributions

$$r \sim \mathcal{N}(1, \sigma) \qquad \theta \sim U(0, 2\pi)$$

where $\mathcal{N}(m, \sigma)$ is the Gaussian distribution with mean m and standard deviation $\sigma > 0$ and $U(a, b)$ is the uniform distribution on the interval $(a, b) \subset \mathbb{R}$. Following this process, we sample a point cloud $\mathcal{X}$ with $|\mathcal{X}| = 1000$. An example point cloud resulting from this procedure with $\sigma = 0.1$ is given in Figure 5.1. Clearly, this point cloud exhibits plenty of noise as some rather significant outliers.

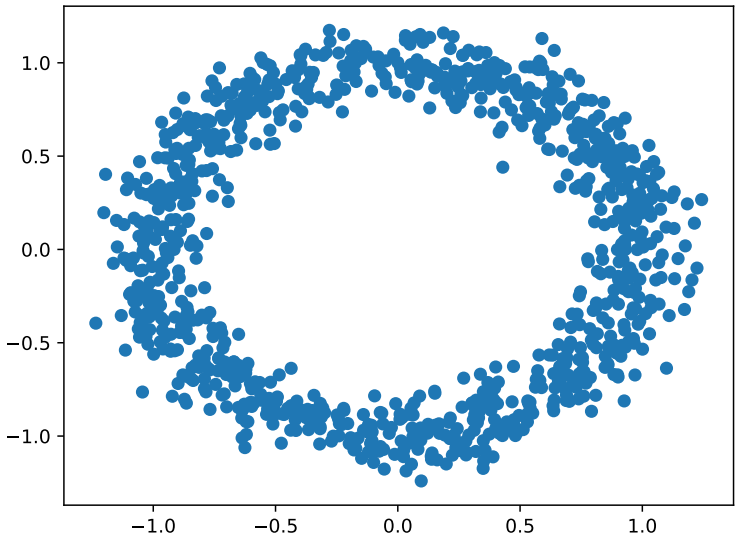


Figure 5.1: Point Cloud Sampled from S^1

For this manifold, we can formulate the geometric error function $E_G(\mathcal{Y})$ for the smoothed

point cloud $\mathcal{Y}$ as

$$E_G(\mathcal{Y}) = \sum_{y \in Y} (1 - \|y\|)^2$$

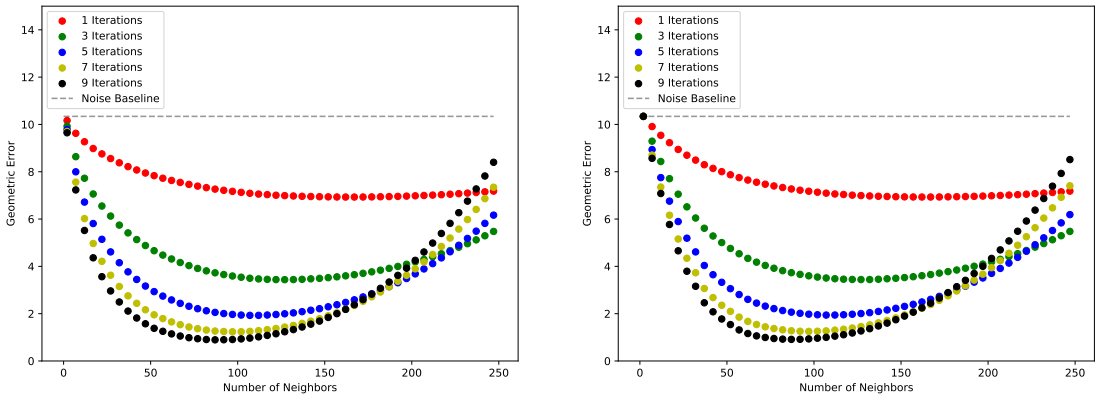
The simplicity of this formula arises from the simplicity of our manifold. In particular, since the closest point y' on S^1 to a point $y \in \mathbb{R}^2$ is given by $y' = y/\|y\|$, we see that the squared Euclidean distance between y and y' is given by

$$\|y' - y\|^2 = \left\| \frac{y}{\|y\|} - y \right\|^2 = \left(\frac{1}{\|y\|} - 1 \right)^2 \|y\|^2 = (1 - \|y\|)^2$$

and so we are really finding the sum of the squared distances between each point $y \in \mathcal{Y}$ and its closest point in S^1.

Notice that the manifold from which we are sampling, S^1, is a one dimensional manifold which we are embedding in $\mathbb{R}^2$. Hence, it has codimension one. Therefore, in the normal bundle flow, we will be computing a single normal vector at each point, thus constituting our normal bundle. In a later example, we will consider manifolds with codimension different from one.

Since we are interested in finding the optimal parameters for the two algorithms, i.e. the k-nearest neighbor flow and the normal bundle flow, we will perform a parameter sweep on the sampled point clouds. As we will see, the optimal parameters vary with the amount of noise present in the system. To make this clear, we have provided error plots for point clouds sampled from S^1, using the noisy procedure outlied above, using the standard deviation values $\sigma = 0.1$, $\sigma = 0.075$, $\sigma = 0.05$, and $\sigma = 0.025$. Inspecting the error plots for these point clouds makes the connection between the optimal parameters and the level of noise, i.e. σ, rather clear.



(a) k-NN Flow (b) Normal Bundle Flow

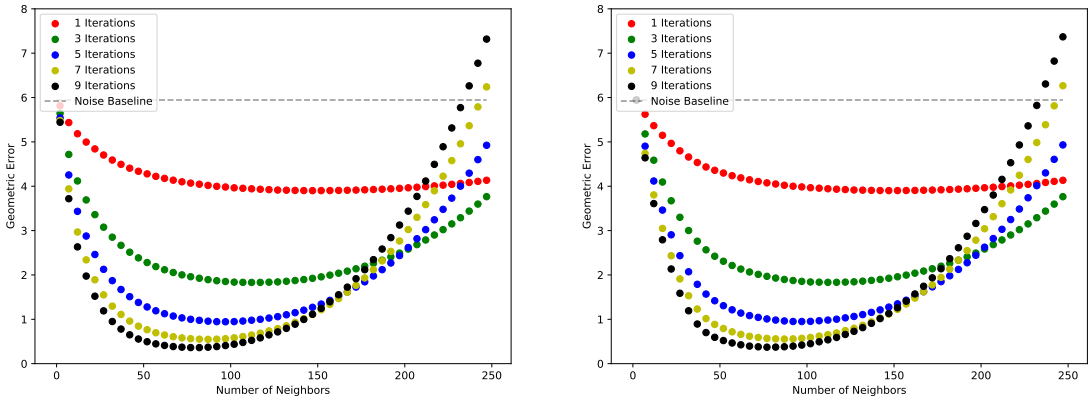
Figure 5.2: Geometric Error Rates for S^1 with $\sigma = 0.1$

Geometric Error

The first error plot we encounter, Figure 5.2, is that for $\sigma = 0.1$. In this figure, we show the geometric error, as calculated by E_G, after running the k-nearest neighbor flow, Figure 5.2(a), and the normal bundle flow, Figure 5.2(b). In each plot, we show the error for different values of k (i.e. the number of neighbors). Additionally, we show the original error in the point cloud $\mathcal{X}$, that is $E_G(\mathcal{X})$. This is represented in the plot as a dotted black line, referred to as the *noise baseline* in the figures. Any parameter values which produce an error below this line can be considered to be improving the point cloud from a noisiness perspective.

From Figure 5.2, we see that both the k-nearest neighbor and normal bundle flows performed best when allowed to run for greater iterations. We have decided to only show the error values for iterations $t = 1, 3, 5, 7, 9$ since for greater values, the plots begin to overlap. This is because the gradient flows begin converging around $t = 9$. While hard to see from the plots, the normal bundle flow does actually perform slightly better than the k-nearest neighbor flow, achiving a lower overall error.

We also see that both the k-nearest neighbor flow and the normal bundle flow have strikingly similar performance on this point cloud. This will be a theme that we see throughout the Simple Geometries section. However, the similarity will disappear when we consider more complicated geometries in the sections to follow. For now, notice that both flows reach optimality around $k = 75$ for $t = 9$. However, the optimal value of k changes depending on how many iterations the flows are run. For example, while $k = 75$ yields around the best performance for the k-nearest neighbor flow with $t = 9$, if we look at $t = 3$, we get better performance around $k = 100$. Thus, there seems to be a trade off between the value of k and t. Higher values of k require fewer iterations to reach their minmial error. However, this minimal error will be greater than the minimal error achieved using nine iterations and around 75 neighbors.

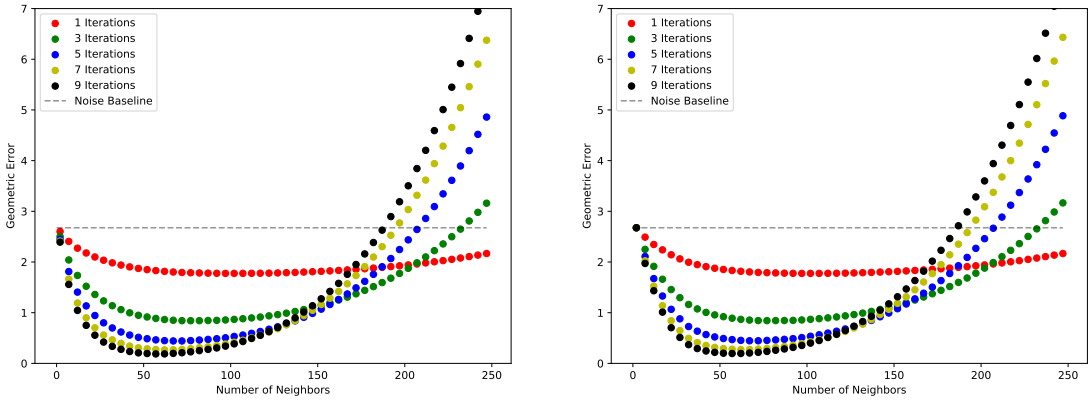


(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.3: Geometric Error Rates for S^1 with $\sigma = 0.075$

Figures 5.3, 5.4, and 5.5 show the same error plots for points clouds sampled from S^1 with $\sigma = 0.075$, 0.05, and 0.025 respectively. Of course, as σ decreases, so too does the noise baseline. Perhaps more interesting, we also see that as we decrease σ, the optimal value of k decreases. This makes sense since increasing the value of k has the effect of increasing the amount of smoothing being applied to the point cloud. If we have less initial error, then as expected we should require a lower value of k. In the four cases we investigated, associated

with the four values of σ, the optimal value of k went from around $k = 75$ ($\sigma = 0.1$), to around $k = 70$ ($\sigma = 0.075$), to around $k = 50$ ($\sigma = 0.05$), and finally to around $k = 40$ ($\sigma = 0.025$). As expected, the minimal error also decreased for both algorithms as we decreased σ.

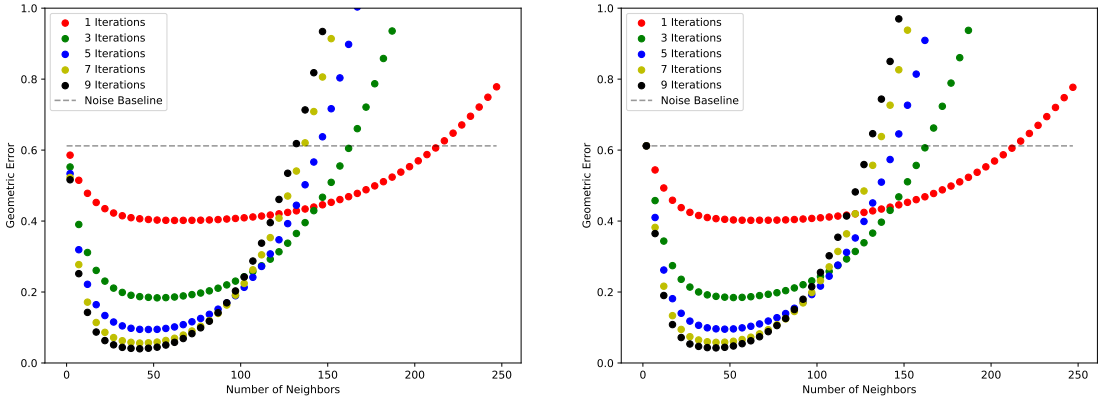


(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.4: Geometric Error Rates for S^1 with $\sigma = 0.05$

Looking closer at the errors we see that the flows initially decrease the value of E_G as we increase k, however after the optimal value of k, we see E_G rise sharpely. This is due to volumetric shrinking of the point cloud. This occurs because we are utilizing the barycenter of a collection of points. If that collection happens to represent some portion of a convex surface, the barycenter will lie *below* the convex surface. Hence, over time, the volume of the point cloud will shrink. Although combatting volumetric shrinking is outside the scope of this dissertation, it is important to note that just because the geometric error may be increasing beyond some values of k and t, the topological error may not be increasing. It could, in fact, continue decreasing. This is why we must also look at topological measures of error.

Before we turn to our topological discussion, we would like to call attention to Figure 5.6. Here, we see the two point clouds produced by the optimal values of k and t for both the

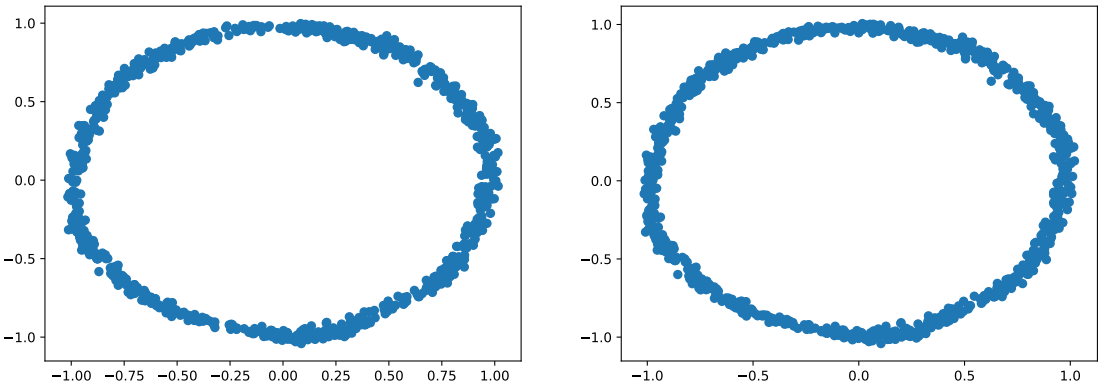


(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.5: Geometric Error Rates for S^1 with $\sigma = 0.025$

k-nearest neighbor flow, Figure 5.6(a), and the normal bundle flow, Figure 5.6(b). Recall that the normal bundle flow was motivated by the fact that the k-nearest neighbor flow would result in points clustering together over time. This is evident in Figure 5.6(a) where several gaps are plainly visible. However, in Figure 5.6(b), there are no visible gaps. Of course, each point is zero-dimensional and thus takes up zero area, however the scatter plots here can be seen as a covering of the point cloud $\mathcal{Y}$. From such a covering, we can construct the Cěch-complex and compute the homology. In this case, Figure 5.6(b) would exhibit a topology more similar to S^1 than would the point cloud in Figure 5.6(a).

Finally, notice that in the first three plots, for the single iteration line (i.e. $t = 1$), after intially descending, the error values remain relatively flat as we increase k. This is simply an artifact of the choice for the range of the bottom axis. In actuality, if we continued to increase k, the first three plots would begin to look like the fourth plot (i.e. $\sigma = 0.025$) and begin to rises sharpely for large enough k.



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.6: Geometrically Optimal Point Clouds for S^1 with $\sigma = 0.1$

Topological Error

As we saw in the previous section, both the k-nearest neighbors flow and the normal bundle flow reduce the geometric error of the point cloud. Of course, the degree to which the geometric error is reduced, depends on the chosen parameter values. However, geometric error is not the only way to measure error in a point cloud.

One issue that was clear from the plots in the previous section was volumetric shrinking. When considering geometric error, volumetric shrinking will be a major problem since once the smoothed point cloud becomes smaller than the sampled manifold, the geometric error increases sharpely. On the other hand, volumetric shrinking is not an issue for the topological error. This is because topological considerations are scale invariant. For example, if we are given a point cloud $\mathcal{X}$ and we create another point cloud $\mathcal{X}' = \{2x : x \in \mathcal{X}\}$, the the persistence diagrams of $\mathcal{X}$ and $\mathcal{X}'$ would be identical. Thus, even though $\mathcal{X}$ is much smaller than $\mathcal{X}'$ (in a volumetric sense), the persistence diagrams agree and so topologically, these point clouds can be considered the same.

