On Intrinsic Dimension Estimation and Minimal Diffusion Maps Embeddings of Point Clouds

von

Johannes von Lindheim

Masterarbeit in Mathematik
vorgelegt dem Fachbereich Mathematik und Informatik
der Freien Universität Berlin
am 11.05.2018

Betreuer: Dr. Ralf Banisch
Zweitgutachter: Prof. Dr. Christof Schütte

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Berlin, den 11.05.2018

Johannes von Lindheim
Institut für Mathematik
Arnimallee 6
Freie Universität Berlin
14195 Berlin
jojovli@gmx.de

Contents

Acknowledgments

I would like to thank my thesis supervisor Dr. Ralf Banisch for the academic encouragement, lots of useful advice and all the continual support. I also would like to thank the head of my research group at the Zuse Institute Berlin, PD Dr. Marcus Weber, for interesting discussions and the large freedom I had in my work. Special thanks to Luzie Helfmann and Lukas Polthier for proofreading this thesis and the helpful feedback.

1. Introduction

Since the beginning of the information age, the number of ways to capture information with sensors, apps, videos, tracking of consumer behavior on the Internet, social media etc. has been quickly increasing. According to an IBM Cloud Marketing study for 2017 [Clo16], 2.5 quintillion bytes are created every day. Often, this data is given in the form of a number of n *observations*, each of them described by a number of D *variables* or *attributes*, where D is high. One views such a set X of these observations as a cloud of data points $\{x \in X\} \subset \mathbb{R}^D$. This data is often hard to deal with because of the *curse of dimensionality*. For a high number D of variables, not only humans struggle with interpreting this data, but also algorithms and machines.

There are a lot of situations, however, where the *intrinsic* structure of a point cloud X requires a much lower number of so-called *features* to describe it, compared to its *extrinsic* structure. This number is called the *intrinsic dimension (ID)* d of the data, with $d \ll D$. The task of *manifold learning* of *nonlinear dimensionality reduction* is to find a map $\Phi : \mathbb{R}^D \to \mathbb{R}^d$ or an embedding $X \subset \mathbb{R}^d$ respectively, such that the intrinsic structures of the original point cloud are preserved (see figure 1.1).

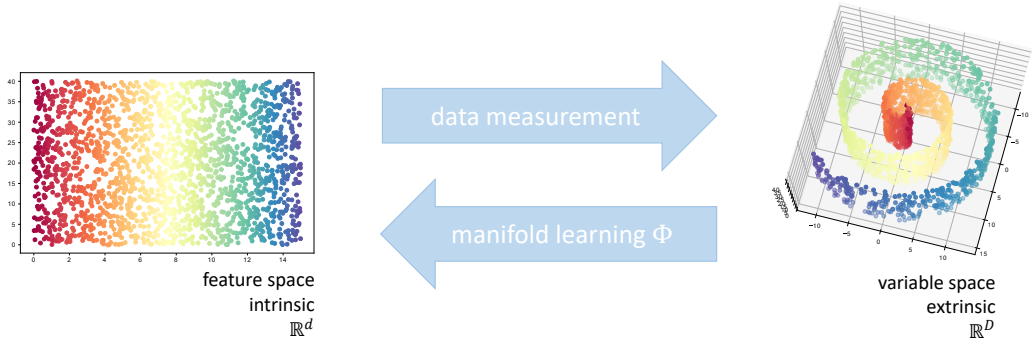


Figure 1.1.: Schematic illustration of manifold learning. $\Phi : \mathbb{R}^D \to \mathbb{R}^d$ shall map the data from the variable space to a lower-dimensional feature space, such that the intrinsic structure is preserved. The colormap indicates one feature or *parameter* of the dataset, which has intrinsic dimension $d = 2$. The extrinsic dimension in this example is $D = 3$.

A reduction of the dimension has the following advantages:

- It mitigates the curse of dimensionality effect, so that potentially less data points are required to achieve certain tasks.
- It saves computation time and storage.
- It sometimes makes the data *interpretable*, breaking it down to the "human processing power".

This procedure has numerous applications, including statistics, economics, molecular dynamics, genomics, finance and machine learning. There are many algorithms to

successfully achieve a dimensionality reduction, e.g. Principal Component Analysis (`PCA`) [AW10] and its variants, Multidimensional Scaling (`MDS`) [Kru64] and its variants, Locally-Linear Embedding (`LLE`) [RS00] and its variants, `ISOMAP` [TDSL00], diffusion maps [CL06a] etc. pp.

Most of these algorithms require the a priori unknown intrinsic dimension d of the data as input. That is, they need to know, to how many dimensions to reduce the data. Therefore, it is often desirable to estimate the ID beforehand, which is termed *intrinsic dimension estimation*. We will consider this topic in chapter 2. In chapter 3, we turn to one the the "state-of-the-art" algorithms of manifold learning, called diffusion maps. One unsolved problem with that algorithm is that it does not reduce the data to its minimal dimension because of the so-called *higher harmonics problem*. It occurs whenever the data is *anisotropic*, that is, it does not extend equally wide into all directions. In the course of chapter 3, a new algorithm based on *importance sampling* resolving this problem in most of the cases is presented.

My Contributions

- A novel framework for intrinsic dimension estimation based on importance sampling.
- A discussion of the higher harmonics problem of diffusion maps and a novel algorithm designed to resolve it.
- Implementations and numerical experiments in Python to test the presented approaches. Comparison of the novel ID estimators to existing algorithms from the literature.

For an overview on some notation used throughout this thesis, see table 1. Chapter-specific notation is introduced on the fly.

Symbol	Description
X	finite dataset, $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^D$
n	number of data points, i.e. $n = \#X$
$\mathcal{M}$	underlying manifold of X
d	intrinsic dimension of X resp. dimension of $\mathcal{M}$
D	extrinsic dimension of X, i.e. $X \subset \mathbb{R}^D$
x	data point $x \in X$ (or $x \in \mathcal{M}$, depending on the context)
N_x	a neighborhood (nearby data points) of $x \in X$, i.e. $N_x \subset X$
$\sim$	neighborhood relation, i.e. $x \sim y :\Leftrightarrow y \in N_x$
y, y_i	(generic) neighbor of x, i.e. $y, y_i \in N_x$
n_x	number of neighbors of x, i.e. $n_x = \#N_x$
N	number of pairs of neighbors, i.e. $N = \sum_{x_i \sim x_j \in X, i<j} 1$

Table 1.1.: General notation used in this thesis.

2. Intrinsic Dimension Estimation

Suppose that we have a dataset $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^D$ in D-dimensional real space. We assume that the data is sampled according to some probability density q, that has support on a submanifold $\mathcal{M} \subset \mathbb{R}^D$ of dimension d, i.e. $\operatorname{supp}(q) \subset \mathcal{M}$ with $\dim(\mathcal{M}) = d$. We then say that our data X has *intrinsic dimension (ID)* equal to d. In a lot of real-world cases, the data may be also perturbed with noise. The problem intrinsic dimension estimation is concerned with, is to give an estimate $\hat{d}$ of the ID from a given dataset without any knowledge about q or $\mathcal{M}$. Sometimes, the task is also to estimate ID *locally* for all data points $x \in X$, for instance if the data is sampled from multiple manifolds with different dimension. Consider e.g. a line intersected with a plane, then the estimated ID should be 2 for points in the plane and 1 for points in the line.

We proceed as follows. First, some mathematical notations of the dimension of sets are introduced in 2.1. In section 2.2 some of the wide range of ID heuristics existing in the literature are briefly reviewed. A novel ID estimation approach is presented in section 2.3. After starting this section by motivating the approach, we introduce two strategies in subsections 2.3.1 and 2.3.2 implementing this ansatz. Heuristics for choosing the important scale parameters are presented in subsection 2.3.3 and subsection 2.3.4 completes the derivation of several variants of the new approach by showing an optimization and closing the section a pseudocode and a brief computational complexity analysis. These algorithms are tested qualitatively for increasing intrinsic dimension and noise scale in 2.4.1. Additionally, they are compared to a large variety of different ID heuristics from the literature in a benchmark precision test in subsection 2.4.2. Finally, section 2.5 highlights some unsolved questions and possible spots for future work.

2.1. Background

We will first briefly introduce some of the mathematical notions of the dimension of sets, on which many intrinsic dimension estimators rely upon. The first mathematical definition of dimension of sets is due to Hausdorff [Hau18].

Definition 2.1 (Hausdorff dimension)**:** *Let Ω be a metric space. For $d \in [0, \infty)$, the d-dimensional* Hausdorff content *of Ω is defined as*

$$C_H^d(\Omega) := \inf\left\{ \sum_i r_i^d : \exists \text{ cover of } \Omega \text{ by a countable set of balls with radii } r_i > 0 \right\}.$$

Then the Hausdorff dimension *of Ω is defined as*

$$\dim_H(\Omega) := \inf\{d \geq 0 : C_H^d(\Omega) = 0\}.$$

One may make sense of the definition by recalling that the volume of d-dimensional balls with increasing radius r grows proportional to r^d in Euclidean space. Hence,

the Hausdorff dimension intuitively is the lowest number, for which the set Ω can be covered with d-dimensional balls. Note that this definition is applicable to more general sets than manifolds like fractal sets, possibly having a real-valued Hausdorff dimension in general.

It is also possible to define a notion of dimension locally, that is, in one single data point [You82].

Definition 2.2 (Pointwise dimension)**:** *Let $B_r(x)$ be a closed ball of radius r and center $x \in R^D$. If p is a probability measure such that the limit*

$$d_p(x) = \lim_{r \to 0} \frac{\log p(B_r(x))}{\log r}$$

exists, then $d_p(x)$ is called the pointwise dimension *in the point x.*

As an intuition, if we take p to be the normalized Lebesgue measure measuring volume on the d-dimensional manifold $\mathcal{M}$, then for $x \in \mathcal{M}$ we get

$$d_p(x) = \lim_{r \to 0} \frac{\log(c \cdot r^d)}{\log r} = \lim_{r \to 0} \frac{\log c + d \log r}{\log r} \xrightarrow{r \to 0} d,$$

since $\mathrm{vol}(B_r(x))$ scales like r^d.

Moreover, there is a topological definition of dimension [Jam99].

Definition 2.3 (Lebesgue covering dimension)**:** *Given a topological space $\mathcal{X}$, a set $\mathcal{Y} \subset \mathcal{X}$ and a covering $\mathcal{C} = \{\mathcal{C}_i\}$ of $\mathcal{Y}$ so that $\bigcup_i \mathcal{C}_i \supseteq \mathcal{Y}$, a* refinement of the cover $\mathcal{C}$ *is any other covering $\mathcal{C}'$, so that for every $C' \in \mathcal{C}'$ there is a set $C \in \mathcal{C}$, such that $C' \subset C$.*

Then the topological dimension of $\mathcal{X}$ *is the smallest number $d \in \mathbb{N}_0$, such that every open cover $\mathcal{C}$ of $\mathcal{X}$ admits an open refinement $\mathcal{C}'$ such that every point $x \in \mathcal{X}$ is contained in at most $d+1$ sets in $\mathcal{C}'$. If no such minimal integer value exists, $\mathcal{X}$ has infinite topological dimension.*

These concepts may be useful for designing ID estimators. We cannot directly use them in our practical problem, however, since we only have a finite point cloud, which has dimension zero for all of these definitions. This is why the intrinsic dimension problem is *ill-posed* in the situation of a fixed finite dataset and no knowledge about $\mathcal{M}$, which is unfortunately the usual case. After all, the point cloud is a finite collection of zero-dimensional manifolds. Assuming that the data is sampled from a single manifold does not solve the problem, since the data always lies on a one-dimensional curve.

What makes this problem also ill-posed is the allowed scale of the (full-dimensional) noise the data is perturbed with. One may recognize a low-dimensional manifold as low-dimensional, if the noise has a low variance relative to the manifold. However, what is the threshold for the scale of the noise, at which one would assign $d = D$ as intrinsic dimension to the data X?

Moreover, ID is dependent on the scale, at which one looks at the data. This is related to the previous paragraph. For instance, if we have a one-dimensional curve,

perturbed with two-dimensional noise, then the manifold looks two-dimensional at a fine scale, since the noise looks full-dimensional. On a slightly coarser scale, the data looks more or less one-dimensional. On an even coarser scale, ID estimation is hard, since linear techniques overestimate ID because of the curvature of $\mathcal{M}$.

These are three of the various problems that make ID estimation hard. According to [CS16], an *ideal* ID estimator has the following properties:

1. It is *computationally feasible*.
2. Is is *robust to multiscaling*.
3. It is *robust to high intrinsic dimensions*.
4. It *has an operative range*, that is, provide guarantees about when the ID estimator gives reliable estimates with regard to properties of the dataset X, e.g. the cardinality $n = \#X$.
5. It needs to be *accurate*, that is, yield estimates close to the dimension of the underlying manifold $\mathcal{M}$, the data is sampled from.

The goal in this chapter is to present a novel estimation approach that shall be precise at least on noise-free datasets and is capable of measuring ID locally, that is, the heuristic should be able to assign an individual estimate to every data point $x \in X$. The estimator will fulfill properties 1, 3 and 5.

2.2. Relevant Work

In this section, we will get an overview on the different categories ID estimators can be divided into, and some notable examples from each category. The categorization mainly follows the more detailed survey [CCCR15] on intrinsic dimension estimation methods. All algorithms that we will use in section 2.4 for the benchmark comparison of the novel approaches are presented in this section.

A first distinction can be made between *global* and *local* methods. Global methods try to estimate the ID of the dataset as a whole, while local methods only use local neighborhoods separately to estimate ID. If a global estimate is needed in case of constant ID on the entire dataset, the local estimates are combined. It is claimed in [CCCR15], however that all recent methods are local methods, since the results are not reliable when trying to estimate ID on the largest scale of the data. Therefore, we will not distinguish between global and local in the following.

Projective

Projective methods try to project the data to a lower-dimensional subspace in some way. The dimension of this subspace is then the estimated ID of the data. This is exactly the situation of manifold learning, introduced in chapter 1. In fact, a lot of manifold learning techniques like multidimensional scaling (`MDS`), `ISOMAP`, locally

linear embedding (LLE) etc. can be used to estimate ID. The methods in the literature either try to preserve distances in the projection (like MDS and its variants) or try to minimize the projection error between the projected and the original data, like Principal Component Analysis and its variants (PCA, local PCA (LPCA), Kernel PCA (KPCA) etc.).

MLSVD is one of the most notable projective methods because its construction is explicitly taking into account the multiscaling of the data. It builds Singular Value Decompositions (SVDs; basically a variant of PCA) in D-dimensional balls $B_r(x)$ around the data points $x \in X$ for different choices of r. Using a least-squares fitting procedure, the analysis of the scale-dependent singular values $\sigma_1(r), \ldots, \sigma_D(r)$ reveals a range of scales $\{r_{\min}, \ldots, r_{\max}\}$, which is not influenced by noise (too small scales) or curvature (too large scales). The estimate $\hat{d}$ is taken to be the index j maximizing the *spectral gap* $\sigma_j(r_s) - \sigma_{j+1}(r_s)$ for $r_s \in \{r_{\min}, \ldots, r_{\max}\}$.

Topological

Topological estimators try to estimate the covering dimension (see definition 2.3). There are only few techniques doing so, one being the Tensor Voting Framework (TVF) [MM04] and its variants [LCC08]. The TVF iteratively diffuses local information on the tangent space of every data point $x \in X$ in the form of tensors, whose eigenvectors span the local tangent space. Although interesting, these methods are either computationally not feasible for high extrinsic dimensions D or not robust to the high dimensionality or curvature.

Fractal

Fractal ID estimators are called that way, because they do not assume a dimension, which is a whole number, but also allow to estimate the dimension of fractals. These techniques rely on the idea that the volume of d-dimensional balls with radius r grows proportional to r^d.

One of the most prominent algorithms of this type is the correlation dimension (CD). Set

$$C_n(r) := \binom{n}{2}^{-1} \sum_{i=1, i<j}^{n} \#\{j : \|x_i - x_j\|_2 < r\}$$

which is counting the fraction of all pairs of points, which lie at most a neighbor radius r apart. Then, for a countable dataset, the *correlation dimension* (CD) is defined as

$$\dim_{\text{corr}} = \lim_{r \to 0} \lim_{n \to \infty} \frac{\log C_n(r)}{\log r}.$$

This is related to the pointwise dimension (see definition 2.2). In practice the estimate is computed using a range of radii $\{r_i\}$ and fitting a line through the points $(\log r_i, \log C_n(r_i))$.