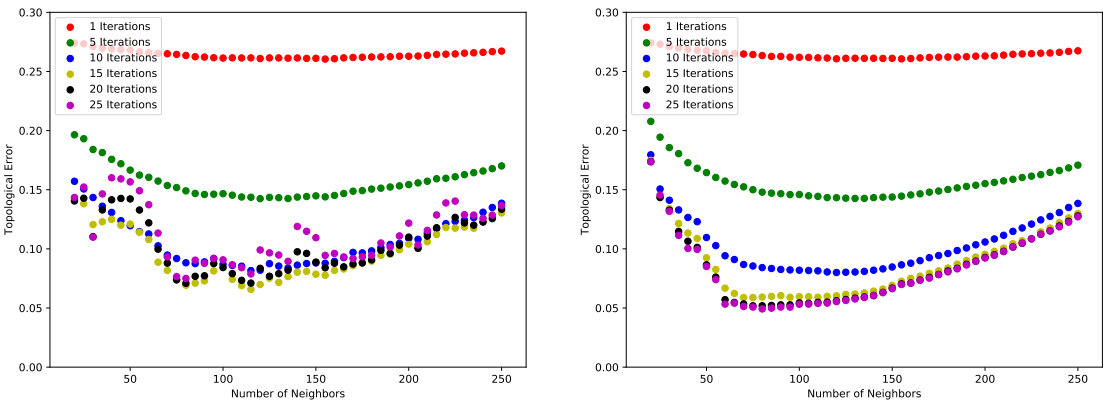
Since we know the topology of S^1, we know that the one-dimensional persistence diagram of S^1, denoted $\mathrm{Dgm}_1(S^1)$, will consist of a single point at $(0, 1)$. To see this, supposed we

uniformly sampled S^1, generating n points on S^1, denoted $\mathcal{X}_n$. Next, let r_n be the minimum resolution r for which the α-complex, $\text{Alpha}_{\mathcal{X}_n}(r)$, contains a single connected component (just as S^1 contains a single connected component). Then as $n \to \infty$, we will have $r_n \to 0$ since the sampling of S^1 becomes increasingly dense.

From this fact, we can construct a function, E_T^p, for the topological error between S^1 and a point cloud $\mathcal{Y}$. Recall that is function was developed at the end of Section 2.4. In particular, we let E_T^p be the bottleneck distance between the p-dimensional persistence diagram of S^1 and the p-dimensional persistence diagram induced by the filtration $\mathcal{F}_\mathcal{Y}$ of the α-complex corresponding to $\mathcal{Y}$. That is,

$$E_T^p(\mathcal{Y}) = d_B(\text{Dgm}_p(S^1), \text{Dgm}_p(\mathcal{F}_\mathcal{Y})) \tag{5.1}$$

Using this function, we have a notion of the topological difference between a point cloud $\mathcal{Y}$ sampled from a manifold, in this case S^1. In the sections that follow, we will redefine E_T^p to reflect the manifold being sampled.



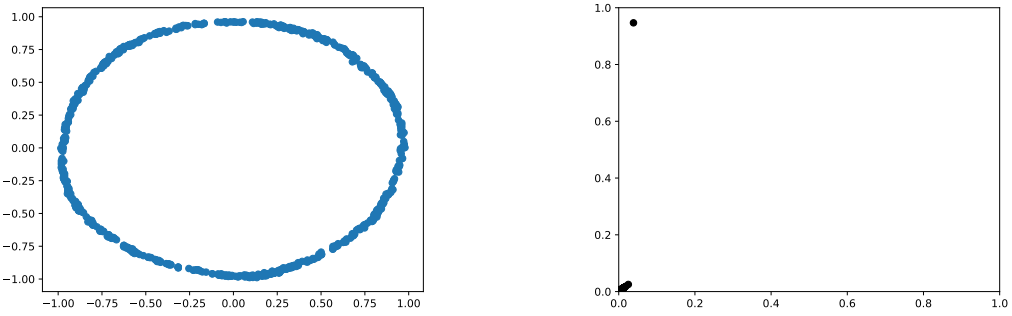
(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.7: Topological Error Rates for S^1 with $\sigma = 0.1$

In Figure 5.7, we show the one-dimensional topological error $E_T^1(\mathcal{X}_t)$ for point clouds

smoothed using the same parameters we chose previously. These figures are quite interesting. The most immediate observation is that the smoothing induced by the normal bundle flow is much more topologically stable as the iteration count increases than is the smoothing induced by the k-nearest neighbor flow. This can be seen by comparing Figures 5.7(a) and 5.7(b). In particular, while the error rates form relatively smooth curves for both the the k-nearest neighbor flow (a) and normal bundle flow (b) with few iterations, after around 10 iterations, the k-nearest neighbor flow begins losing stability across values of k. On the other hand, the normal bundle flow maintains its stability across k even after 25 iterations.

We can also see that the optimal topological error achieved by the normal bundle flow is lower than the optimal error achieved by the k-nearest neighbor flow. Specifically, the normal bundle flow achieves a minimum topological error of around $E_T^1(\mathcal{Y}) = 0.0487$ for the point cloud $\mathcal{Y}$ created by evolving the original point cloud $\mathcal{X}$ using the parameters $k = 80$ and $t = 30$, while the k-nearest neighbor flow achieves a minimum topological error around $E_T^1(\mathcal{Y}) = 0.0658$ for $k = 115$ and $t = 15$. Thus, we see that not only does the normal bundle flow achieve a lower topological error, it also does so with fewer neighbors. This helps since nearest neighbor computations are expensive.

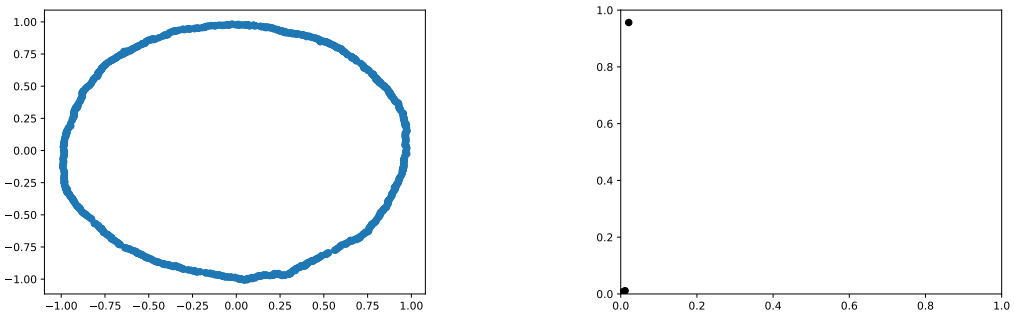


(a) Smoothed Point Cloud

(b) Persistence Diagram

Figure 5.8: Topologically Optimal Point Cloud for k-NN Flow

In Figures 5.8 and 5.9, we show the point clouds resulting from the topologically optimal parameter values associated with the k-nearest neighbor flow and the normal bundle,



(a) Smoothed Point Cloud

(b) Persistence Diagram

Figure 5.9: Topologically Optimal Point Cloud for Normal Bundle Flow

respectively. In addition to the topologically most accurate point cloud, we show the one dimensional persistence diagram associated with the point cloud. From this, it is clear that the persistence diagram in Figure 5.9(b) more closely resembles the persistence diagram of S^1, consisting of a single point at $(1, 0)$, than does the persistence diagram in Figure 5.9(b), corresponding to the k-nearest neighbor flow. This reflects the fact that the normal bundle flow achieves lower topological error than the k-nearest neighbor flow. However, it is worth noting that the topologically optimal point cloud for the k-nearest neighbor flow does produce a point cloud which appears rounder, and hence more like the unit circle, than the normal bundle flow. The lower topological error provided by the normal bundle flow comes strictly from the fact that the points in Figure 5.9(b) provide a denser sampling of the unit circle and therefore the resolution r required for the α-complex of the point cloud to exhibit a single connected component is lower for the normal bundle flow than for the k-nearest neighbor flow.

In this section, we have shown how the k-nearest neighbor flow and the normal bundle flow perform when smoothing a point cloud drawn from the unit circle, S^1. We have seen that the normal bundle flow produces both the lower geometric error point cloud as well as the lower topological error point cloud. In the sections that follow, we will perform the same anlaysis with point clouds drawn from different manifolds. To add variability and difficulty

to the smoothing process, each manifold was chosen to reflect a unique set of smoothing requirements. In the next section, we consider a geometric shape which has varying degrees of curvature throughout the manifold.

5.1.2 Capsule

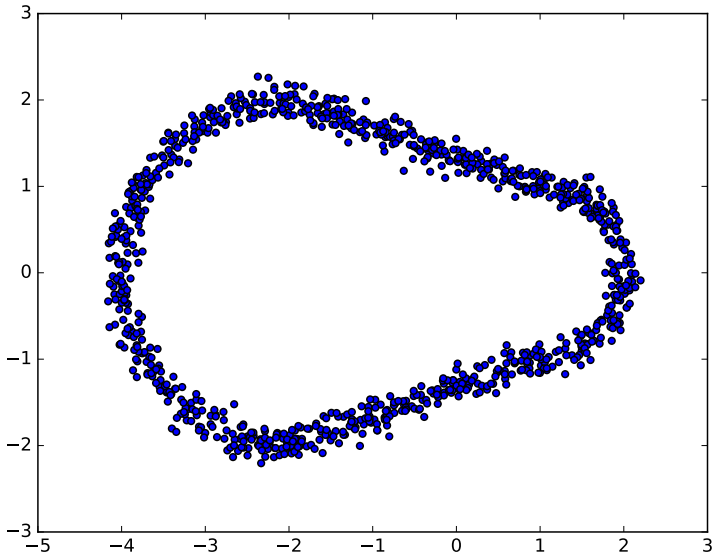


Figure 5.10: Point Cloud Sampled from W

We now turn our attention to a geometric shape we refer to as a *capsule*. This shape is simply two half circles of differing radii connected by two line segments. In particular, the shape is made of four components, the left half circle C_ℓ which has radius r_ℓ and is centered at $(-r_\ell, 0)$, the right half circle C_r of radius r_r and centered at $(r_r, 0)$, the top line segment L_t which is a line from $(-r_\ell, r_\ell)$ to (r_r, r_r), and the bottom line segment L_b which is a line from $(-r_\ell, -r_\ell)$ to $(r_r, -r_r)$. Taken together, these four components form the capsule $\mathrm{Cap}(r_\ell, r_r)$.

To sample the capsule, we first compute the perimeter $p(r_\ell, r_r)$ of $\mathrm{Cap}(r_\ell, r_r)$ using the

equation

$$p(r_\ell, r_r) = (r_\ell + r_r)\pi + 2\sqrt{(r_r + r_\ell)^2 + (r_r - r_\ell)^2}$$

Then for each components C, we compute the ratio $F(C)$ of the perimeter that is due to that component. That is, for one of the half circle components, either C_ℓ or C_r, we let $F(C_i)$ (for $i = \ell$ or $i = r$) be defined as

$$F(C_i) = \frac{r_i \pi}{p(r_\ell, r_r)}$$

and for either of the line segments L, we let

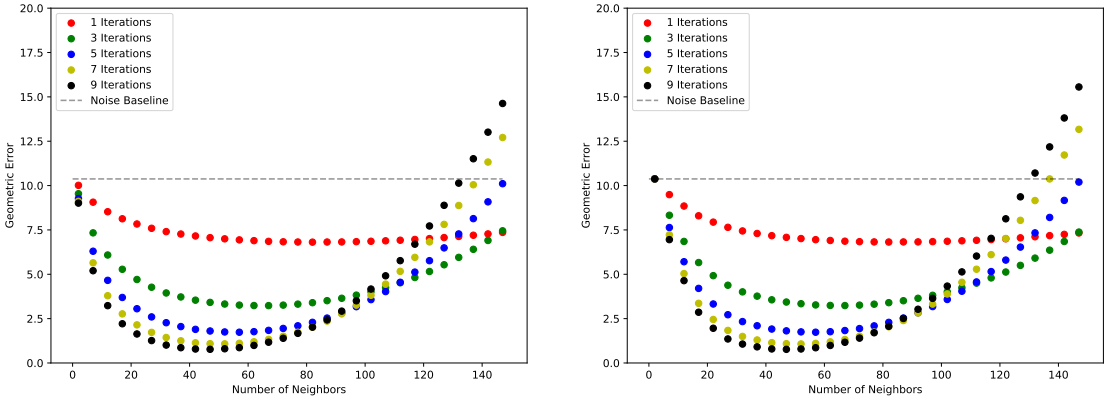
$$F(L) = \frac{\sqrt{(r_r + r_\ell)^2 + (r_r - r_\ell)^2}}{p(r_\ell, r_r)}$$

Then to generate a sampling of $\mathrm{Cap}(r_\ell, r_r)$, we randomly choose a component to sample where the probability of a component C is given by $F(C)$. Once the component is chosen, we sample from that component and add a zero-centered Gaussian noise term $\mathcal{N}(0, \sigma)$ with standard deviation σ. An example of a point cloud sampled from this process is shown in Figure 5.10. In this case, we set $r_\ell = 2$ and $r_r = 1$. Notice that the point cloud in this figure is uniformly sampled.

Geometric Error

To compute the geometric error $E_G(\mathcal{Y})$, we take the sum over all the points of the square distance to $\mathrm{Cap}(r_\ell, r_r)$. In particular, for a point $x \in \mathbb{R}^N$, we set

$$d_{\mathrm{Cap}(r_\ell, r_r)}(x) = \min\left\{d_{C_\ell}(x), d_{C_r}(x), d_{L_t}(x), d_{L_b}(x)\right\}$$

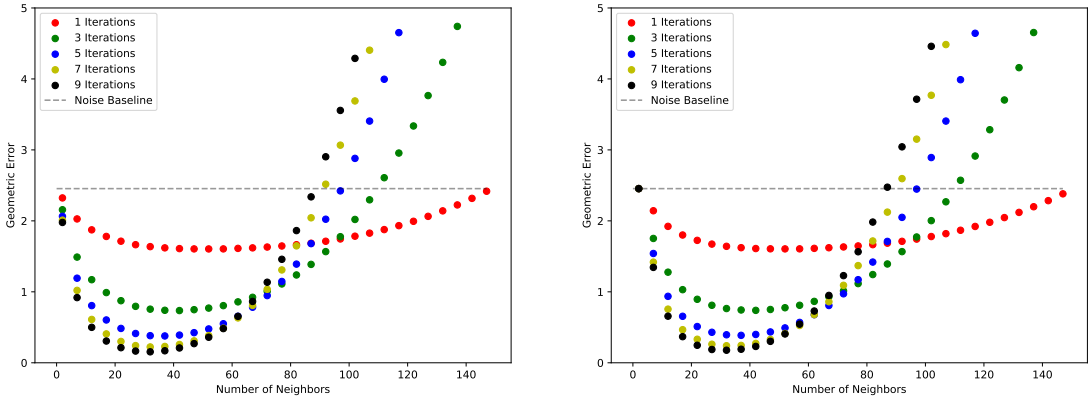


(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.11: Geometric Error Rates for $\mathrm{Cap}(2,1)$ with $\sigma = 0.1$

where $d_C(x)$ is the distance from x to the component C. Since each component is either a line or a half circle, it is easy to compute $d_C(x)$. Then the geometric error is given by

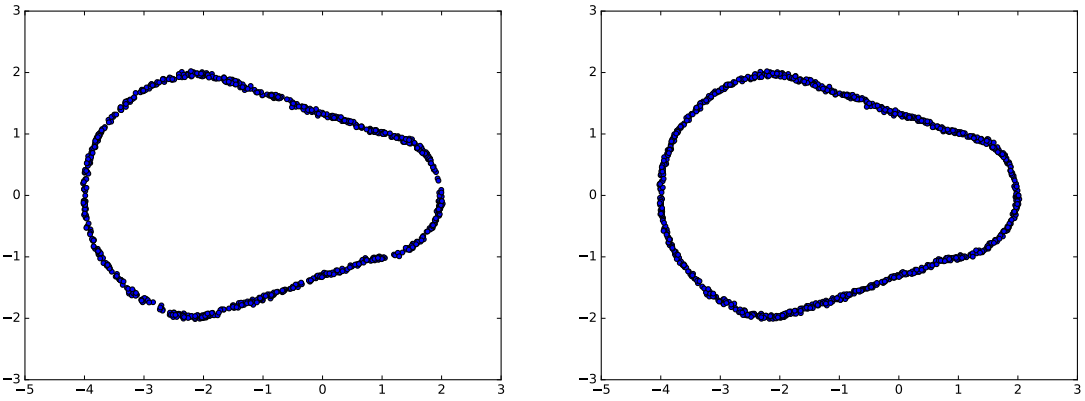
$$E_G(\mathcal{Y}) = \sum_{y \in \mathcal{Y}} d_{\mathrm{Cap}(r_\ell, r_r)}(y)^2$$



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.12: Geometric Error Rates for $\mathrm{Cap}(2,1)$ with $\sigma = 0.05$

As in the previous section, we show the geometric error rates $E_G(\mathcal{X}_t)$ for various degrees of noise (as determined by the value of σ). However, instead of showing all four plots for $\sigma \in \{0.1, 0.075, 0.05, 0.025\}$, we instead show the results for $\sigma = 0.1$ and $\sigma = 0.05$. First, in Figure 5.11, we see that the two methods perform remarkably similar. This holds true as well when $\sigma = 0.05$, as shown in Figure 5.12. This is because our geometric error does not capture the difference between the two gradient flows effectively. In fact, if every single point of the original point cloud was moved to the same point on $\text{Cap}(r_\ell, r_r)$, then the geometric error would be 0, even though the point cloud has collapsed to a single point. Instead, we need to investigate the topological error, which we do shortly.