A smoothed version of this estimator was proposed in [HA05], denoted by `Hein` in the following. The idea is to replace the "hard" counting by a smooth kernel k_h having bandwidth h. Precisely, they define

$$U(n,h,d) := \binom{n}{2}^{-1} \sum_{i=1,i<j}^{n} \frac{1}{h^d} k_h(\|x_i - x_j\|^2/h^2)$$

where d is a possible intrinsic dimension of the data. By varying the different parameters (varying n by subsampling), they analyze the convergence behavior of $U(n,h,d)$. The estimate is taken to be the number $d \in \{1,\dots,D\}$ which minimizes the maximum slope when varying n. This work is also notable for the synthetic datasets specifically designed to test the precision of the proposed heuristic, which are also employed in other works.

`CD` is heavily influenced by the choice of the scale of parameters, hence in [Tak85] Takens proposes an estimator that estimates the expectation value of `CD` by a maximum likelihood approach on the distribution of distances between the points. The estimate $\hat{d}$ is computed as

$$\hat{d} = -\left(\frac{1}{\#Q} \sum_{i=1}^{\#Q} r_k \right)^{-1},$$

where $Q = \{r_k : r_k < r\}$ and the r_k are the Euclidean distances between the data points.

Nearest-Neighbor-Based

Methods of this type try to estimate ID by studying distributions of points in small neighborhoods. This category includes `MLE`, one of the most cited ID algorithms, presented in [LB05]. `MLE` models the neighbors y_i of a data point x for growing radii r as events of a Poisson process and the distances $\|x-y_i\|_2$ between x and the neighbors y_i as the corresponding arrival times. This depends on d, so d can be estimated by maximizing the log-likelihood of the empirically observed processes, i.e. the ones given by the data. A global ID estimate is obtained by averaging the values for all points. `MLE` has multiple variants, e.g. trying to remove the parameter choice of the number of neighbors to be considered or to remove biases of the estimator.

Another important set of estimators consists of the $\texttt{MiND}_{\texttt{ML}}$-heuristics, $\texttt{MiND}_{\texttt{KL}}$, as well as `DANCo` and its fast implementation `FastDANCo`. The $\texttt{MiND}_{\texttt{ML}}$-heuristics are presented in [RLC+12] and exploit the probability density function $g(r,k,d) = kdr^{d-1}(1-r^d)^{k-1}$ of the distance of the nearest neighbor y of x. The ID estimate $\hat{d}$ is then computed by a maximum-likelihood-approach. This can be done by trying all different possible $d \in \{1,\dots,D\}$, which is called $\texttt{MiND}_{\texttt{MLi}}$. The other possibility, being able to produce a fractal dimensionality estimate, is called $\texttt{MiND}_{\texttt{MLk}}$, which maximizes the same log-likelihood by setting its derivative to zero and solving for d, which produces a real-valued estimate in general. In the same paper, $\texttt{MiND}_{\texttt{KL}}$ is introduced, which compares

the empirical probability density function of the neighborhood distances from the data with the analytical one obtained from uniformly sampled points on hyperspheres of known increasing dimensionality. The ID estimate is obtained by minimizing the Kullback-Leibler-divergence ($\mathcal{KL}$-divergence) between these two distributions.

An extension of $\texttt{MiND}_{\texttt{KL}}$ is `DANCo` [CBR+14]. This heuristic does not only take into account the nearest-neighbor-distances but also the angles between them. So additionally to the distance distribution, it also compares the empirical angle distribution with the Mises-Fisher-distribution (the analog of the Gaussian distribution in real space on a d-dimensional hypersphere). `DANCo` again chooses the dimension which minimizes the $\mathcal{KL}$-divergence, more precisely the sum of the two $\mathcal{KL}$-divergences for the norms and the angles, respectively. A fast variant `FastDANCo` is also presented in that paper, which uses fitted functions trained using some example datasets, instead of the true statistics employed in the `DANCo`-algorithm.

A recent variant of the nearest-neighbor-approach is `TWO-NN` [FdRL17], which is easy to implement and also tries to account for the multiscaling by subsampling the data. In principle, it only employs the distance between the two nearest neighbors of each data point (hence its name).

Graph-Based

Tools from graph theory can be used to solve a wide range of problems. Also in the field of ID estimation, several techniques have been introduced. Examples include the k-nearest-neighbor-graph (`kNNG`) [CH06], the Minimum Spanning Tree (`MST`) [FR83], the sphere influence graph (`SIG`) [PY01] and their corresponding (generalized) variants. They use several statistics on the constructed graphs, e.g. the *length functional* (the sum of all edge weights) in the case of the `MST`.

A recent method employing graph distances in a multiscale fashion is presented in [GC16], yielding promising results in terms of precision.

2.3. The Gaussian Annulus Approach

To estimate the intrinsic dimension of a finite dataset X, a statistic $T : \mathbb{R}^{n\times D} \to \mathbb{R}$ on the data is needed that depends on the ID of X. We evaluate the statistic for our given dataset and rearrange the equation, such that one side depends on the measurement and the other side is our ID estimate $\hat{d}$.

Motivation

The statistics chosen in this thesis are inspired by a property of the standard multivariate (d-dimensional) Gaussian (or normal) distribution with density function

$$p_d(x) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\|x\|_2^2\right).$$

Namely, standard d-dimensional Gaussian distributed data will have the property that most of the probability mass lies on an annulus of Euclidean distance $\sqrt{d}$ from the center of the Gaussian. More precisely, one can state the following theorem:

Theorem 2.4 (Gaussian Annulus Theorem)**:** *For a d-dimensional spherical standard Gaussian with unit variance in each direction, for any $\beta \leq \sqrt{d}$, all but at most $3e^{-c\beta^2}$ of the probability mass lies within the annulus*

$$\sqrt{d} - \beta \leq \|x\| \leq \sqrt{d} + \beta \tag{2.1}$$

For a proof, the reader is referred to [BHK16].

Note that this property just depends on the norms $\|x\|$ and not on the angles (the d-dimensional Gaussian distribution is rotationally invariant), although there are also e.g. concentration phenomena regarding angles. As mentioned in section 2.2, there are also ID heuristics working with angles (e.g. `DANCo`).

As an intuition, why the radius of the annulus depends on d, consider a centered d-dimensional normal distribution and its variance. One may easily check that the standard multivariante normal distribution is made up from d one-dimensional Gaussians:

$$p_d(x) = \frac{1}{(2\pi)^{d/2}} \exp\left(\sum_{i=1}^{d} -x_i^2\right) = \prod_{i=1}^{d} \frac{1}{(2\pi)^{1/2}} \exp\left(-x_i^2\right) = \prod_{i=1}^{d} p_1(x_i).$$

Therefore, d appears in a number of ways in the context of the Gaussian distribution. For instance, consider the mean squared distance from the center of d-dimensional Gaussian distributed data (the center is chosen to be 0 here):

$$\mathbb{E}_{p_d}\left[\|x\|_2^2\right] = \mathbb{E}\left[\sum_{i=1}^{d} x_i^2\right] = d\,\mathbb{E}_{p_1}\left[x_1^2\right] = d. \tag{2.2}$$

This fact is used in Strategy 2, see subsection 2.3.2. The Gaussian annulus theorem 2.4 states that not only is this expectation value (2.2) equal to d, but that random samples of $\|x\|_2^2$ with $x \sim p_d$ are even tightly concentrated around d. This is useful for Strategy 2, since therefore there are not many data points required to get a reliable estimate.

Reweighting

Recall that the goal is to measure ID locally at all data points $x \in X$. How should the ideas with the Gaussian distribution be used, since we do not have Gaussian distributed data at x to begin with? In this thesis we locally reweigh the data points in such a way that they are indeed Gaussian distributed. In fact, the ideas behind *importance sampling* will be used for this approach (see appendix A). We assume that both the sample density q and the manifold $\mathcal{M}$ are smooth enough, so that locally (in the neighborhood of the data point $x \in X$, at a sufficiently small scale σ) the sample density q is well approximated by a uniform distribution $\mathcal{U}(B_r^{T_x\mathcal{M}}(x))$ on a ball of

radius r around x on $T_x\mathcal{M}$. $T_x\mathcal{M}$ denotes the *tangent space* at $x \in \mathcal{M}$, which is the affine space spanned by the tangential vectors at x. Then we can assign weights

$$w_\sigma^u(i) := p_{x,d,\sigma}(y_i) = \exp\left(-\frac{\|x - y_i\|_2^2}{\sigma^2}\right)$$

to the neighbors y_i of a data point x, where we want to estimate ID. The superscript "u" stands for unnormalized, for more details on the choice of this notation see subsection 2.3.2, where we introduce the key idea of importance sampling. (Note that we omitted the normalizing constant $(2\pi)^{d/2}$ here because of proportionality.) As an intuition, one could view a weight w_i as the fraction of the probability mass located at y_i, that is, if $w_i = \frac{1}{5}$, one may think of "one fifth of a data point". In order to arrive at a normalized probability distribution, we normalize the weights:

$$w_\sigma(i) := \frac{w_\sigma^u(i)}{\sum_{j=1}^{n_x} w_\sigma^u(j)} \tag{2.3}$$

The key to this is that the assignment of these weights does not depend on d or on the tangent space $T_x\mathcal{M}$ but just $\|x - y_i\|_2$; nevertheless, the reweighted uniform distribution will be a d-dimensional standard Gaussian distribution (or more precisely, a Gaussian distribution with cutoff at radius r). This also directly follows from the decomposition of the Gaussian into one-dimensional Gaussians: The restriction of a d-dimensional Gaussian density function to some k-dimensional hyperplane though x yields a k-dimensional Gaussian (apart from the normalization constant).

Two concrete ways to perform these reweightings are considered in this thesis. They result in two different strategies to perform an ID estimate. The first one, see section 2.3.1, directly tries to assemble the case of Gaussian distributed data using a "weighted kernel density estimation", while the second one samples the quantity $\|x - y_i\|_2^2$ from a Gaussian using importance sampling.

Note that σ is always an important scale parameter to choose here. Therefore, after deriving the two strategies, a heuristic designed for this purpose is presented in subsection 2.3.3.

To use Gauss kernels instead of the given distribution is on the one hand motivated by the beautiful and direct connection of the used statistics in equations (2.1) and (2.2) with the intrinsic dimension d. On the other hand, it is confirmed e.g. by [HA05] that using Gaussian kernels instead of hyperspheres, for instance (corresponding to a step function $\mathbb{1}_{B_r(x)}$), can be useful. They achieve better results with their heuristic `Hein` using Gaussian kernels compared to the step function analog, the correlation dimension `CD` [GP83].

2.3.1. Strategy 1: Norm Density Peak

The first strategy is directly trying to assemble the situation of Gaussian distributed data and then determining the radius of the annulus from theorem 2.4. To get a

real-valued estimate $\hat{d}$, we need to remove the β-parameter in inequality (2.1). We do this by computing the exact maximum of the density of the norms $\|x-y\|_2$ instead of looking at intervals, where x is the center of our Gaussian now and y is a variable neighboring point.

The Gauss kernel is rotationally symmetric since its density function just depends on $\|x-y\|_2$. Consequently, we can integrate out the angular coordinates (i.e. we integrate over the $(d-1)$-dimensional hyperspheres) and arrive at the *one-dimensional* probability density function, just depending on the distance $r = \|x-y\|_2$ from the center. That is, we use the probability distribution of the one-dimensional random variable $Z = \|x-y\|_2$, whenever we draw a sample y from the d-dimensional Gaussian. We then expect this probability density function (pdf) to have a bump around $\sqrt{d}$, since most mass is concentrated at $\|x-y\|_2 = \sqrt{d}$ (the radius of the Gaussian annulus).

Instead of performing the explicit integral calculation, using the idea "mass = density × volume", it is clear that for the density function $\xi : \mathbb{R} \to \mathbb{R}$ just depending on $r = \|x-y\|_2$ (compared to $p_d : \mathbb{R}^d \to \mathbb{R}$) we must have $\xi_d(r) \propto \exp\left(-r^2/2\right) \cdot r^{d-1}$. Here, $\exp\left(-r^2/2\right)$ is the probability density part and r^{d-1} is the (proportional) volume of the $(d-1)$-dimensional hypersphere of radius r. In order to arrive at a probability density function, we have to compute the normalizing constant. Using the computer algebra program Mathematica [WRI], we obtain

$$\xi_d(r) = C^{-1} \exp\left(\frac{-r^2}{2}\right) \cdot r^{d-1} \quad \text{with} \quad C = \Gamma\left(\frac{d}{2}\right) 2^{\frac{d}{2}-1}$$

where Γ is the Gamma function.

In practice, however, a choice for the variance of the standard Gaussian has to be made. Denote the variance of the one-dimensional Gaussians by σ^2. The density function for the d-dimensional normal distribution with center x (and covariance matrix $\Sigma = \sigma^2 I$) is

$$p_{x,d,\sigma}(y) = \frac{1}{(2\pi)^{d/2}\sigma^d} \exp\left(-\frac{\|x-y\|_2}{2\sigma^2}\right).$$

Using the argument

$$p_{x,d,\sigma}(y) = C \exp\left(-\frac{\|x-y\|_2^2}{\sigma^2}\right) = C \exp\left(-\left\|\frac{x-y}{\sigma}\right\|_2^2\right) = p_{x,d}(y/\sigma)$$

directly yields the formula for the density of the norms in the case of arbitrary variance of the Gaussian:

$$\xi_{d,\sigma}(r) = C^{-1} \exp\left(\frac{-r^2}{2\sigma^2}\right) \cdot r^{d-1} \quad \text{with} \quad C = \sigma^d \Gamma\left(\frac{d}{2}\right) 2^{\frac{d}{2}-1}.$$

We will call ξ_d or $\xi_{d,\sigma}$ (depending on the context) the *annulus bump function* in the following. Figure 2.1 includes a plot of this function and an empirical validation of

the formulas. We see that indeed most of the area under the graph is concentrated around $\sigma\sqrt{d}$.

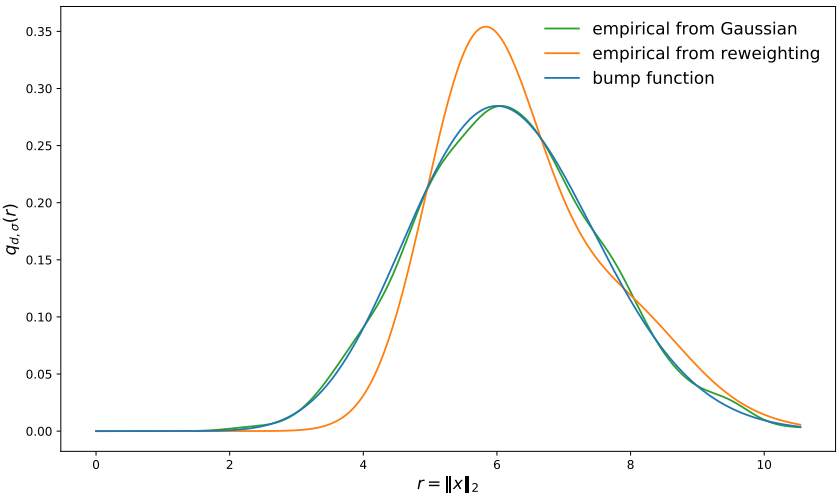


Figure 2.1.: In blue, the annulus bump function is plotted for $d = 10$ and $\sigma = 2.0$. The graph in green is the pdf estimated with a Gaussian kernel density estimation of the norms $\|x\|_2$ of 1000 data points drawn from a centered Gaussian distribution with the same d and σ. The graph in orange was obtained by "simulating" a Gaussian distribution: 1000 data points were drawn uniformly from the 10-dimensional cube $2.0 \cdot \sigma\sqrt{d-1} \cdot [-1, 1]^{10}$, which is chosen large enough to fully "contain" the annulus. Then a weighted kernel density estimation was performed for $\sigma = 2.0$ (see appendix A and the text for more details).

Finally, some further calculus performed by Wolfram Mathematica [WRI] yields that the maximum of the annulus bump function lies at

$$r_{\max} = \sigma\sqrt{d-1}. \tag{2.4}$$

In other words, if we sample from a d-dimensional Gaussian with some σ, the norms of the samples will be maximally concentrated at $r_{\max} = \sigma\sqrt{d-1}$. Interestingly, the maximum does *not* lie exactly at $\sigma\sqrt{d}$ as one would expect from the Gaussian annulus theorem 2.4. This is because the bump is not symmetric but slightly skewed: The maximum is "tilted to the left" a bit. See figure 2.1 again for an example; the maximum indeed lies at $\sigma\sqrt{d-1} = 2\sqrt{10-1} = 6$ in this example.