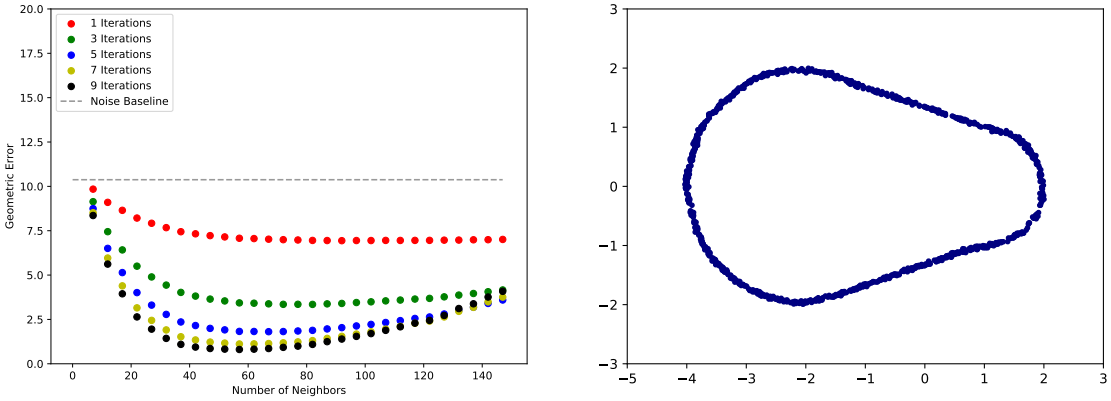
(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.13: Geometrically Optimal Point Clouds for W

The difference between the two algorithms is better revealed in Figure 5.13 which shows the optimal point clouds under the geometric error function $E_G(\mathcal{Y})$. From this figure, it is clear that the k-nearest neighbors flow contains gaps along $\text{Cap}(r_\ell, r_r)$, whereas the normal bundle flow has much smaller gaps. Thus, when we consider the topological error, we expect the normal bundle flow to produce the more topologically correct point cloud.

In Figure 5.14 we show the results of applying the adaptive smoothing discussed in Section 4.3 to the normal bundle flow. In Figure 5.14a we show the geometric error rates as

we did for the non-adaptive flow. Then in Figure 5.14b, we show the geometrically optimal point cloud. This point cloud was obtained after 9 iterations with $k = 57$. Although the minimal error achieved by the adaptive flow is approximately equal to that of the k-nearest neighbor flow and the normal bundle flow, we see that the error grows much slower as we increase k past the optimal value than it does for the k-nearest neighbor and normal bundle flows. For those two flows, once the optimal value of k has been passed, the geometric error increases rapidly. This is because the adaptive flow has the ability to lower the value of k whereas the non-adaptive flows are stuck with the original value of k for the entire point cloud.



(a) Geometric Error

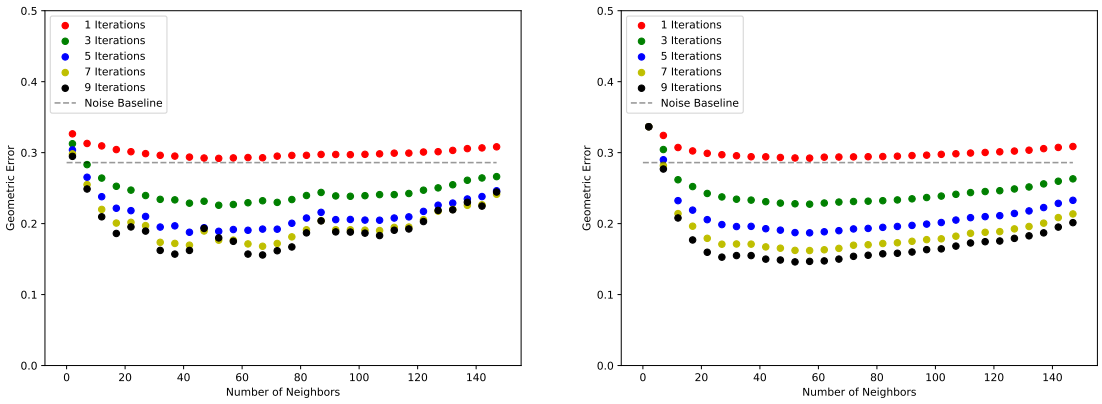
(b) Optimal Point Cloud

Figure 5.14: Adaptive Smoothing for Cap$(2, 1)$ with $\sigma = 0.1$

Topological Error

We now turn our attention to the topological error analysis for the capsule Cap(r_ℓ, r_r). Just as in Section 5.1.1, we use the persistence diagram to compute the p-dimensional topological error $E_T^p(\mathcal{Y})$ of a point cloud $\mathcal{Y}$. Notice that the topology of the capsule Cap(r_ℓ, r_r) is identical to S^1. Therefore, we can almost use the same topological error function as defined in Equation 5.1. The only difference will be that instead of a single component at $(0, 1)$ as

is the case for the one-dimensional persistence diagram of S^1, the capsule will have a single component at $(0, 2)$ in its one-dimensional persistence diagram. This is because the hold in the middle of the capsule forms immediately. However, the hole disappears when the radius r in the α-complex is $r = 2$. This is due to the left hand half circle in the capsule having radius $r_\ell = 2$.

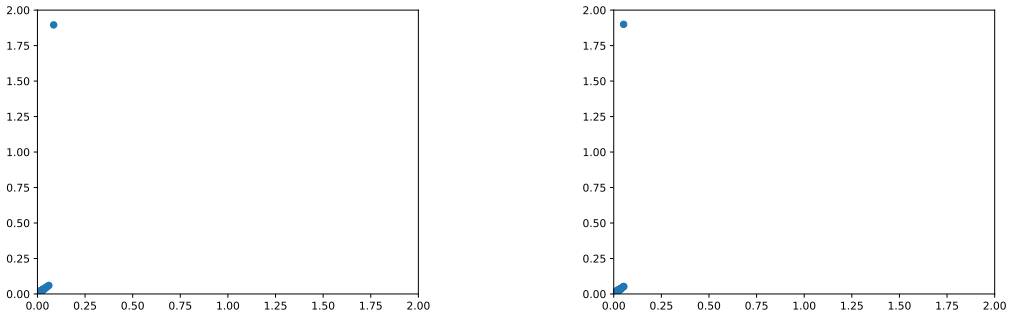


(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.15: Topological Error Rates for $\text{Cap}(r_\ell, r_r)$ with $\sigma = 0.1$

Once again we see that the minimal topological error rate was achieved under the normal bundle flow. Furthermore, we also see that the k-nearest neighbor flow is less topologically stable than the normal bundle flow as we increase the value of k. From this figure, it is clear that the topologically optimal point cloud is achieved for the k-nearest neighbor flow at around $k = 35$ or $k = 65$ and with $t = 9$. Of course, it is hard to tell since there is so much variability in the topological error across values of k. On the other hand, the minimal topological error is achieved for the normal bundle flow around $k = 50$ and $t = 9$. Another interesting observation is that initially, i.e. after a single iteration, both of these algorithms introduce some topological error as indicated by the red points. Thus, it is clear that running the smoothing for more than a single iteration is required for this point cloud.

In Figure 5.16, we show the dimension one persistence diagrams associated with the topologically optimal point clouds produced by the k-nearest neighbor flow and the normal



(a) k-NN Flow

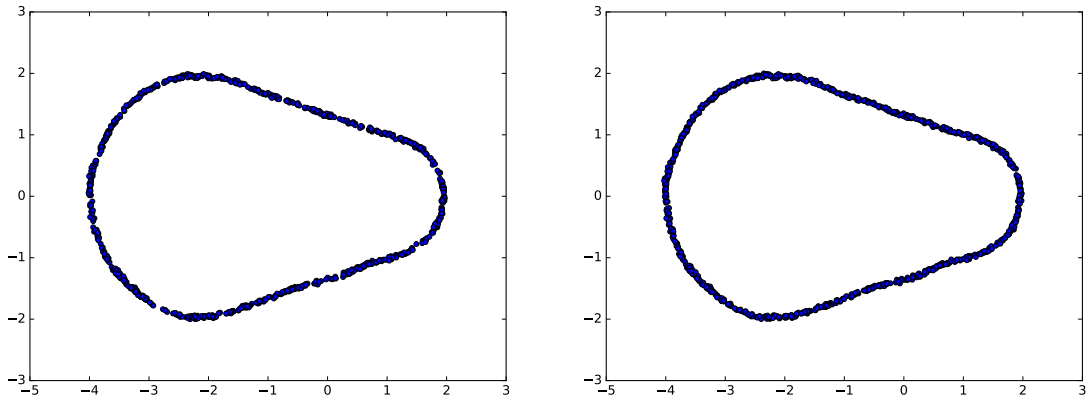
(b) Normal Bundle Flow

Figure 5.16: Optimal Persistence Diagrams for $\mathrm{Cap}(r_\ell, r_r)$ with $\sigma = 0.1$

bundle flow. From these persistence diagrams, we can see that the single hole in $\mathrm{Cap}(r_\ell, r_r)$ is formed sooner under the normal bundle flow than under the k-nearest neighbor flow. This can be seen by considering the most persistent component in each diagram (i.e. the point at the top left). The distance of this point from the vertical axis tells us when this component was formed. Since this component corresponds to the one dimensional homology group, this point tells us when the hold was formed. Since the most persistent component under the normal bundle flow is closer to the vertical axis, we know the hold formed sooner under this flow than under the k-nearest neighbor flow. This is unsurprising given the results in Figure 5.17, which shows the topologically optimal point clouds under both flows. Since the gaps are bigger for the cloud produced by the k-nearest neighbor flow, the radius in the α-complex will have to grow larger before the complex forms the large hole present in $\mathrm{Cap}(r_\ell, r_r)$. Thus, the normal bundle flow outperforms the k-nearest neighbor flow when considering the topology of the resulting point clouds.

5.1.3 Circle in $\mathbb{R}^3$

Having considered two one-dimensional manifolds embedded in $\mathbb{R}^2$, we now look at a one-dimensional manifold embedding in $\mathbb{R}^3$. This allows us to not only test a point cloud in $\mathbb{R}^3$, but more importantly to test a manifold whose codimension is greater than one. To make



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.17: Topologically Optimal Point Clouds for $\mathrm{Cap}(r_\ell, r_r)$ with $\sigma = 0.1$

the comparison simple, we will investigate how the gradient flows perform when used to smooth a point cloud drawn from the unit circle S^1 embedded in $\mathbb{R}^3$. We follow a similar sampling procedure as in Section 5.1.1 but instead of using a two dimensional Gaussian error term, we use a three dimensional Gaussian error. An example point cloud generated by this sampling procedure is shown in Figure 5.18.

Geometric Error

Although we are using the unit circle S^1 as in Section 5.1.1, we cannot use the same geometric error function. This is due to the fact that in $\mathbb{R}^3$, the geometric error function in Section 5.1.1 acts as the geometric error function of the unit sphere. Therefore, we must instead derive the correct error function. Of course, we can parameterize S^1 in $\mathbb{R}^3$ via $S(\theta) = (\cos\theta, \sin\theta, 0)$. Thus, the squared distance from a point $(x, y, z) \in \mathbb{R}^3$ to $S(\theta)$ is given by

$$D(\theta) = (x - \cos\theta)^2 + (y - \sin\theta)^2 + z^2$$

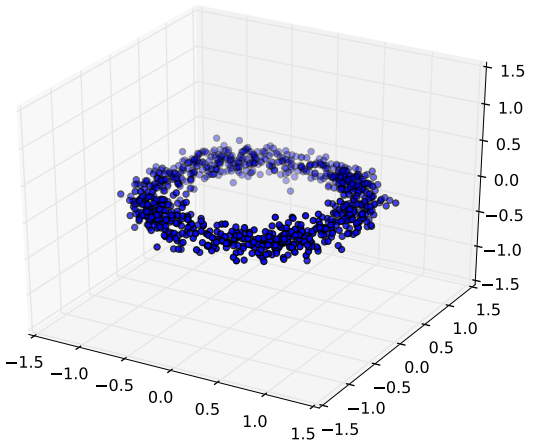


Figure 5.18: Sample point cloud drawn from S^1 in $\mathbb{R}^3$

and so taking the derivative of this function, we have

$$\frac{dD}{d\theta} = 2(x - \cos\theta)\sin\theta - 2(y - \sin\theta)\cos\theta$$

Setting this equal to zero and searching for the critical points, we find

$$2x\sin\theta = 2y\cos\theta$$

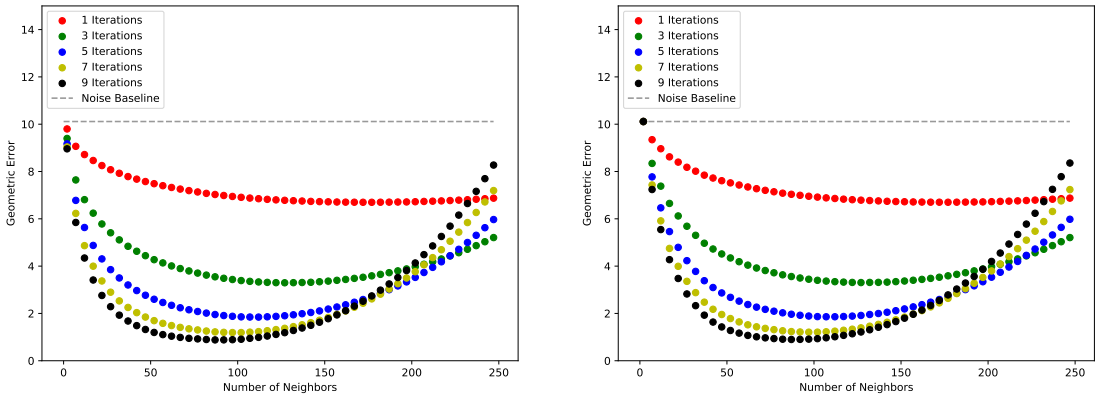
which tells us that

$$\frac{y}{x} = \tan\theta$$

Therefore, the point $S(\theta)$ which is closest to (x, y, z) is given by the point in S^1 closest to (x, y) when considered in $\mathbb{R}^2$. Thus, we see that the geometric error function $E_G(\mathcal{Y})$ should be defined

$$E_G(\mathcal{Y}) = \sum_{(x,y,z)\in\mathcal{Y}} (1 - \|(x, y, 0)\|)^2 + \|z\|^2$$

The geometric error rates for a point cloud sampled from S^1 in $\mathbb{R}^3$ with $\sigma = 0.1$ are shown



(a) k-NN Flow (b) Normal Bundle Flow

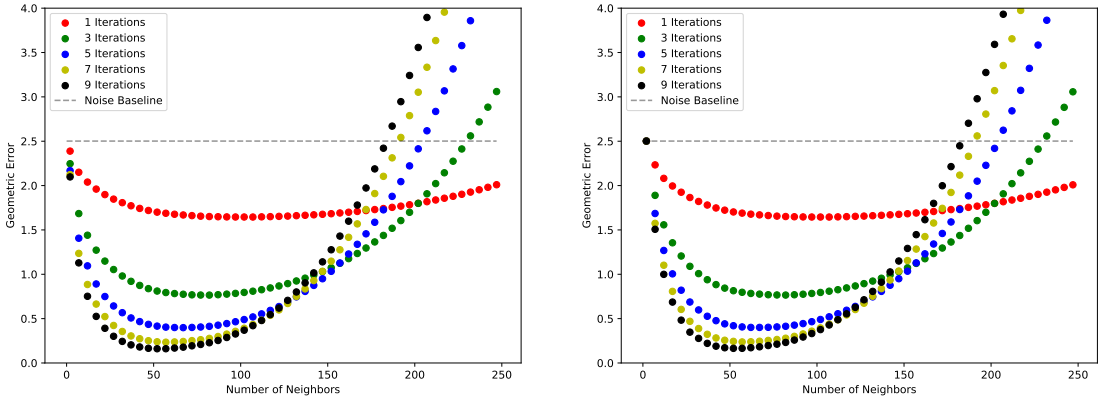
Figure 5.19: Geometric Error Rates for S^1 in $\mathbb{R}^3$ with $\sigma = 0.1$

in Figure 5.19. Here, we see that the two gradient flows perform similarly as in the previous examples. Both the flows achieve optimal geometric error around $k = 75$. Furthermore, both flows achieve the best error after nine iterations of the gradient flow. We show the geometric error rates for point clouds sampled from S^1 in $\mathbb{R}^3$ with $\sigma = 0.05$ in Figure 5.20. Notice that the number of neighbors required to achieve the optimal error rates has decreased for both gradient flows. Now the optimal value of k is closer to $k = 50$. On the other hand, the optimal number of iterations is the same as for $\sigma = 0.1$.