Consequently, we obtain our ID estimate via

$$r_{\max} = \sigma\sqrt{d-1} \quad \Leftrightarrow \quad d = \left(\frac{r_{\max}}{\sigma}\right)^2 + 1 \tag{2.5}$$

i.e. we *measure* $r_{\max}$ and estimate $\hat{d} := \left(\frac{r_{\max}}{\sigma}\right)^2 + 1$ at the point x.

The question remains how to measure the position of the bump when we do not have Gaussian distributed data. This is where the reweighting inspired by importance sampling is applied: We use a weighted kernel density estimation (weighted KDE). Briefly summarized, we are not adding a full Gauss hat for every norm of a neighbor $\|x-y_i\|_2$ to the overall estimation of the density function, but we rescale them according to their weight $w_\sigma^u(i)$ from equation (2.3); see appendix A for more details. Note that we do *not* include x as its own neighbor, since it would not be a randomly drawn point from either the uniform or the Gaussian distribution.

Figure 2.1 shows such a "simulated" Gaussian bump function. 1000 data points were drawn from a uniform distribution and a weighted kernel density estimation with the same σ as the analytical bump function was applied, trying to resemble the original annulus bump function using this reweighting described above. There is not a perfect agreement between the graphs, but still the density peak of the "simulated" bump function lies at about the same location $r_{\max} \approx 6$.

In summary, Strategy 1 tries to manually and locally assemble annulus bump functions that we would have in the case of Gaussian distributed data at the points $x \in X$. Then it extracts a local ID estimate from the measurement of the maximum density spot $r_{\max}$ of that function, since we expect it to be at $r_{\max} = \sigma\sqrt{d-1}$.

2.3.2. Strategy 2: Mean Squared Distance from the Center

The second strategy presented in this thesis does not assemble the annulus function and then extract some statistics, but it directly estimates the mean squared distance $\mathbb{E}_{p_{x,d,\sigma}}[\|x-y\|_2^2]$ from the center x of the Gaussian. We are interested in this quantity because

$$\mathbb{E}_{p_{x,d,\sigma}}\left[\|x-y\|_2^2\right] = \mathbb{E}\left[\sum_{i=1}^{d}(x-y)_i^2\right] = d\,\mathbb{E}_{p_{x,1,\sigma}}\left[(x_1-y_1)^2\right] = \sigma^2 d, \tag{2.6}$$

analogously to equation (2.2). Hence, if we choose σ and estimate the expected mean squared distance of the neighbors from the center x of the Gaussian distribution, we have an estimate for d by dividing (2.6) by σ^2.

The corresponding estimation shall be performed using ideas from *importance sampling*. The key idea of that procedure applied to our context is described in appendix A.

Estimating the Mean Squared Distance from the Center

How may we use the quite general theorem A.1 on self-normalizing importance sampling estimates for an ID estimator, if we assume our data in the neighborhood N_x around x to be distributed uniformly in a unit ball $B_r(x)$? The natural idea is to view the neighbors $y_i \in N_x$ as samples from this uniform distribution as the importance distribution, and take the Gaussian distribution as the nominal distribution. Let $B_r(x)$ denote the ball of radius r lying in the tangent space $T_x\mathcal{M}$. Mo-

reover, set $p^u := p^u_{x,d,\sigma} = \mathbb{1}_{B_r(x)} p_{x,d,\sigma}$ defined on the tangent space $T_x\mathcal{M}$, that is $\operatorname{supp}(\mathbb{1}_{B_r(x)} p_{x,d,\sigma}) \subset T_x\mathcal{M}$. Let $C_r \in \mathbb{R}$ be the corresponding normalizing constant so that $C_r \int_{T_x\mathcal{M}} \mathbb{1}_{B_r(x)}(y) p^u_{x,d,\sigma}(y) dy = 1$. Let $p := C_r p^u = C_r \mathbb{1}_{B_r(x)} p_{x,d,\sigma}$ be the normalized Gaussian distribution on $T_x\mathcal{M}$ with cutoff for simplicity of notation. Then the following corollary from theorem A.1 shows convergence of the corresponding self-normalized importance sampling estimate to our quantity of interest.

Corollary 2.4.1: *Let $Y_i \overset{i.i.d}{\sim} \mathcal{U}(B_r(x))$ be uniformly distributed in the d-dimensional unit ball centered at x so that $B_r(x) \subset T_x\mathcal{M}$ and let $X \sim p$ be Gaussian distributed with cutoff on the tangent space $T_x\mathcal{M}$. Then*

$$\mathbb{P}\left(\frac{\sum_{i=1}^{n_x} p_{x,d,\sigma}(Y_i) \|x - Y_i\|_2^2}{\sum_{i=1}^{n_x} p_{x,d,\sigma}(Y_i)} \overset{n_x \to \infty}{\longrightarrow} \mathbb{E}_p\left[\|x - X\|_2^2 \right] \right) = 1.$$

Proof. Additional to the previous notation $p^u = p^u_{x,d,\sigma} = p_{x,d,\sigma} \mathbb{1}_{B_r(x)}$ and $p = C_r p^u$, set

$$\begin{aligned} q^u(\cdot) &:= 1 \\ q(\cdot) &:= \operatorname{vol}(B_r(x))^{-1} \\ f(\cdot) &:= \|x - \cdot\|_2^2 \\ n &:= n_x \\ \mu &:= \mathbb{E}_p\left[\|x - X\|_2^2 \right], \end{aligned}$$

in accordance to the notation in the general theorem A.1. Then we have

$$\begin{aligned} \frac{\sum_{i=1}^{n_x} p_{x,d,\sigma}(Y_i) \|x - Y_i\|_2^2}{\sum_{i=1}^{n_x} p_{x,d,\sigma}(Y_i)} &= \frac{\sum_{i=1}^{n} p^u(Y_i) f(Y_i)}{\sum_{i=1}^{n} p^u(Y_i)} \\ &= \frac{\sum_{i=1}^{n} p^u(Y_i) f(Y_i) / q^u(Y_i)}{\sum_{i=1}^{n} p^u(Y_i) / q^u(Y_i)} \\ &= \frac{\sum_{i=1}^{n} w^u(Y_i) f(Y_i)}{\sum_{i=1}^{n} w^u(Y_i)} \\ &= \tilde{\mu} \\ &\overset{n \to \infty}{\longrightarrow} \mu \end{aligned}$$

by theorem A.1 with probability 1. □

It is easy to see that

$$p = C_r \mathbb{1}_{B_r(x)} p_{x,d,\sigma} \overset{r \to \infty}{\longrightarrow} p_{x,d,\sigma},$$

which gives

$$\mathbb{E}_p\left[\|x - X\|_2^2 \right] \overset{r \to \infty}{\longrightarrow} \mathbb{E}_{p_{x,d,\sigma}}\left[\|x - X\|_2^2 \right] \overset{(2.6)}{=} \sigma^2 d.$$

In other words, if we increase the radius of the ball in which we are measuring, we approach the expectation of a real Gaussian distribution in the limit for the data points.

Therefore, obtaining an ID estimate $\hat{d}$ for a large enough radius and sufficiently many data points is straightforward: Choose σ, compute the weights $w_\sigma^u(i) = p_{x,d,\sigma}(y_i) = \exp\left(-\|x - y_i\|_2^2/\sigma\right)$ and $w_\sigma(i) = w_\sigma^u(i)/\sum_{j=1}^{n_x} w_\sigma^u(j)$ like above. Let y be a randomly sampled neighbor of x, then

$$\mathbb{E}_{p_{x,d,\sigma}}\left[\|x-y\|_2^2\right] = \sigma^2 d \quad \Rightarrow \quad \hat{d} = \frac{\sum_{i=1}^{n_x} w_\sigma^u(i)\|x-y_i\|_2^2}{\sigma^2 \sum_{i=1}^{n_x} w_\sigma^u(i)} = \frac{1}{\sigma^2}\sum_{i=1}^{n_x} w_\sigma(i)\|x-y_i\|_2^2 \tag{2.7}$$

Again, x is not taken as a neighbor of itself, since it would not be randomly drawn from some distribution after fixing the center as x.