We show the geometrically optimal point clouds for both flows in Figure 5.21. As in the previous examples, the k-nearest neighbor flow exhibits some visual gaps, which lead to greater topological error. On the other hand, the point cloud produced under the normal bundle flow has a much more uniform distribution of points around the circle.

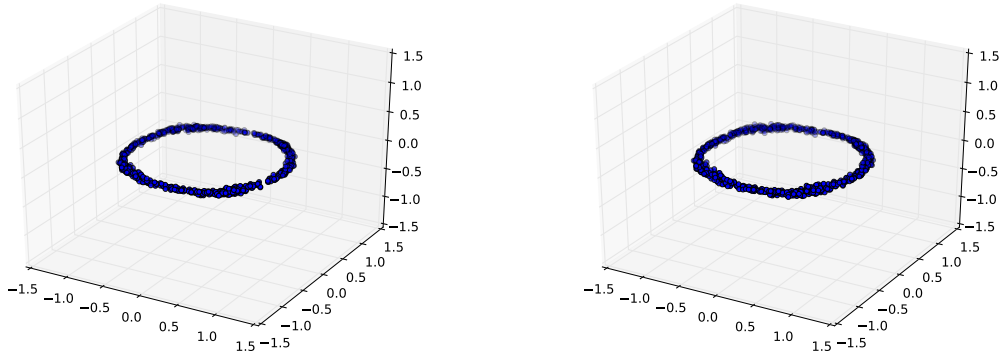
Topological Error

For the topological error estimates, we are still interested in the one-dimensional persistence diagram. This is due to the fact that the circle S^1 has a single hole which we would like to



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.20: Geometric Error Rates for S^1 in $\mathbb{R}^3$ with $\sigma = 0.05$

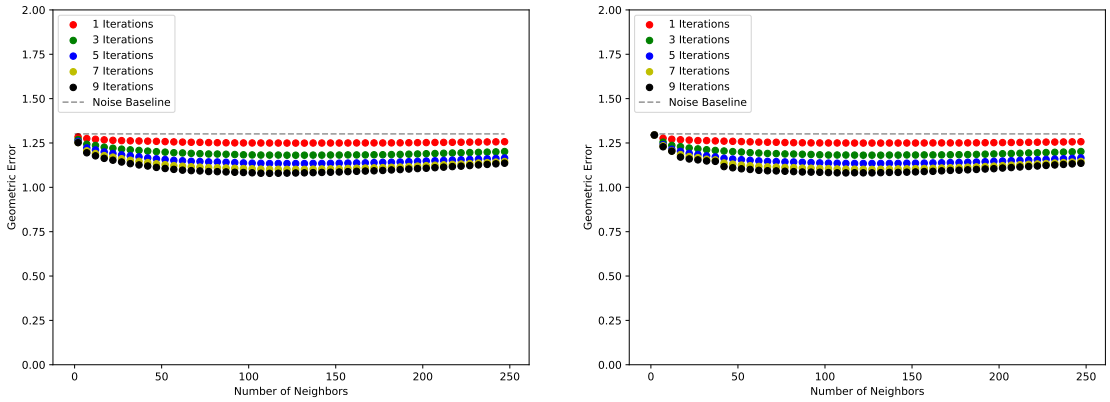


(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.21: Geometrically Optimal Point Clouds for S^1 in $\mathbb{R}^3$ with $\sigma = 0.1$

pick up from the persistence diagrams. Thus, our topological error function $E_T^1(\mathcal{Y})$ remains the same as Section 5.1.1. We show the topological error rates in Figure 5.22. Unlike the previous examples, the k-nearest neighbor flow is quite stable with increasing values of k. Additionally, we see that both the gradient flow systems exhibit optimal performance around $k = 100$. Although hard to see, the normal bundle flow achieved a lower minimum

topological error, just as in all the previous examples We can view the topologically optimal



(a) k-NN Flow

(b) Normal Bundle Flow

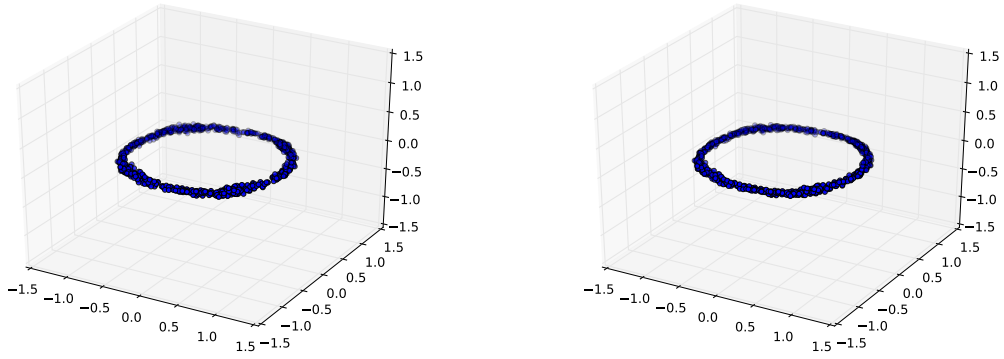
Figure 5.22: Topological Error for Point Clouds Drawn from S^1 in $\mathbb{R}^3$ with $\sigma = 0.1$

point clouds in Figure 5.23 and their associated persistence diagrams in Figure 5.24. We can see that the topologically optimal point cloud for the k-nearest neighbor flow has some regions where there is a lower density of points while the normal bundle flow point cloud appears uniformly dense. This results in the persistence diagram of the normal bundle flow, shown in Figure 5.24b, is closer to the ground truth persistence diagram (i.e. a single component at $(0, 1)$).

5.1.4 Sphere

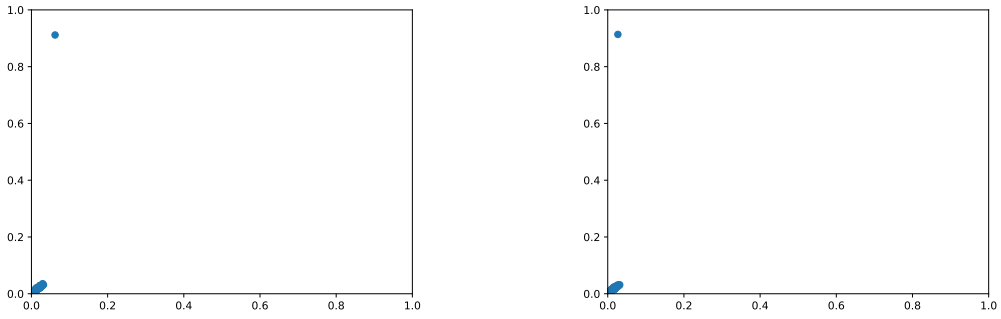
In our final simple geometry example, we analyze point clouds drawn noisily from the unit sphere S^2. This will test the algorithm's ability to operate in three dimensions with a codimension one sampled manifold. To draw samples from S^2, we follow the simple method of randomly sampling three numbers (x, y, z) from the standard Gaussian distribution, that is

$$x \sim \mathcal{N}(0, 1), \quad y \sim \mathcal{N}(0, 1), \quad z \sim \mathcal{N}(0, 1)$$



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.23: Topologically Optimal Point Clouds for S^1 in $\mathbb{R}^3$ with $\sigma = 0.1$



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.24: Optimal Persistence Diagrams for S^1 in $\mathbb{R}^3$ with $\sigma = 0.1$

We then set

$$z' = \frac{(x, y, z)}{\|(x, y, z)\|}$$

Thus, since the point z' has unit norm, it lies on the sphere S^2. We then add noise to the point by setting

$$z = rz'$$

where $r \sim \mathcal{N}(1, \sigma)$. This has the effect of adding noise in the radial direction. We drew 1000 samples following this procedure to create a noisy point cloud sampled from S^2. An example point cloud drawn from this process is shown in Figure 5.25.

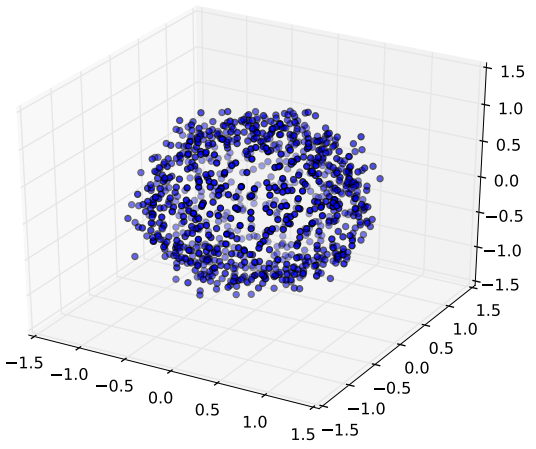


Figure 5.25: Example Point Cloud Drawn from S^2

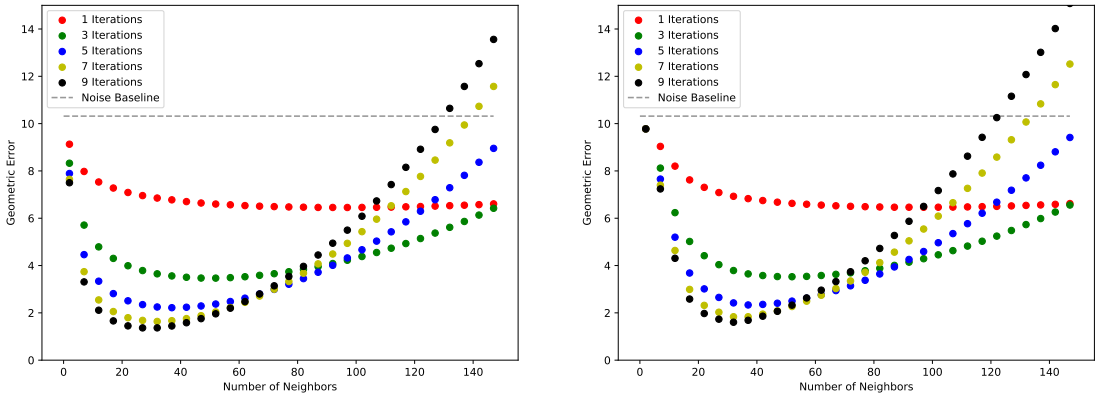
Geometric Error

The geometric error function for this example is identical to the geometric error function used in Section 5.1.1 except now we take the Euclidean norm in $\mathbb{R}^3$ instead of $\mathbb{R}^2$. In particular, the geometric error function $E_G(\mathcal{Y})$ is given by

$$E_G(\mathcal{Y}) = \sum_{y \in \mathcal{Y}} (1 - \|y\|)^2$$

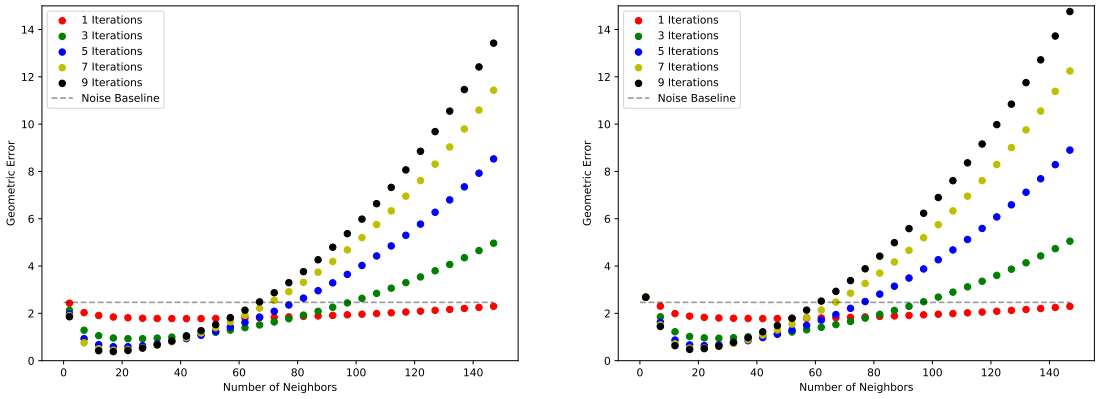
The error rates associated with different values of k are shown in Figure 5.26 for $\sigma = 0.1$ and in Figure 5.27 for $\sigma = 0.05$. Similar to all previous examples, the geometric error plots are quite similar between the two algorithms. Additionally, the optimal value of k once again decreases with a lower value of σ. This is to be expected since a point cloud with

lower initial noise will require less smoothing.



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.26: Geometric Error Rates for Point Clouds for S^2 with $\sigma = 0.1$

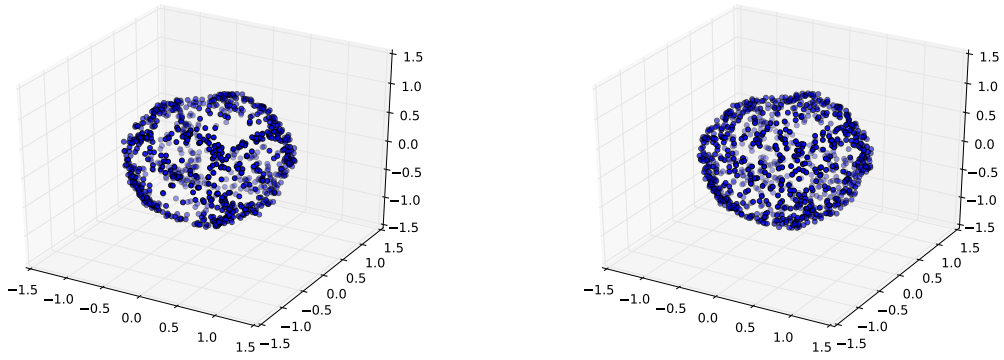


(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.27: Geometric Error Rates for Point Clouds for S^2 with $\sigma = 0.05$

The geometrically optimal point clouds for S^2 are shown in Figure 5.28. In the k-nearest neighbors case (the left hand figure), the clustering tendency of the k-nearest neighbors gradient flow is plainly evident. Although this does not cause a problem for our geometric error function, this particular point cloud would score quite poorly under the topological

error function. On the other hand, the optimal point cloud under the normal bundle flow is much more uniformly distributed across the surface of the sphere. This results in fewer and smaller gaps.



(a) k-NN Flow (b) Normal Bundle Flow

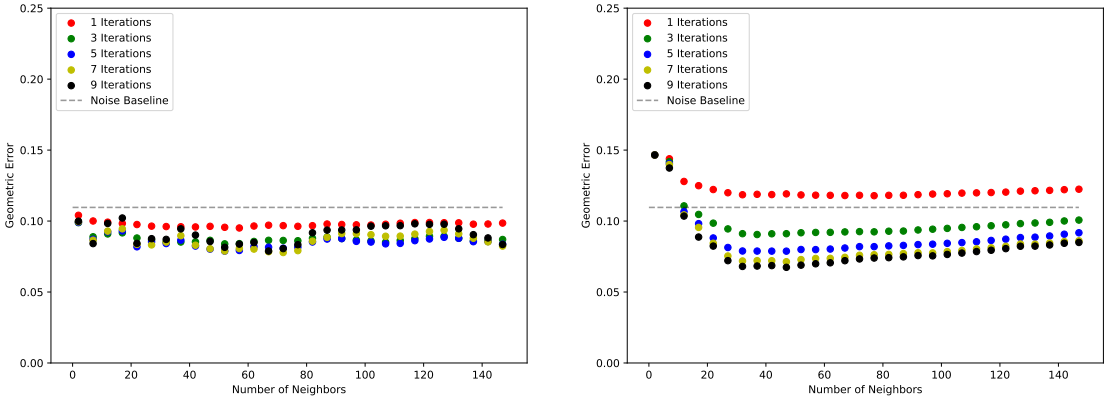
Figure 5.28: Geometrically Optimal Point Clouds for S^2 with $\sigma = 0.1$

Topological Error

Now that we are working with a two-dimensional manifold, our topological considerations will be slightly different. Instead of working with the one-dimensional persistence diagrams, we will instead use the two-dimensional persistence diagrams. Thus, instead of looking for holes in the point cloud as was the case for the circle, capsule, and unit circle in $\mathbb{R}^3$, we will be looking for three-dimensional *voids*. In the case of the unit sphere, there is a single void: the interior of the sphere. For the unit sphere, this void is born at 0 and dies at 1 (since the radius of the unit sphere is 1). Therefore, the optimal two-dimensional persistence diagram will have a single component at $(0, 1)$. Here we use the topological error function $E^2_T(\mathcal{Y})$ given by

$$E^2_T(\mathcal{Y}) = d_B(\mathrm{Dgm}_2(S^2), \mathrm{Dgm}_2(\mathcal{Y}))$$

In Figure 5.29, we show the topological error $E^2_T(\mathcal{Y})$ for point clouds $\mathcal{Y}$ produced by

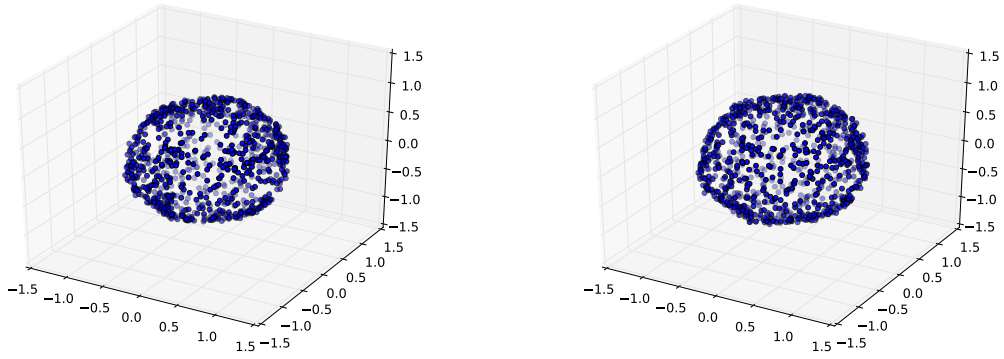


(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.29: Topological Error for Point Clouds Drawn from S^2 with $\sigma = 0.1$

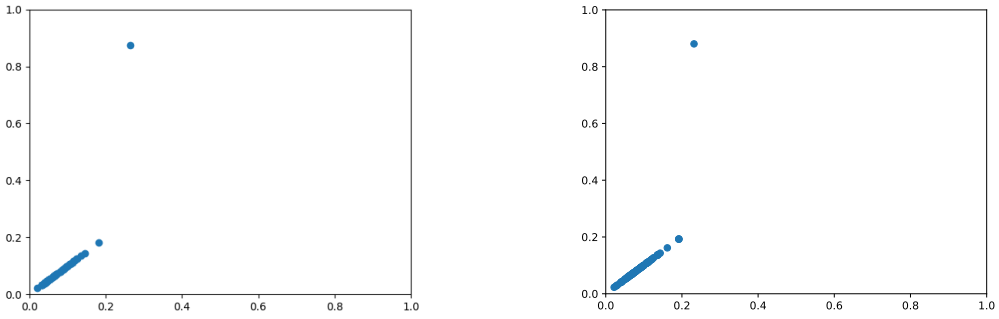
running the two gradient flows using various parameters of k and number of iterations. As we can see, the k-nearest neighbor gradient flow has a lower topological error for every set of parameters considered, however it stays relatively constant as we increase the value of k. On the other hand, the normal bundle flow has an initial dive in topological error as we increase k, and then the error begins climbing after about $k = 35$. Thus, the optimal value of k for the normal bundle flow is $k = 35$. Additionally, the optimal number of iterations for the normal bundle flow in this case is $t = 9$. However, the error rates begin to converge so there is not much difference in error for $t = 7$ and $t = 9$.

The topologically optimal point clouds are shown in Figure 5.30 and their associated two-dimensional persistence diagrams are shown in Figure 5.31. Here, we can see that the two optimal point clouds both exhibit some clustering, but overall are much more uniformly dense than geometrically optimal point clouds. This is what causes these point clouds to exhibit lower topological error. In the persistence diagrams, we can see the single component which corresponds to the interior of the sphere. The most persistent component of the k-nearest neighbor point cloud was formed at $r = 0.263$ and dies at $r = 0.876$ while the most persistent component of the normal bundle point cloud was formed at $r = 0.232$ and dies at $r = 0.880$. Thus, we see that the topologically optimal point cloud obtained under the



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.30: Topological Optimal Point Clouds for S^2 with $\sigma = 0.1$



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.31: Topological Optimal Point Clouds for S^2 with $\sigma = 0.1$

normal bundle flow achieves a lower topological error than that of the k-nearest neighbor flow.

5.1.5 Five-dimensional Sphere

All the of examples we have presented thusfar concern point clouds in two or three dimensions. Since our algorithms were designed to operate on arbitrary dimensions, we now show the results of applying the gradient flow algorithms to a five dimensional sphere embedded

in $\mathbb{R}^6$, i.e. the sphere S^5. Since we are working in $\mathbb{R}^6$, the sampled manifold is codimension one. Furthermore, since we have increased the dimensionality, we increase the number of points in the point cloud as well to avoid issues with sampling density. In particular, we drew a point cloud of 10,000 points from S^5. Since we are using S^5, the geometric error function $E_G(\mathcal{Y})$ is simply

$$E_G(\mathcal{Y}) = \sum_{y \in \mathcal{Y}} (1 - \|y\|)^2$$

as before.

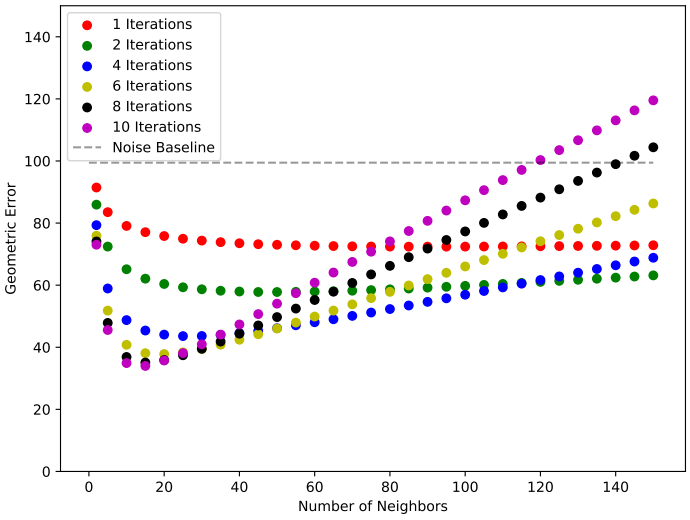


Figure 5.32: Geometric error for a point cloud drawn from S^5 under the k-nearest neighbors flow

The results of running the flow are shown in Figures 5.32 and 5.33. From these images, we see that the gradient flows were able to drastically reduce the original geometric error, despite the problem being posed in $\mathbb{R}^6$. Thus, we see that the algorithms still work as expected for higher dimensions. Furthermore, we see that the optimal parameters differ for the k-nearest neighbor flow and the normal bundle flow. In particular, the k-nearest neighbors flow performs best with $k = 15$ and 10 iterations. On the other hand, the normal

bundle flow reaches minimum geometric error when $k = 30$ and the number of iterations is set to 8. Furthermore, the k-nearest neighbor flow actually achieves a lower geometric error, unlike all the previous examples. The smallest error obtained by the k-nearest neighbors flow is $E_G(\mathcal{Y}) = 33.99$ while the minimum error obtained by the normal bundle flow is $E_G(\mathcal{Y}) = 42.61$. Likely, the k-nearest neighbor flow outperformed the normal bundle flow because the normal and tangent space approximations were less effective for this point cloud. Although we have increased our dimension by a factor of two, we have only used $10,000$ points to form this point cloud. When the dimensionality of the ambient space doubles, we must square our sample size to maintain the same sampling density. Therefore, to achieve comparable performance, we would likely need to use $1,000,000$ points. Due to the considerable computational cost of running a parameter sweep on such a large point cloud, we leave this as speculation. Of course, both these error rates are still far lower than the original error, $E_G(\mathcal{X}) = 99.46$. Therefore, we see that both methods improve the geometric accuracy of the sampled point clouds.

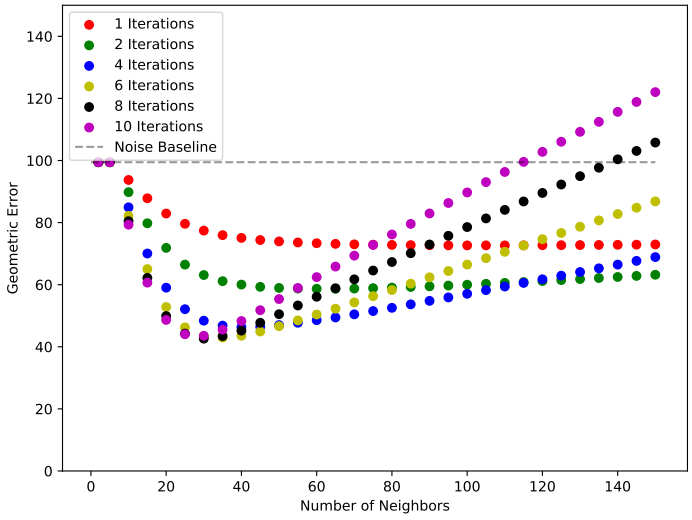


Figure 5.33: Geometric error for point cloud drawn from S^5 under the normal bundle flow

From these results, we see that the gradient flow algorithms can help in higher dimensions. As for computational cost, let $\mathcal{X}_2$ and $\mathcal{X}_5$ be two point clouds drawn from S^2 and S^5, respectively. Let $|\mathcal{X}_2| = |\mathcal{X}_5| = 10000$ and let $k = 1000$. Then a single iteration of the k-nearest neighbor gradient flow for $\mathcal{X}_2$ takes approximately 5.706 seconds while a single iteration of the gradient flow for $\mathcal{X}_5$ takes approximately 8.910 seconds. For the normal bundle flow, a single iteration takes 6.054 seconds for $\mathcal{X}_2$ and 9.518 seconds for $\mathcal{X}_5$. As expected, the high dimensional point cloud takes longer to process for the same point cloud size and number of neighbors. However, the extra expense is not large given the ability to smooth these higher dimensional point clouds. Furthermore, although the normal bundle flows took slightly longer than their corresponding k-nearest neighbor flows, the extra incurred cost going from $\mathbb{R}^3$ to $\mathbb{R}^6$ was about the same in both cases. Furthermore, the computational cost is dominated by the size of the point cloud and the value of k. These factors drive the most expensive part of the algorithms, the determination of the k-nearest neighbors for each point in $\mathcal{X}$.

5.2 3D Scanning and LiDAR

Now that we have applied the k-nearest neighbor flow and the normal bundle flow to some simple geometries and found that the flows lower both the geometric and topological noise in the point cloud, we turn to applying the two flows on richer and more complicated geometries. In particular, we will apply the smoothing algorithms to point clouds obtained from 3D scanning and from LIDAR. We will show that the smoothing algorithms once again succeed in reducing the noise in these more complex point clouds, just as we saw on the simple geometries in the previous section. Unlike the previous section, the point clouds we analyze in this section reflect a more realistic use of the smoothing algorithms since both 3d scanning and LIDAR are in common use in modern technology.

5.2.1 Stanford Bunny

We begin our analysis using a point cloud sampled from the classic Stanford Bunny model. The Bunny was created using by Greg Turk and Marc Levoy in [53] while at Stanford University in 1994. To create the Bunny, Turk and Levoy developed a technique for reconstructing surfaces from multiple *range images*. A range image is produced by a range scanner, which produces a 2d array of distance values describing the distance from the scanner to the nearest surface. Since most objects self occulde, the Stanford Bunny included, Turk and Levoy had to develop a method for combining multiple range images of a single object and reconstructing the surface of the object. Their technique essentially consisted of three steps: align the meshes, zipper together the adjacent meshes, and finally compute a locally weighted average of the surfaces on the overlapping regions. Their algorithm helped open the door to rich 3d scans of entire objects.

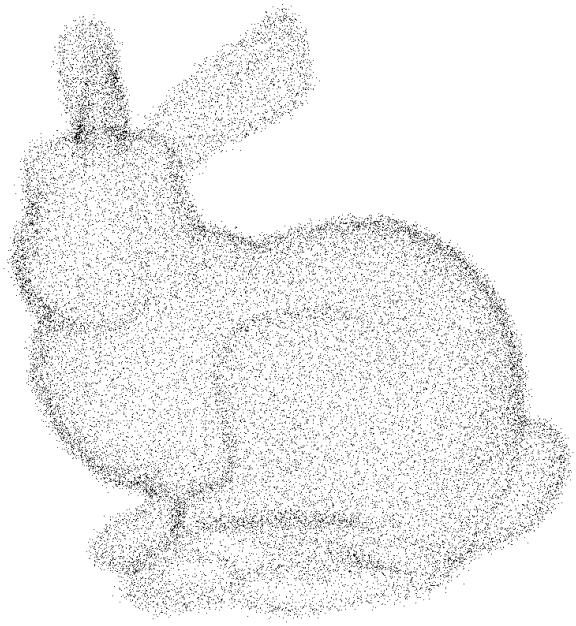


Figure 5.34: Noisy Stanford Bunny

In the paper describing their surface reconstruction technique (see [53]), Turk and Levoy provided several examples to demonstrate the quality of the algorithm. However, the Stanford Bunny easily became one of the most famous of these examples. The Stanford Bunny has a rich history of use in testing various 3d modeling algorithms. Here, we follow in the footsteps of many before us and analyze the performance of the smoothing algorithms on the Stanford Bunny. However, we also heed the advise Greg Turk provides on his website [3] and will evaluate the algorithm on more complicated shapes later in this section.

For our evaluation using the Stanford Bunny, we will take a point cloud sampled from the surface of the Bunny and add artificial noise to the point cloud. Following this approach will allow us to measure the geometric error of the point cloud precisely since we know the surface from which the point cloud was sampled. Additionally, since we can control the level of noise, we can evaluate the performance of the smoothing algorithms across a range of noise parameters. Therefore, we will be able to guage how robust the algorithm is to varying levels of noise. This evaluation approach will contrast with the approach we must take later to measure the noise in LIDAR data. With LIDAR data, we have no known ground truth and so we cannot precisely measure the reduction in the geometric and topological error after the smoothing algorithms have been applied.