How large does r have to be for a good approximation? Intuitively, the ball $B_r(x)$ has to be large enough to "contain" the Gaussian annulus, i.e. $r > \sigma\sqrt{d-1}$ by equation (2.4). Exact integration (assisted by Mathematica [WRI]) shows that

$$\begin{aligned}
\mathbb{E}_p\left[\|x-X\|_2^2\right] &= C_r \int_{B_r(x)} p_{x,d,\sigma}(y)\|x-y\|_2^2\, dy \\
&= \frac{1}{\int_0^r \xi_{d,\sigma}(s)\, ds}\int_0^r \xi_{d,\sigma}(s)s^2\, ds \\
&= \frac{\sigma^2\left(d\Gamma\left(\frac{d}{2}\right) - 2\Gamma\left(\frac{d}{2}+1, \frac{r^2}{2\sigma^2}\right)\right)}{\Gamma\left(\frac{d}{2}\right) - \Gamma\left(\frac{d}{2}, \frac{r^2}{2\sigma^2}\right)},
\end{aligned}$$

where $\Gamma(a,x)$ denotes the upper incomplete gamma function $\Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}dt$. Figure 2.2 shows a plot of $\mathbb{E}_p\left[\|x-X\|_2^2\right]$ for different r and exemplary choices $\sigma = 2, d = 10$.

One indeed observes that $\mathbb{E}_p\left[\|x-X\|_2^2\right]$ converges quickly to $\mathbb{E}_{p_{x,d,\sigma}}\left[\|x-X\|_2^2\right] = \sigma^2 d = 2^2 \cdot 10 = 40$ in this example, as soon as r is large enough so that $B_r(x)$ "fully contains" the annulus, which has its peak at $\sigma\sqrt{d-1} = 2\sqrt{10-1} = 6$.

2.3.3. Heuristics for the Scale Parameters

All formulas derived above for ID estimation depend on the scale parameter σ. Moreover, we would like to use the ε-nearest-neighbor method (see appendix A for more details) to avoid the computation of all distances between the data points in order to save computation time. This subsection describes the heuristics used in order to compute a good choice of these parameters σ and ε.

A Heuristic for the Kernel Scale σ

In an ideal scenario, the choice of the scale of reweighting does not matter, since the influence σ is considered in the formulas for the Strategy 1 and 2 ID estimates $\hat{d}$. The fact that the radius of the Gaussian annulus depends on σ is compensated for in

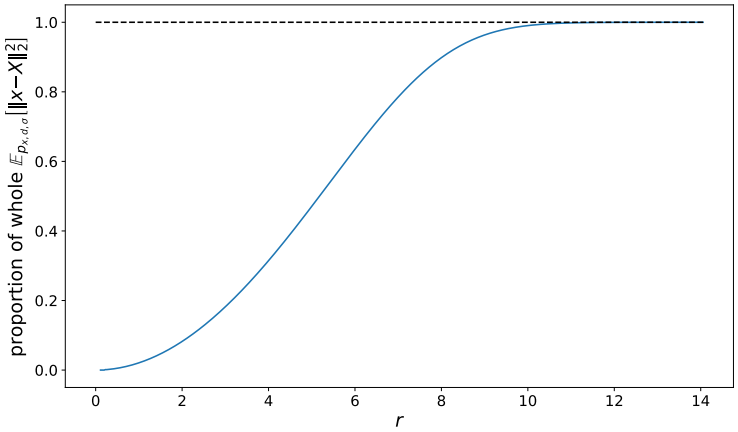


Figure 2.2.: Plot of $\mathbb{E}_p\left[\|x-X\|_2^2\right] / \mathbb{E}_{p_{x,d,\sigma}}\left[\|x-X\|_2^2\right]$ for exemplary choices of $\sigma = 2, d = 10$. The density peak of the importance sampled Gaussian lies at $\sigma\sqrt{d-1} = 2\sqrt{10-1} = 6$.

equations (2.5) and (2.7). In practice, however, the data lies on the curved and noisy manifold $\mathcal{M}$ and the reweighting or importance sampling does *not* resemble perfect Gaussian distributions independent of the scale.

Consequently, we try to choose σ using some optimality criterion. In this case, on the one hand we want a σ that is not too small, so that enough data points are included (regarding the "effective sample size", which we will not define here) and the noise is not dominant. On the other hand, σ shall also not be too large, in order to still produce a distribution that looks "a lot like" a Gaussian distribution, despite the curvature of $\mathcal{M}$. More precisely, we choose to minimize the deviation of the "empirical center" $Z = \sum_{i=1}^{n_x} w_\sigma(i) y_i$ (estimation of expected value) of the importance sampled Gaussian (where we again define x not to be a neighbor of itself). That is, we want $\|x - Z\|_2$ to be small.

For small choices σ, however, we also expect *small* deviations, for instance. Therefore, we normalize by the standard deviation of a random weighted arithmetic mean $A = \sum a_i y_i$, which is $\sqrt{\mathbb{V}[A]} = \sqrt{\sum a_i^2 \mathbb{V}(y_i)} = \sigma\sqrt{\sum a_i^2}$, since the standard deviation of every "importance sampled" y_i is σ. We obtain the normalized deviation function

$$\Delta_x(\sigma) := \frac{\|x - \sum_{i=1}^{n_x} w_\sigma(i) y_i\|}{\sigma\sqrt{\sum_{i=1}^{n_x} w_\sigma(i)^2}}.$$

Let us analyze the limiting behavior of Δ_x for varying σ. Since $y_i \in N_x$, we will have $w_\sigma^u(i) \overset{\sigma\to\infty}{\longrightarrow} 1$ and consequently $w_\sigma(i) \overset{\sigma\to\infty}{\longrightarrow} \frac{1}{n_x} = \text{const}$. Moreover, $\sqrt{\sum_{i=1}^{n_x} w_\sigma(i)^2} \overset{\sigma\to\infty}{\longrightarrow} \sqrt{n_x \cdot \frac{1}{n_x^2}} = \frac{1}{\sqrt{n_x}} = \text{const}$. On the other hand, with probability 1 there is a unique nearest neighbor $y_\star$ of x, so for $\sigma \to 0$, we get $w_\sigma(\star) \overset{\sigma\to 0}{\longrightarrow} 1 = \text{const.}$, $w_\sigma(i \neq \star) \overset{\sigma\to 0}{\longrightarrow} 0 = \text{const.}$ and $\sqrt{\sum_{i=1}^{n_x} w_\sigma(i)^2} \overset{\sigma\to 0}{\longrightarrow} 1 = \text{const}$. Therefore, the σ in the denominator of Δ_x controls

the limiting behavior for both $\sigma \to 0$ and $\sigma \to \infty$, so that we always have $\Delta_x(\sigma) \overset{\sigma \to 0}{\longrightarrow} \infty$ and $\Delta_x(\sigma) \overset{\sigma \to \infty}{\longrightarrow} 0$. Hence, the part in between is the interesting part.

Figure 2.3 shows some typical graphs of Δ_x. We observe a local minimum, which we interpret as a good compromise between making use of enough information about the neighbors (not having too unbalanced weights) and minimizing the deviation from the center (relative to σ). We therefore choose

$$\sigma_\star := \text{ argument of first local min of } \Delta_x(\sigma)$$

as our kernel scale for the data point x (if such a minimum exists). Figure 2.3 illustrates this.

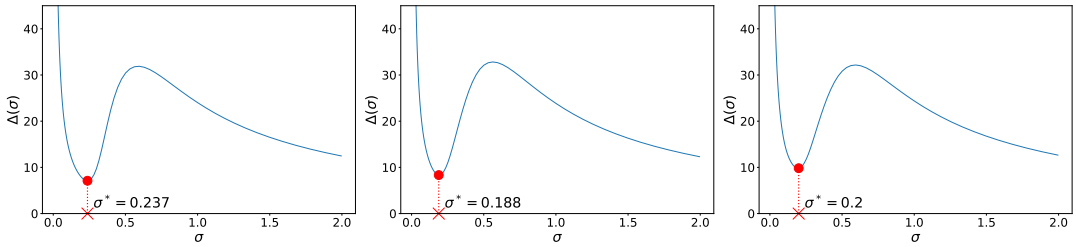


Figure 2.3.: Deviation function $\Delta_x(\sigma)$ for three examplary data points x of a dataset consisting of 1000 points uniformly drawn from a unit hypersphere with $d = 10$ embedded randomly in $\mathbb{R}^{15}$. We choose the argument of the local minimum of this function as the kernel scale (indicated in red).