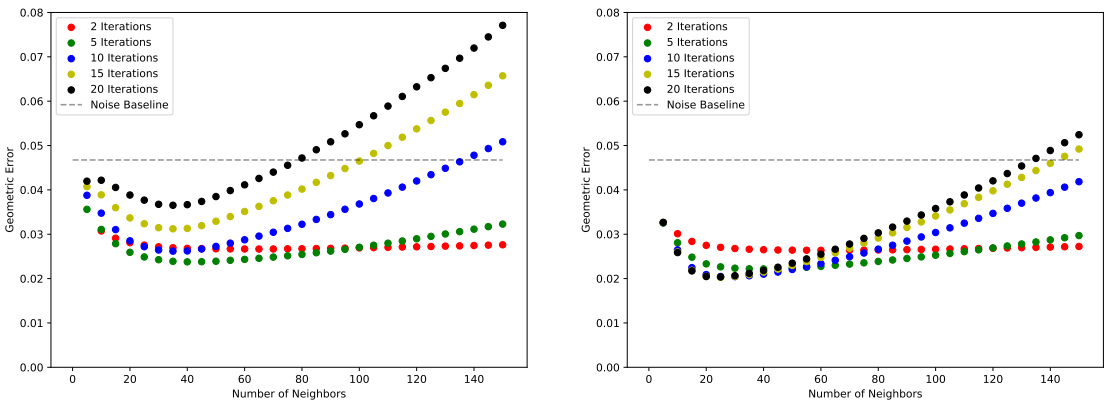
The version of the Stanford Bunny we use is a point cloud $\mathcal{X}$ of 35,947 points. The point cloud is contained in a rectangular cuboid R which measures approximately

$$R = (-0.0972, 0.0631) \times (0.0304, 0.1889) \times (-0.0633, 0.0607)$$

Given these dimensions, we add a randomly sampled 3-dimensional Gaussian distribution of mean 0 and standard deviation $\sigma = 0.001$ to each point of $\mathcal{X}$, thereby producing a noisy point cloud $\mathcal{X}_\sigma$. We then apply the k-nearest neighbor flow of Chapter 3 and the normal bundle flow of Section 4.1 for varying values of k and for a varying number of iterations. For a given value of k and a given number of iterations t, we then measure the geometric error $L(\mathcal{X}, \mathcal{Y})$ of the resulting smoothed point cloud $\mathcal{Y}$ using the following equation

$$L(\mathcal{X}, \mathcal{Y}) = \sum_{y \in \mathcal{Y}} \min_{x \in \mathcal{X}} \|x - y\|^2 \tag{5.2}$$

Note that the above equation approximates the exact geometric error of each point which would be measured by taking the distance to the manifold. However, since the points of $\mathcal{X}$ all lie on the surface of the underlying manifold and provide a thorough sampling of the manifold, the point cloud based error function L provides an accurate approximation of the true geometric error. This error function also benefits from not relying on knowledge of the underlying manifold. This will be important when we investigate the algorithms' performance LiDAR data, where there is no underlying manifold to compare against. For this reason, we have chosen to compute all the geometric errors using the function L, thereby providing consistency for the three experiments presented in this section.



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.35: Geometric Error for Stanford Bunny

The geometric error rates for the two algorithms are presented in Figure 5.35. Here, we show the value of L obtained by setting $k = 5i$ where $i \in \mathbb{N}$ with $2 \leq i \leq 30$. We reported the error rate after t iterations, where t is set to be $2, 5, 10, 15$, and finally 20. The *noise baseline* is also reported. This is the value of L computed directly on the pair $\mathcal{X}$ and $\mathcal{X}_\sigma$

(i.e. the initial error of the noisy point cloud).

Some immediate conclusions can be drawn from these two figures. In the k-nearest neighbor flow in Figure 5.35(a), it is clear that the optimal value of k lies somewhere around $k = 35$. Additionally, the optimal number of iterations to run the algorithm was $t = 5$. On the other hand, for the normal bundle flow in Figure 5.35(b), the optimal values of k and t were closer to $k = 20$ and $t = 20$. Therefore, the normal bundle flow in this experiment required fewer neighbors but more iterations than the k-nearest neighbor flow.

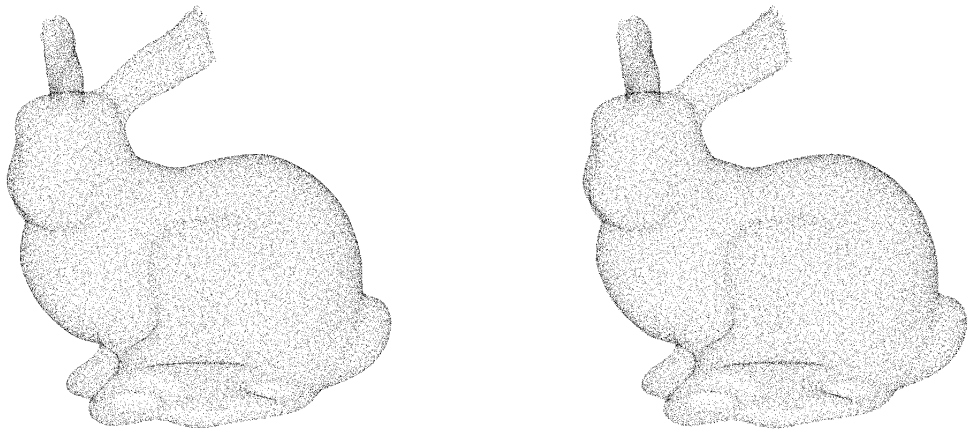
It is also clear that the overall error obtained under the optimal parameters is lower for the normal bundle flow than for the k-nearest neighbor flow. Furthermore, the error initially drops more rapidly for the normal bundle flow than for the k-nearest neighbor flow and it also grows slower once the optimal parameters have been passed.

It is worth noting that the increasing error both algorithms experience after their optimal parameters have been passed is due to volumetric shrinking of the point clouds. That is, when the point clouds are smoothed too much, their volumes begin to shrink. Thus, while the shape of the point cloud may still be an accurate representation of the orignial point cloud, it's volume has shrunken enough that the smoothed point cloud is much smaller than the original point cloud, hence the increasing error. A geometric explanation for this volumetric reduction can be see by considering the convexity of the point cloud. Since the bunny contains many locally convex regions, if the chosen nearest neighbors around a point exhibit this convexity, their barycenter will lie within the bunny. Hence, when smoothing, the points will move closer to the center of the bunny and reduce the overall volume of the point cloud.

Finally, notice that while the error rates for the k-nearest neighbor flow with $t = 5, 10, 15$ and 20 never cross each other as k increases, the error rates for $t = 2$ cross both the $t = 5$ and $t = 10$ error rates. This makes sense since the initial iterations of the flow produce the biggest decrease in error. Then, once the optimal value of k is passed (i.e. around $k = 35$), the volume begins to shrink, an effect felt less when there are only two iterations. Also, notice that the error rates produced by $t = 20$ are the worst for every value of k. This

indicates the the k-nearest neighbor smoothing should be run for far fewer iterations. In fact, other than for $t = 2$, the error rates decrease at every value of k for decreasing values of t.

Similar crossing can be seen in the normal bundle flow error shown in Figure 5.35(b). However, in this case, the error rates for $t = 2$ are the worst of the five reported values when k is small. In fact, at the optimal value of k (i.e. around $k = 20$), the error rates exhibited for $t = 2$ are the highest. Only after we increase k to be greater than 120 do we see $t = 2$ produce the best error rates. In contrast with the k-nearest neighbor flow, we see setting $t = 20$ produces the best error rates. Thus, the normal bundle flow seems to require more iterations than the k-nearest neighbor flow to produce optimal results. Although, note that when k is around 20, the error rates reported for $t = 10, 15$, and 20 are all approximately the same. Thus, we have seem some convergence around this value of k. For larger values of k, the three values of t do not produce convergent error rates.



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.36: Optimal Smoothing for the Stanford Bunny

5.2.2 Stanford Dragon

For our next experiment, we analyze the performance of the k-nearest neighbors flow and the normal bundle flow when applied to a noisy version of the Stanford Dragon point cloud. The Stanford Dragon is a point cloud created by the Stanford University Computer Graphics Laboratory using the Cyberware 3030 MS scanner. The point cloud first appeared in [19]. To obtain the point cloud, approximately seventy scans were taken of the dragon, producing $437,645$ points.

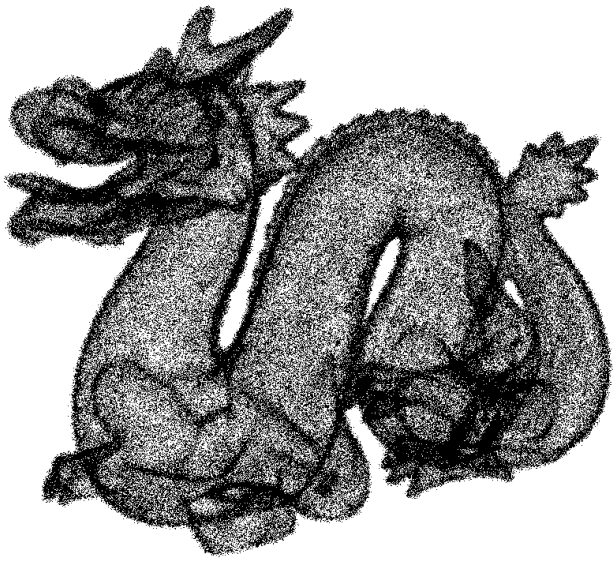


Figure 5.37: Noisy Stanford Dragon

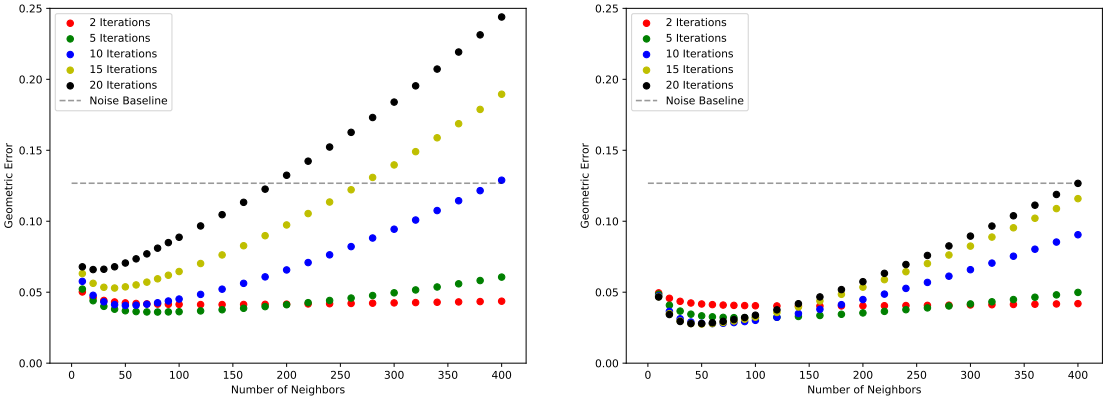
Similar to the Stanford Bunny in Section 5.2.1, the points lie directly on the sampled manifold. Therefore, to analyze the smoothing quality of our algorithms, we will artificially add noise to the point cloud. This allows us to precisely measure the geometric error of the resulting point cloud. As in the previous section, we will use notation $\mathcal{X}$ to represent the original point cloud and $\mathcal{X}_\sigma$ to represent the noisy point cloud. Additionally, we once again

use the geometric error function L, given by Equation 5.2.

The Stanford Dragon is contained in a rectangular cuboid R of dimensions

$$R = (-0.109, 0.097) \times (0.0527, 0.198) \times (-0.050, 0.042)$$

since the size of R is different for the Stanford Bunny than for the Stanford Dragon, we will use a different standard deviation σ for the Gaussian noise. In particular, we set $\sigma = 0.0005$ since the dragon is considerably thinner than the bunny. We then perform a similar parameter sweep to the sweep we did in Section 5.2.1. Since this point cloud has an order of magnitude more points than the Stanford Bunny, we perform the sweep against larger values of k. Due to the increasing computational demands as we increase k, we sampled the parameter space more densely for lower values of k. That is, we set $k = 10i$ for $i = 1, ..., 10$ and then set $k = 20j$ for $j > 5$ We use the same iteration values as in the previous experiment.



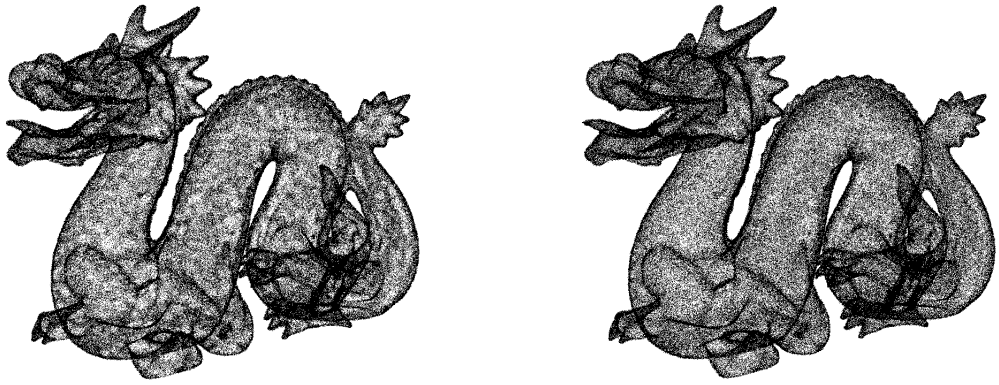
(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.38: Geometric Error for Stanford Dragon

The results of our parameter sweep are shown in Figure 5.39. We once again show the results for the k-nearest neighbor flow (Figure 5.39(a)) and the normal bundle flow (Figure 5.39(b)). We also show the value of $L(\mathcal{X}, \mathcal{X}_\sigma)$, represented by the dotted *noise baseline*.

From these figures, it is clear that once again the normal bunlde flow outperforms the k-nearest neighbor flow. Not only does the normal bundle flow reach a lower overall geometric error, the growth of its error once the optimal value of k has been passed is much slower than the growth in the$k$-nearest neighbor flow. In fact, up to $k = 400$ none of the resulting point clouds exhibit more error than in the original noisy point cloud $\mathcal{X}_\sigma$. This is in stark contrast to the k-nearest neighbor flow where the noise baseline is passed rather quickly.

Furthermore, we once again see that the $t = 20$ iteration setting performs worst in the k-nearest neighbor flow and best in the normal bundle flow. This again suggests that the k-nearest neighbor flow requires fewer iterations to achieve an optimal value. However, unlike the pervious experiment with the Stanford Bunny, the k-nearest neighbor flow does not require fewer nearest neighbors (i.e. a lower value of k) than the normal bundle flow. In this case, both algorithms reach peak performance with fewer than 50 nearest neighbors and for some values of t the normal bundle flow actually requires fewer neighors to reach peak performance.



(a) k-NN Flow (b) Normal Bundle Flow

Figure 5.39: Optimal Smoothing for the Stanford Dragon

5.2.3 LIDAR Data

Having seen the effectiveness of the various smoothing algorithms when applied to point clouds obtained via 3d scans, we turn our attention point clouds obtain from LIDAR data. LIDAR, which stands for *Light Detction and Ranging*, is a remote sensing technique used to measure the distance from a LIDAR sensor to the Earth. For an overview of LIDAR data, we will quickly summarize the relevant parts of Campbell and Wynne's *Introduction to Remote Sensing* (see [11]). LIDAR uses a transmitter to emit pulsed laser light and a light detector. To measure the distance to the ground, the difference in time between when a pulse was emitted and when the reflected pulse was detected by the detector is computed. The distance the pulse travelled can then be deduced using the speed of light.

Often these measurements are taken from an airplane or a helicopter. While flying over an area of interest, the LIDAR sensor measures and records the distance from the sensor to the ground. Simultaneously, a GPS unit in the aircraft records the position of the sensor and an inertial measurement unit (IMU) records precise information about the orientation of the aircraft, including its roll, pitch, and yaw. Since LIDAR scanners can transmit up to 300,000 pulses per second, very detailed measurements of the ground elevation can be recorded. Most LIDAR collection missions involve flying parallel strips over an area of interest and patching together the recordings at the end of the mission.

Unlike most 3d scanning techniques, LIDAR can *see through* some objects while pulses make their way to the ground. This often occurs when a pulse hits vegetation above the ground. In this case, only some of the light is reflected up to the aircraft, with the rest of the light continuing toward the ground. This results in multiple *returns* being registered for a single pulse. Constraints on the detector response time lead to something called the *dead time* during which no additional returns can be registered. When there is vegetation directly above the ground, this can lead to the ground elevation being detected as artificially high. This occurs because first the vegetation return registers and then the ground return arrives at the detector during the dead time. A consequence of this complication results

in the surface of the Earth appearing artificially rough when there is considerable vegetation ground cover. Other issues such as atmospheric effects and surfaces with complicated reflectance properties can contribute to the noise inherent in the sampling process.