We will see in section 2.4 that the heuristic works well at least for noise-free data. Unfortunately, this is currently not justified by theoretical results. For instance, the motivation with the standard deviation $\sigma\sqrt{\sum a_i^2}$ of random weighted arithmetic means $A = \sum a_i y_i$ cannot serve as an explanation, because in this situation, the weights a_i must be *fixed*. We deal with *random* weights $w_\sigma(i)$, however, so using $\sum_{i=1}^{n_x} w_\sigma(i)^2$ in the calculation of the standard deviation of $Z = \sum_{i=1}^{n_x} w_\sigma(i) y_i$, which does not make sense. (A vague conjecture is that we are normalizing by some approximation of the true standard deviation of $\|x - Z\|_2$, but this would need to be shown.) It can also not be explained, why a typical graph of Δ_x qualitatively looks like the ones in figure 2.3, in particular why the function almost always has a local minimum. Interestingly, leaving out any of the normalization factors in the denominator of Δ_x leads to functions that generally do not possess these good local minima.

A Heuristic for the Cutoff Radius ε

In practical applications of manifold learning, one often chooses not to compute the kernel values $k(x_i, x_j)$ for *all* $x_i, x_j \in X$, because this is computationally inefficient. It would require to compute $\mathcal{O}(n^2)$ kernel values, although most of the distances between the data points are not very meaningful or cannot be trusted anyway because

of the curvature of the underlying manifold $\mathcal{M}$. Therefore, one uses nearest-neighbor-methods (for more details, see appendix A). We choose an ε-nearest-neighbor-approach here, not computing all distances, but just the ones to the neighbors inside a ball of neighboring radius ε.

The radius ε is also a parameter to choose, however. We want to use ideas from a heuristic first described in [CSSS08] and refined in [BH16]. However, they use this heuristic for determining an appropriate scale for the kernel (which is σ in our case), and not as a cutoff radius, so we introduce some changes here. The basic principle is to compute the sum of all Gaussian kernel values $k_\varepsilon(x_i, x_j) = \exp\left(-\|x_i - x_j\|_2^2/\varepsilon\right)$:

$$T(\varepsilon) := \frac{1}{n^2} \sum_{i,j=1}^{n} k_\varepsilon(x_i, x_j) \approx \frac{\int_{\mathcal{M}} \int_{T_x\mathcal{M}} k_\varepsilon(x, y)\, dy\, dV(x)}{\mathrm{vol}(\mathcal{M})^2} = \frac{(4\pi\varepsilon)^{d/2}}{\mathrm{vol}(\mathcal{M})} \tag{2.8}$$

where the approximations are obtained by replacing the sum $\sum_{i,j=1}^{n} k_\varepsilon(x_i, x_j)$ with the mean value integral $\frac{n^2}{\mathrm{vol}(\mathcal{M})^2} \int_{\mathcal{M}} \int_{\mathcal{M}} k_\varepsilon(x, y) dx\, dy$ and locally approximating the manifold $\mathcal{M}$ by the tangent space $T_x\mathcal{M}$. Hence, $T(\varepsilon)$ approximates a term related to the growing volumes for growing ε of the small balls around the data points. Consequently, it should be well approximated by a power law $T(\varepsilon) \propto \varepsilon^a$ with ($a = \frac{d}{2}$ in this case). Rearranging yields $a = \frac{\log T(\varepsilon)}{\log \varepsilon}$. Because of our finite data situation, one instead observes $T(\varepsilon) \xrightarrow{\varepsilon \to 0} \frac{1}{n}$ and $T(\varepsilon) \xrightarrow{\varepsilon \to \infty} 1$. So for ε very small or very large, $T(\varepsilon)$ becomes flat and is monotonically increasing with higher slopes in between. This suggests to take the ε maximizing the slope of a, i.e. to maximize

$$\frac{d \log T(\varepsilon)}{d \log \varepsilon} \approx \frac{\log T(\varepsilon + h) - \log T(\varepsilon)}{\log(\varepsilon + h) - \log(\varepsilon)} \tag{2.9}$$

in order to choose the bandwidth so that the kernel has "maximum resolution". Note that this also yields a strategy for ID estimation using $a = \frac{d}{2}$, since (2.8) depends on d. This is somewhat related to the `Hein` estimator. Empirically, this simple approach does not seem to work very well, however.

Maximizing (2.9) is equivalent to maximize the slope of $T(\varepsilon)$ in a "loglog-plot". Since we want to use ε as a cutoff distance and not a bandwidth, we decide to choose it a bit larger, since computing more distances should not hamper the heuristics, but rather give a bit more information. For an appropriate choice of σ, these distances might not have a large influence anyways, if they are not needed. We just pay a slightly higher price regarding the computation time. To obtain qualitatively similar results, however, we maximize the slope of a similar function, namely of

$$\frac{dT(\varepsilon)}{d \log \varepsilon} \approx \frac{T(\varepsilon + h) - T(\varepsilon)}{\log(\varepsilon + h) - \log(\varepsilon)}$$

(with respect to ε). That is, we do not maximize the slope of T in the "loglog-plot", but in the "semilogx-plot" instead. This "stretches out" regions of higher function values in the y-direction and therefore increases the slope there. Empirically, this

results in an ε, which is a factor of about 3 higher, and achieves more precise results in the overall ID estimation algorithm.

One more important subtlety still needs to be resolved, however. To compute (2.8) one needs to know all distances between the data points (since we use the Gaussian kernel here), which we wanted to avoid in the first place. Hence, in the algorithm, we first perform a ($k = 100$)-nearest-neighbor search (or some other k, which is appropriate in the specific setting), compute the kernel values for all pairs of neighbors and then perform the ε-heuristic described above for just these kernel values in the sum (2.8). The hope is that despite the arbitrary choice of k, using the ε-heuristic still yields a more sensible choice of the cutoff parameter then just fixing an arbitrary k or ε.

2.3.4. Towards an Algorithm

In this subsection, after introducing a possible optimization step for the two strategies designed for obtaining a global ID estimate for all data points (if the ID is constant on $\mathcal{M}$), we will finally present the algorithm and its computational complexity.

Kullback-Leibler-Divergence Optimization

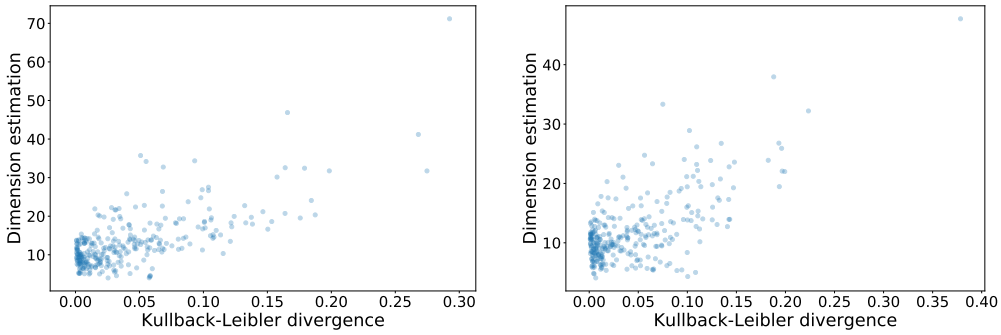
Both strategies presented in this chapter measure some statistics of an importance sampled Gaussian distribution. Sometimes, however, this does not succeed as well as desired, e.g. because a data point has few neighbors. Since for a good ID estimate we expect the manually assembled annulus bump function (used in the Strategy 1) to look similar to its corresponding analytical distribution, we want to compare them. We borrow a similarity measure between two probability distributions from information theory called Kullback-Leibler-divergence, which is defined as

$$\mathcal{KL}(p, q) := \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)}\, dx$$

for two probability measures p and q on $\mathbb{R}$, which is *not* symmetric. We always have $\mathcal{KL}(p, q) \geq 0$. High $\mathcal{KL}$-divergences mean very dissimilar distributions, while $\mathcal{KL}(p, q) = 0 \Leftrightarrow p = q$ almost everywhere.