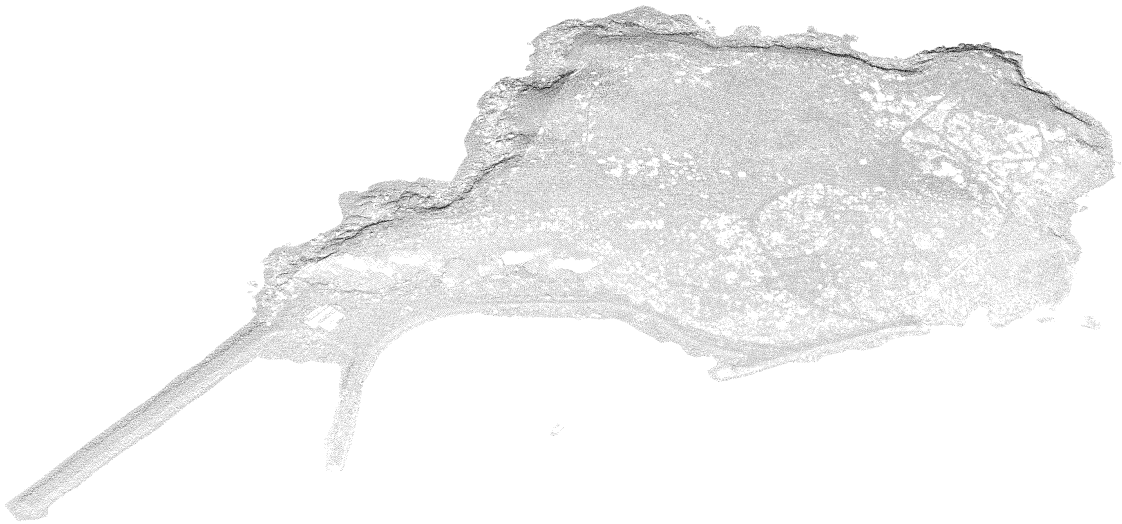


Figure 5.40: Granite Island Point Cloud

Point clouds obtained from LIDAR data are often considerably richer than those obtained from 3d scans. In addition to the bare Earth, there are often man-made objects, vegetation, and water present in the LIDAR scans. Therefore, such point clouds can provide quite a challenge. Furthering this challenge, point clouds obtained from LIDAR can be much larger than those obtained from 3d scans, often containing many millions of points. For our evaluation, we will analyze how our smoothing algorithms perform on real-world LIDAR data.

For our analysis of the gradient flow algorithms when applied to LiDAR data, we will focus on the computational expense of running these flows. We choose this focus because, unlike the previous experiments, we have no *ground truth*. Of course, the purpose of LiDAR is to measure the terrain. Thus, the closest we can get to the ground truth is the LiDAR

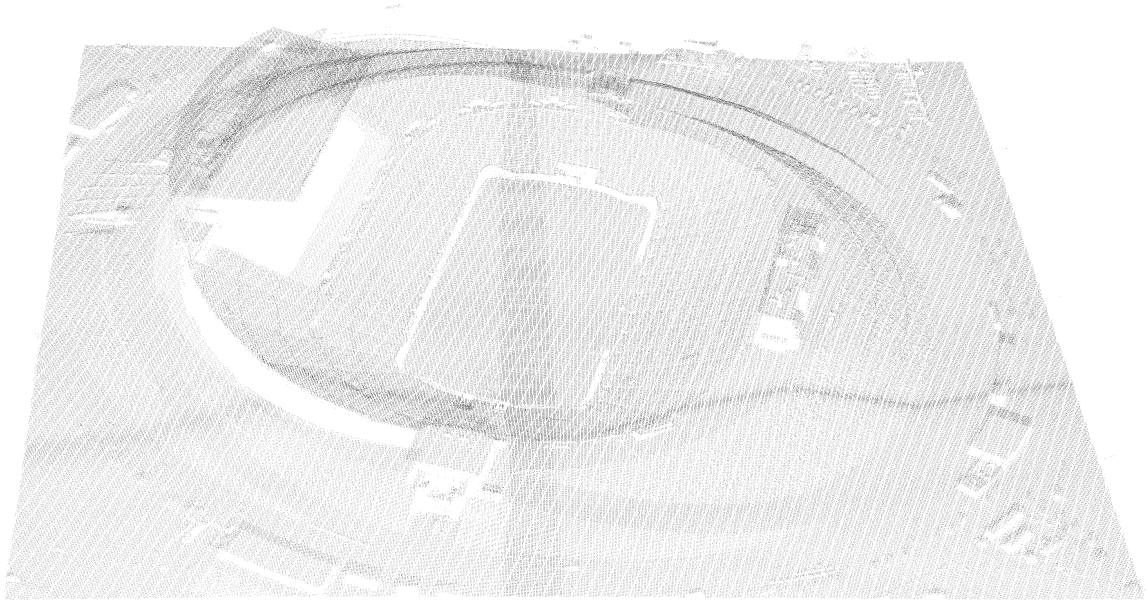


Figure 5.41: Stadium Point Cloud

itself. All we can do is smooth the point cloud, and visualize the results. We cannot measure the geometric or topological error. Therefore, we will use this section to analyze the performance of the k-nearest neighbor gradient flow and the normal bundle gradient flow.

In particular, we will run the two discretized gradient flows on three different LiDAR samples. We refer to these samples by the names Granite, Stadium, Potomac. The Granite and Potomac point clouds were obtained from the OpenTopography repository [2]. The Granite point cloud contains 391,236 points and the Potomac point cloud contains 908,216 points. The Stadium point cloud was obtained from the LibLAS samples webpage [1]. This point cloud contains 693,895 points. These three point clouds can be seen in Figures 5.40 (Granite), 5.41 (Stadium), and 5.42 (Potomac). We show the point clouds as smoothed by the normal bundle flow in these figures. However, due to the size of the point clouds, it is hard to visually notice the difference between the smoothed point clouds and the original, unsmoothed point clouds when zoomed out far enough to capture the entire point cloud.

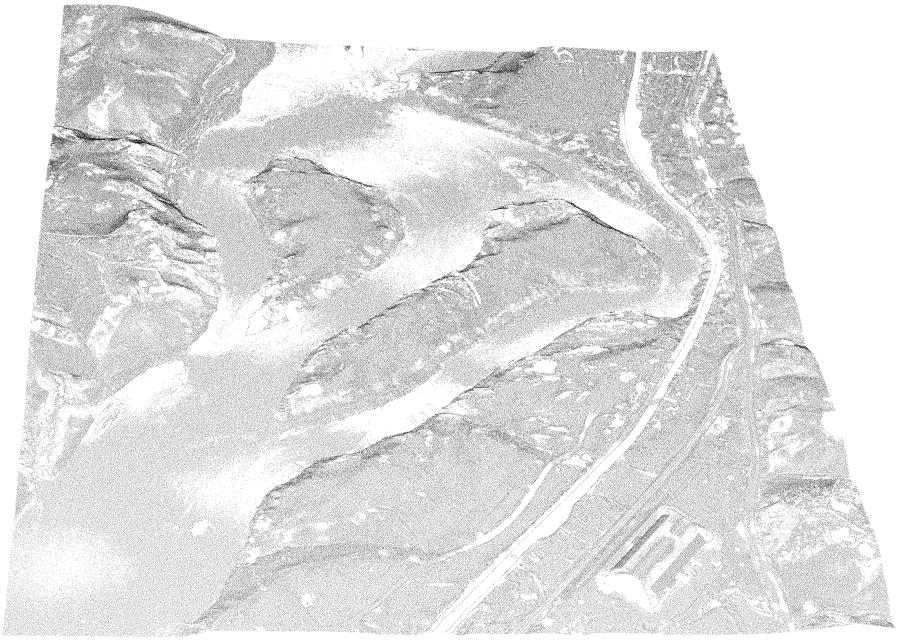


Figure 5.42: Potomac Point Cloud

Therefore, we opt to save space and only show one example of each point cloud.

Given the considerable size of each point cloud, it is not surprising that running the discretized gradient flows is computationally quite expensive. For these experiments, we ran the two gradient flows using eight threads on an Intel Core i7-7700K clocked at 4.20 GHz. Running the gradient flow in parallel was a simple modification to the gradient flow routine as each point gets updated independent of every other point. Additionally, each update only needs access to the common k-d tree, however since the update process only reads from this tree, there are no memory issues. Thus, we can partition the updates across the eight threads.

The runtime, in seconds, for a single iteration of the k-nearest neighbor gradient flow is shown in Figure 5.43. We show the runtime across several values of k to get an idea of the scalability of the gradient flow as we increase k. Clearly, this algorithm scales superlinearly. Unsurprisingly, we see that the computational cost increases as we increase the number of points in the point cloud.

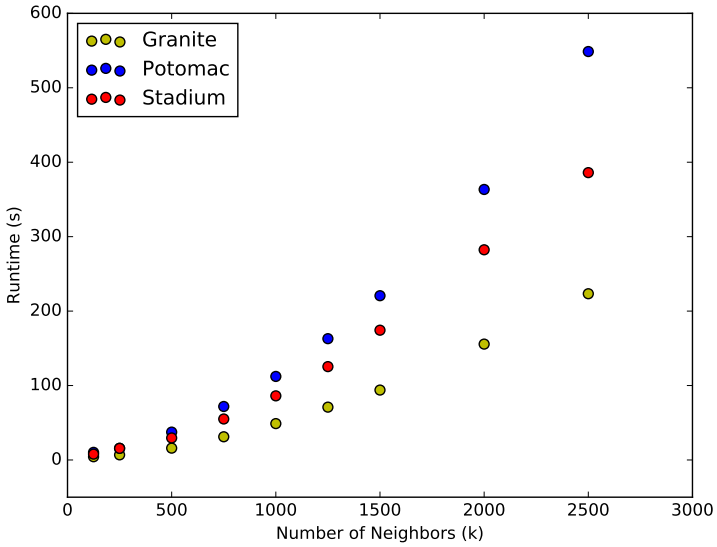


Figure 5.43: Runtime for k-nearest neighbor gradient flow

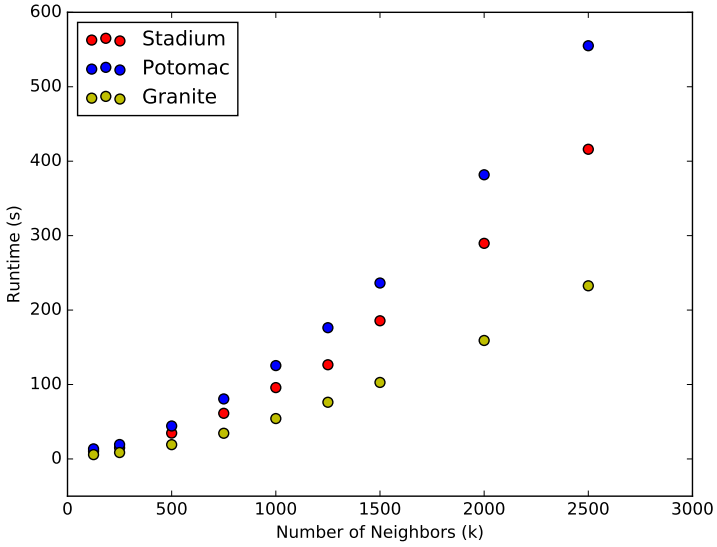


Figure 5.44: Runtime for normal bundle gradient flow

For comparison, we show the runtime, again in seconds, for a single iteration of the normal bundle flow in Figure 5.44. We see very similar scaling, although the total runtime

has increased for every value of k. This is expected since the normal bundle flow has the additional computational cost associated with computing the covariance matrix $C(u)$ defined in Equation 4.3 as well as the eigenvalue decomposition of $C(u)$. However, these operations are quite fast and therefore do not add much to the overall runtime.

It is important to keep in mind that these figures only show the runtime for a single iteration of the gradient flows. As seen in the previous sections, it if often desirable to run the gradient flows for several iterations. Therefore, in practice these point clouds may take considerably longer to run than the figures indicate. However, due to the horizontal scaling the parallelization affords, if a point cloud is run across more threads, the runtimes will correspondingly decrease. As we saw here, a single iteration of a point cloud with close to a million points (Stadium) takes over nine minutes to complete on eight threads. By increasing the number of threads by a factor of ten (i.e. eighty threads), we could potentially get the runtime down to under a minute for a single iteration.

Another potential approach for scaling the gradient flows to very large point clouds would involve breaking up the LiDAR point cloud into many chunks and running the discrete gradient flows on each chunk. Then the number of points in each point cloud would be much smaller. Additionally, the flows could be run on separate machines and then reunited at the end. The difficulty with this approach, of course, is the boundary behavior of the gradient flows. In particular, the boundaries may not align between adjacent chunks after the flow. To mitigate this issue, each chunk would have an overlapping boundary with all its adjacent chunks. Then, when stiching together the smoothed chunks, one could remove half of the overlapping boundary from each chunk and align the resulting new boundaries. Another, potentially more effective approach, would be averaging the position of the pairs of points from each chunk. This form of horizontal scaling would be much more cost effective than horizontal scaling using threading. However, this form of horizontal scaling falls outside the scope of this dissertation.

Chapter 6: Conclusion

In this dissertation, we investigated the gradient flow induced by the distance-to-measure function. In particular, we focused on the distance-to-measure function induced by the empirical distribution of a point cloud $\mathcal{X}$. This version of the distance-to-measure function is a k-nearest neighbor energy function. In Chapter 3, we built a theoretical foundation for the study of the gradient flow induced by this k-nearest neighbors function. The foundation was built on the notion of the higher order Voronoi diagrams. For each k-order Voronoi region V in the k-order Voronoi diagram $V^k(\mathcal{X})$, we determined a condition which characterizes when the region contains a sink of the gradient flow. This occurs precisely when the region V satisfies the barycentric sink condition, $\text{Bar}_G(V) \in V$. If the region does not satisfy this condition, then points in V will eventually flow outside the region.

Due to the gradient not being well-defined along the boundary $\partial V_i \cap \partial V_j$ of two k-order Voronoi regions V_i and V_j, we needed to find an appropriate means of defining the gradient. For this, we turned to the notion of a piecewise smooth gradient flow and showed that if we define the gradient along the boundary using Filippov's convex method, we obtain a piecewise smooth gradient flow with a well defined gradient along the boundary. For Filippov systems, the behavior along the boundaries is important. We are not too concerned about boundaries exhibiting repulsive sliding since solutions cannot reach these boundaries in forward time. For boundaries exhibiting aligned gradients we can simply follow the flow to define the gradient. On the other hand, of particular concern are boundaries $\partial V_i \cap \partial V_j$ which exhibit attractive sliding, that is the gradient from V_i points into V_j and the gradient from V_j points into V_i. These boundaries are important because the boundary can be reached in forward time under the gradient flow. However, we showed in Section 3.3.1 that the k-nearest neighbors gradient flow does not exhibit any attractive sliding. This is a

critical result for numerical implementations of the k-nearest neighbors gradient flow since if attractive sliding were present, we would need access to the k-order Voronoi diagram for an accurate implementation. Since k-order Voronoi diagrams are notoriously expensive to compute in high dimensions, requiring this diagram for an accurate implemetation would make such implementations prohibitively expensive.

Once the gradient is defined on the boundary, we can establish some additional characteristics of the k-nearest neighbors gradient flow. In particular, the flow does not contain any periodic orbits. Additionally, the continuity of the gradient flow yields continuity in t of the persistence diagrams arising from the point clouds $\mathcal{X}_t$, where $\mathcal{X}_t$ is the evolved point clouds after the gradient flow has been run for time t. This establishes that the vineyards, i.e. the stacked persistence diagrams, are also continuous. Thus, as the gradient flow is run, the topology of the point cloud, as measured through the persistence diagrams, evolves continuously.

In Chapter 4, we developed several extensions to the k-nearest neighbors gradient flow to address several shortcomings that were identified during the analysis of the original gradient flow. To begin, we extended a method from surface reconstruction to approximate the normal and tangent bundles of the manifold from which the point cloud $\mathcal{X}$ was sampled. This techniques relies on computing a covariance matrix of the neighborhood $\mathrm{NN}^k_{\mathcal{X}}(x)$ (i.e. the k-nearest neighbors of $\mathcal{X}$). Once the covariance matrix is computed, we then take the eigendecomposition of the matrix and use the span of the smallest eigenvectors, as measured by their eigenvalues, to approximate the normal space around x. Additionally, we can let the span of the largest eigenvectors, again as measured by their corresponding eigenvalues, approximate the tangent space. Thus, for every point we have an approximated coordinate frame, decomposed into normal and tangent spaces.

Building on the normal and tangent bundle approximations, we can project the k-nearest neighbor gradient at a point x into the approximated normal space around x, yielding a new vector $g_N(x)$. We then induce a gradient flow using $g_N(x)$ instead of $\nabla E^k_{\mathcal{X}}(x)$. This allows the point x in a k-order Voronoi region V to flow toward $\mathrm{Bar}_G(V)$, but only through

the normal space. This technique drastically reduces clustering, a common issue with the k-nearest neighbors gradient flow. While this modification only uses the normal space, we can incorporate the tangent space as well if we add a diffusive term to the gradient flow. In particular, if we compute the k-nearest neighbors of x from the point cloud $\mathcal{X}_t$ instead of $\mathcal{X}_0$, and compute the gradient $E^k_{\mathcal{X}_t}(x)$ using these neighbors, then we can project this vector into the tangent space around x, yielding the vector $g_T(x)$. We then induce a gradient flow using the vector $-g_N(x) + \lambda g_T(x)$, where we choose $\lambda \in [0, 1]$ to control the level of diffusion. Since the sign in front of the projected gradient $g_T(x)$ is opposite that of $g_N(x)$, we are forcing the points to push against one another when they become close in $\mathcal{X}_t$.

Not every point cloud requires the same amount of smoothing everywhere. Often, in real data, the level of noise varies at different points of the point cloud. Therefore, we would like to add adaptivity to our gradient flow so that the degree of smoothing is determined by local geometric conditions. Methods of adding adaptivity to surface smoothing techniques exist, however they are often too computationally expensive to extend to higher dimensions. To allow adaptive smoothing in higher dimensions, we developed a new technique for adapting the gradient flow to local conditions. In Section 4.3, we approximate the curvature of the point cloud around a point x using the approximated normal and tangent spaces around x. In particular, we look at the distance of the nearest neighbors of x to the point x in both the normal space and the tangent space. This yields a set of points in $\mathbb{R}^2$ representing how far away each neighbor is in both spaces. We then fit a line of best fit to these points. Using these lines, for a point $x \in \mathcal{X}$ whose line of best fit exhibits a large slope, we know the nearest neighbors move away from x in the normal space faster than in the tangent space. Since curvature can be seen as deviation from the tangent space, we can view the high slope as representing high curvature. On the other hand, points with small slope correspond to points in $\mathcal{X}$ where $\mathcal{X}$ locally exhibits low curvature. Once we have an approximated measure of the local curvature of the point cloud, we can adjust the parameters of the gradient flow to reflect the curvature. In this work, we looked at lowering the value of k when the slope is large, in an attempt to preserve the high curvature features. As we saw through example,

this technique can help alleviate corners being rounded off by the gradient flow, since corners exhibit infinitely high curvature and are thus smoothed with a much lower value of k.

Changing focus slightly, we then turned to the Mahalanobis distance, d_M, a function often used in statistics. Since this function is used to provide a notion of the distance from a point to a probability distribution (a goal in common with the distance-to-measure function), it seems natural to compare th gradient flow induced by d_M with the gradient flow induced by the distance-to-measure function. However, before we could make the comparison, we had to introduce the notion of the local Mahalanobis distance, which computes the Mahalanobis distance on the nearest neighbors of a point. We then found that the unweighted normal bundle flow shares a strong connection with the local Mahalanobis gradient flow. In particular, both flows take the vector $w = x - \text{Bar}_G(V)$, where $x \in V$, and project it into the eigenspace of the covariance matrix. However, the weights for the projection differ. In particular, the local Mahalanobis gradient flow projects w along the eigenvector v_i with a weight of $1/\lambda_i$ where λ_i is the eigenvalue corresponding to v_i. This causes greater flow along eigenvectors with smaller eigenvalue. On the other hand, if we assume the point cloud was sampled from a codimension m manifold, the unweighted normal bundle flow projects the vector w along the eigenvector v_i with weight 1 if $i \leq m$ and weight 0 otherwise (where the eigenvectors are sorted in an increasing order, according to their eigenvalue). This led us to generalize the two flows by creating a new flow with arbitrary weights w_i for each eigenvector v_i. We then introduced the normalized Mahalanobis flow, whose weights were given by $w_i = (\lambda_i \tau)^{-1}$ where τ is the sum of all the eigenvalues $\{\lambda_i\}$.

The final topic we address in Chapter 4 concerns computing nearest neighbors. This calculation is the most expensive part of the algorithm and so great care must be taken when determining how to compute the nearest neighbors. Of course, naïvely computing the nearest neighbors by computing the distance matrix exhibits quadratic complexity. As discussed in Section 4.5, we can employ a $k-d$ tree to improve the computational complexity of finding the nearest neighbors. However, this is only useful for smaller dimensions (up to about 20) and does not scale well to very high dimensions. In order to continue using the

smoothing algorithms discussed in this proposal in the higher dimensional setting, we would need to turn to approximate nearest neighbors instead of exact nearest neighbors. Although this would introduce another source of error, the computational savings allow the gradient flow algorithms to become tractable even in high dimensional spaces. Investigating how the use of approximate nearest neighbors over exact nearest neighbors impacts the performance of the algorithms is, for now, left as an open question. Further research could be aimed at determining how much error is incurred in the switch and how much computational gain comes from the switch.

Finally, in Chapter 5, we looked at how the pointwise geometric error and the topological error of the point clouds changes as the clouds evolve under the smoothing algorithms. We found topologically and geometrically optimal parameters for point clouds sampled from several different manfiolds. Additionally, We saw how the topology of simplicial complexes built from the evolving point clouds, a common mechanism in topological data analysis, changed as we evolved the clouds. To determine this change, we used persistence diagrams. Furthermore, for the simple geometries studied in Section 5.1, we knew the topology of the sampled manifold and therefore we could easily measure the topological accuracy of the smoothed point clouds.

In addition to the simple geometries, we studied the effectiveness of the smoothing algorithms when applied to point clouds obtained through 3d scans of the Stanford Bunny and the Stanford Dragon. For these experiments, we added noise to the original point clouds. Then, we computed the geometric error of the smoothing algorithms by comparing the final smoothed point clouds with original, noise-free point clouds. As we saw, both the k-nearest neighbors gradient flow and the normal bundle flow reduced the geometric error of the noisy point clouds. Additionally, we saw that the k-nearest neighbors clustering issue was quite pronounced when smoothing the Stanford Dragon. However, the normal bundle flow prohibited this clustering and therefore produced a more uniformly dense point cloud. Thus, we saw that the normal bundle flow performed as designed.

Although this work established a theoretical foundation for the study of the gradient

flow induced by the distance-to-measure function d_{μ,m_0}, when μ is chosen to be the empirical distribution, and extended the gradient flow to overcome some critical issues, there is still plenty of work to be done. In general, it would be useful to have automated heuristics which are able to determine effective parameters for the gradient flow. As it stands, one has to experiment with the parameters to find suitable settings. Additionally, several of the modifications to the gradient flow could use further investigation.

Our current formulation of the adaptivity coefficients simply uses a single line of best fit, thus producing a single number indicative of the local curvature. A potentially more appropriate approach would consist of approximinating the curvature along every tangent eigenvector we obtain from the covariance matrix. Then the resulting slopes would act as the principal curvatures, whose product could be used as the final adpativty coefficient. This may produce better curvature approximations. Another extension of the adaptivity computation would involve using weights for each distance pair $(d_T(x,y), d_N(x,y))$. There weights would be a function of the distance $d_T(x,y)$ so that points closer to x in the tangent space would have greater influence over the parameters of the line of best fit. This modification would help to avoid distant points exerting too great an influence over the local curvature approximation.

The connection between the unweighted normal bundle flow and the local Mahalanobis flow was made explicit in Section 4.4. Building on the connection, we generalized this type of flow by adding a weight term to each eigenvector projection. However, we did not do any deep investigation of other weights which may produce an effective flow. Experimenting with various weighting schemes may produce some interesting flows which could outperform the normal bundle flow. This is a topic which could receive greater attention in future work.

Finally, the entirety of this work focused on the distance-to-measure function induced by the empirical distribution. An analysis of the gradient flow induced by the distance-to-measure function under arbitrary probability measures is still left open. Clearly, the k-order Voronoi diagram framework we built in Chapter 3 would not suffice for the study of these

more general flows. Many interesting probability distributions exist for which the distance-to-measure gradient flow may prove effective. For example, if instead of using point masses at each point $x \in \mathcal{X}$, we centered Gaussian distributions at each point, then the resulting probability distribution would be a kernel density estimation of the underlying manifold. How would the gradient flow induced by the distance-to-measure function for a kernel density estimated measure differ from the gradient flow we studied in this dissertation? In this case, it may even be possible to approximate the normal bundle of the underlying manifold not from the covariance matrix we computed in Section 4.1, but instead from the geometry of the kernel density estimation itself. These questions, while interesting, are left open.

Bibliography

[1] Liblas sample lidar data. `http://liblas.org/samples`. Accessed: 2017-01-05.

[2] Opentopography. `http://www.opentopo.sdsc.edu`. Accessed: 2017-01-05.

[3] The stanford bunny. `http://www.cc.gatech.edu/~turk/bunny/bunny.html`. Accessed: 2017-02-11.

[4] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. *Proceedings of the IEEE Visualization*, 2001.

[5] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1), 2003.

[6] Franz Aurenhammer. A new duality result concerning Voronoi diagrams. *Discrete and Computational Geometry*, 5:243–254, 1990.

[7] Alexander I. Bobenko and Peter Schröder. Discrete Willmore flow. *Eurographics Symposium on Geometry Processing*, 2005.

[8] V.I. Bogachev and A.V. Kolesnikov. The Monge-Kantorovich problem: Achievements, connections, and perspectives. *Russian Mathematics Surveys*, 67:785–890, 2012.

[9] Jean-Daniel Boissonnat and Fréderic Chazal. Smooth surface reconstruction via natural neighbour interpolation of distance functions. *16th ACM Symposium on Computational Geometry*, pages 185–203, 2002.

[10] Claire Caillerie, Frédéric Chazal, Jérôme Dedecker, and Bertrand Michel. Deconvolution for the Wasserstein mertric and geometric inference. *Electronic Journal of Statistics*, 5, 2011.

[11] James B. Campbell and Randolph H. Wynne. *Introduction to Remote Sensing, Fifth Edition.*, volume 5th ed. The Guilford Press, 2011.

[12] Piermarco Cannarsa. *Semiconcave Functions, Hamilton-Jacobi Equations, and Optimal Control*. Birkhäuser, 2004.

[13] JC Carr, RK Beatson, JB Cherrie, TJ Mitchell, WR Fright, BC McCallum, and TR Evans. Reconstruction and representation of 3d objects with radial basis functions. *Proceedings of the ACM SIGGRAPH 2001*, pages 67–76, 2001.

[14] Frédric Chazal, David Cohen-Steiner, and Quentin Mérigot. Geometric inference for measures based on distance functions. *INRIA Rapport de recherche*, 2010.

[15] Frédric Chazal, David Cohen-Steiner, and Quentin Mérigot. Geometric inference for probability measures. *Foundations of Computational Mathematics*, 11:733–751, 2011.

[16] Bernard Chazelle and Herbert Edelsbrunner. An improved algorithm for constructing kth order Voronoi diagrams. *IEEE Transactions on Computers*, 36(11), 1987.

[17] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Proceedings of the 21st Annual Symposium on Computational Geometry*, pages 263–271, 2005.

[18] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and vineyards by updating persistence in linear time. *Twenty-Second Annual Symposium on Computational Geometry (SCG '06)*, 2006.

[19] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. *SIGGRAPH 1996 Proceedings*, 1996.

[20] B. Delaunay. Sur la sphére vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[21] Tamal Dey, Joachim Giesen, and Samrat Goswami. Delaunay triangulations approximate anchor hulls. *Compuational Geometry: Theory and Applications*, 36:131–143, 2006.

[22] Tamal K. Dey and Jian Sun. An adaptive mls surface for reconstruction with guarantees. *Eurographics Symposium on Geometry Processing*, 2005.

[23] Tamal K. Dey and Wulue Zhao. Approximating the medial axis from the voronoi diagram with a convergence guarantee. 2002.

[24] M. di Bernardo, C.J. Budd, A.R. Champneys, and P. Kowalczyk. *Piecewise-smooth Dynamical Systems: Theory and Applications*. Springer, 2008.

[25] R.L. Dobrushin. Prescribing a system of random variables by conditional distributions. *Theory of Probability and its Applications*, 15(3):458–486, 1970.

[26] Herbert Edelsbrunner and John L. Harer. *Computational Topology: An Introduction*. AMS, 2010.

[27] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, pages 153–174, 1987.

[28] C.M. Gevaert, C. Persello, R. Sliuzas, and G. Vosselman. Informal settlement classification using point-cloud and image-based features from uav data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 125:225–236, 2017.

[29] Dejan Govc. On the definition of the homological critical value. *Journal of Homotopy and Related Structures*, pages 1–9, 2013.

[30] Joachim Gudmundsson, Mikael Hammar, and Marc van Kreveld. Higher order Delaunay triangulations. *Computational Geometry*, 23(1):85–98, 2002.

[31] Guillemin and Pollack. *Differential Topology.* 1974.

[32] Shin-Yi Hsu. The mahalanobis classifier with the generalized inverse approach for automated analysis of imagery texture data. *Computer Graphics and Image Processing*, 9:117–134, 1979.

[33] Lei Hu, Xiaojun Xu, Lifeng Wang, Na Guo, and Feng Xie. 3d registration method based on scattered point cloud from b-model ultrasound image. volume 10245, pages 102450C–102450C–9, 2017.

[34] L.V. Kantorovich and G.S.H. Rubinshtein. On a space of totally additive functions. *Vestnik St. Petersburg University*, 13(7):52–59, 1958.

[35] Der-Tsai Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Transactions on Computers*, 31(6):478–487, 1982.

[36] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[37] P.C. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Science*, 1936.

[38] Songrit Maneewongvatana and David M. Mount. It's okay to be skinny, if your friends are fat. *4th Annual CGC Workshop on Computational Geometry*, 4, 1999.

[39] Wusheng Chou Mingjie Dong and Bin Fang. Underwater matching correction navigation based on geometric features using sonar point cloud data. *Scientific Programming*, 2017, 2017.

[40] John W. Morgan and Gang Tian. *Ricci Flow and the Poincaré Conjecture.* 2007.

[41] Dmitriy Morozov. Dionysus. `http://www.mrzv.org/software/dionysus/`. Accessed: 2015-06-07.

[42] Elizabeth Munch. *Applications of Persistence Homology to Time Varying Systems*. Deparment of Mathematics, Duke University, 2013.

[43] James Munkres. *Topology: Second Edition*. Pearson, 2000.

[44] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Transactions of Graphics (TOG)*, 22(3), 2003.

[45] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[46] Mark Pauly, Leif Kobbelt, and Markus Gross. Multiresolution modeling of point-sampled geometry. *CS Technical Report #378*, 2002.

[47] B Y R. Penrose and Communicated by J. A. Todd. A generalized inverse for matrices. 1954.

[48] Joseph Rotman. *An Introduction to Homological Algebra*. Springer Universitext, 2008.

[49] M. Hahn M. Mokhtarzade H. Arefi S. Malihi, M.J. Valadan Zoej. 3d building reconstruction using dense photogrammetric point cloud. volume XLI-B3 of *XXIII ISPRS Congress*, 2016.

[50] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers, 2006.

[51] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998.

[52] Philippe Steer, Dimitri Lague, Aurélie Gourdon, Thomas Croissant, and Alain Crave. 3d granulometry: grain-scale shape and size distribution from point cloud dataset of river environments. *EGU General Assembly 2016*, 18, 2016.

[53] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 311–318, New York, NY, USA, 1994. ACM.

[54] L.N. Vasershtein. Markov processes over denumerable products of spaces describing large system of automata. *Problems Inform. Transmission*, 5(3):47–52, 1969.

[55] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981.

[56] Brian White. Evolution of curves and surfaces by mean curvature. *Proceedings of the International Congress of Mathematics*, 1:525–538, 2002.

Curriculum Vitae

Include your *curriculum vitae* here detailing your background, education, and professional experience.