Our strategy is the following: After having computed $\hat{d}_x$ for all $x \in X$, we compute for every x the Kullback-Leibler-divergences $\mathcal{KL}(\delta_x, \xi_{x,\hat{d}_x,\sigma_x^*})$ between the analytical annulus bump function $\xi_{x,\hat{d}_x,\sigma_x^*}$ corresponding to the estimate $\hat{d}_x$ and σ^* and the annulus bump function δ_x, which is obtained by a weighed kernel density estimation like in Strategy 1. Figure 2.4 illustrates the result of this procedure for two 10-dimensional exemplary datasets.

If we want to produce one global ID estimate for the whole manifold (if we can assume constant ID at all data points), then we therefore aim to improve our estimate by not taking into account the $\alpha \cdot 100\%$ of the points with highest $\mathcal{KL}$-divergence. α is



(a) $\mathcal{KL}$-divergences against ID estimates for the sphere dataset.

(b) $\mathcal{KL}$-divergences against ID estimates for the cube dataset.

Figure 2.4.: Scatter plots of the Kullback-Leibler-divergences (of the manually assembled annulus bump functions compared to the analytical distribution), plotted against the corresponding Strategy 2 ID estimates $\hat{d}_x$, for a sphere and a cube dataset. The sphere dataset consists of 300 data points, uniformly sampled from a 10-dimensional unit sphere randomly embedded in $\mathbb{R}^{15}$. The cube dataset was created in the same way. We see that the heuristic indeed tends to overestimate ID at points with high $\mathcal{KL}$-divergence.

a choice made by the user. Empirically, $\alpha = 0.5$ yields good results, that is, discarding the worse half of the local estimators for the global ID estimate.

Algorithm

As a first note, consider that often the intrinsic dimension is constant throughout the entire manifold $\mathcal{M}$. Therefore, we will compute a single number by either taking the mean value of all estimates $\hat{d}_x$, which will be denoted with a subscript `mean`, or by choosing a value that most estimates agree on, denoted by a subscript `vote`. In the latter case, since the estimates are real-valued, we perform an ordinary (unweighted) kernel density estimation with the estimates $\hat{d}_x$ of all points $x \in X$ and choose $\hat{d}$ so that the density of the estimators $\hat{d}_x$ is maximal there.

The algorithm with a Strategy 1 estimate shall be denoted by `GAP` (Gaussian annulus peak), and the second one with `GAV` (Gaussian annulus variance, since we compute a second moment of the importance sampled Gaussian kernel). All in all, we have the four different algorithms $\texttt{GAV}_{\texttt{mean}}$, $\texttt{GAV}_{\texttt{vote}}$, $\texttt{GAP}_{\texttt{mean}}$ and $\texttt{GAP}_{\texttt{vote}}$.

Moreover, let α denote the fraction of the data points, which is kept after the $\mathcal{KL}$-optimization (e.g. for $\alpha = 0.7$, the 30% of the points with highest $\mathcal{KL}$-divergence are discarded). Let $\text{perc}(A, \alpha)$ denote the α-quantile of a set $A \subset \mathbb{R}$ of real values. Then algorithm 1 is the pseudocode of the Gaussian annulus ID heuristics.

Algorithm 1 Gaussian Annulus Heuristic

procedure GAUSSIAN ANNULUS($X, k, r, \alpha, m_1, m_2, h$)
  Compute k-nearest-neighbor distances $\forall x \in X$
  $\varepsilon \leftarrow \arg\max_{\varepsilon_l} \frac{T(\varepsilon_{l+1})-T(\varepsilon_l)}{\log(\varepsilon_{l+1})-\log(\varepsilon_l)}$ for $\varepsilon_l = h^l$ with $l = -m_1, -m_1+1, \ldots, m_1-1, m_1$
  $r_{\text{cut}} \leftarrow \sqrt{\varepsilon \cdot r}$
  Compute ε-nearest-neighbor distances $\forall x \in X$
  for all $x \in X$ **do**
    for all $\sigma_l = \varepsilon \cdot h^l$, $l = -m_2, -m_2+1, \ldots, m_2-1, m_2$ **do** ▷ σ-heuristic
      $w^u_{\sigma_l}(i) \leftarrow \exp(-\|x - y_i\|_2^2/\sigma_l^2)$ for all neighbors $y_i \in N_x$
      $w_{\sigma_l}(i) \leftarrow w^u_{\sigma_l}(i) / \sum_{j=1}^{n_x} w^u_{\sigma_l}(j)$ for all neighbors $y_i \in N_x$
    end for
    $\sigma^* \leftarrow$ first local arg min over σ_l of $\|x - \sum_{i=1}^{n_x} w_{\sigma_l}(i) y_i\|_2 / \left(\sigma_l \sqrt{\sum_{j=1}^{n_x} w_{\sigma_l}(j)}\right)$ (if exists)
    if σ^* does not exist **then**
      Do not compute $\hat{d}(x)$, continue with next $x \in X$
    end if
    $\delta_x(r) \leftarrow$ weighted KDE on $\{\|x - y_i\|_2 : i = 1, \ldots, n_x\}$ with weights $w_{\sigma^*}(i)$
    if `GAV` **then**
      $\hat{d}_x \leftarrow \frac{1}{\sigma^*} \sum_{i=1}^{n_x} w_{\sigma^*}(i) \|x - y_i\|_2$
    else if `GAP` **then**
      $r_{\max} \leftarrow \arg\max_r \delta_x(r)$
      $\hat{d}_x \leftarrow (r_{\max}/\sigma^*)^2 + 1$
    end if
    $\gamma_x \leftarrow \mathcal{KL}(\delta_x, \xi_{x,\hat{d}_x,\sigma^*})$ ▷ For KL-optimization
  end for
  $\mathcal{D} \leftarrow \{\hat{d}_x : \gamma_x \le \text{perc}(\{\gamma_x : x \in X\}, \alpha)\}$ ▷ Discard estimations with high $\mathcal{KL}$-divergence
  if `mean` **then** ▷ Compute one estimator for whole $\mathcal{M}$
    $\hat{d} \leftarrow \frac{1}{\#\mathcal{D}} \sum_{x \in \mathcal{D}} \hat{d}_x$
  else if `vote` **then**
    $\nu(d) \leftarrow$ KDE of $\mathcal{D}$ with equal weights
    $\hat{d} \leftarrow \arg\max_d \nu(d)$
  end if
  return $\hat{d}$
end procedure

Computational Complexity

We do a naïve computational complexity analysis by tracking every costly step in the algorithm. Recall that the number of data points is denoted by n, the number of pairs of neighbors by N and the (extrinsic) dimension of the data by D. The number of different ε tried in the ε-heuristic is $\mathcal{O}(m_1)$ and there are $\mathcal{O}(m_2)$ different σ tried for every point. Moreover, we will denote the typical number of evaluations of kernel density functions with b (e.g. for the peak estimators or the $\mathcal{KL}$-optimizations).

First, a k-nearest-neighbor-search is performed, which is $\mathcal{O}(n \log n \cdot D)$. Performing the ε-heuristic is $\mathcal{O}(n \cdot k \cdot m_1)$, since there are k distances computed for every data point, which are used in $\mathcal{O}(m_1)$ summations of corresponding kernel values. With the

chosen ε, an ε-nearest-neighbor-search is performed, which is also $\mathcal{O}(n \log n \cdot D)$.

From here on, all computations just rely on the real-valued distances (or kernel values computed from these) between the data points, which are already computed and stored at this point. Hence, D does not play a role anymore from here on. Performing the σ-heuristic for all data points described above all in all uses the N computed distances for kernel evaluations $\mathcal{O}(m_2)$ times. Computing the actual ID estimates costs $\mathcal{O}(b \cdot N)$ kernel evaluations for kernel density estimations in `GAP` or just summing $\mathcal{O}(N)$ weighted distances for `GAV` . The same is true for computing the weighted KDEs $\delta_x(r)$ and the $\mathcal{KL}$-optimizations. Finally, sorting the $\mathcal{KL}$-divergences of all data points costs $\mathcal{O}(n \log n)$.

Therefore, the overall computational complexity is $\mathcal{O}(Dn \log n + nkm_1 + m_2 N + bN) = \mathcal{O}(n(D \log n + km_1) + N(m_2 + b))$.

2.4. Numerical Experiments

Let us now apply `GAP` and `GAV` to datasets with known ID and evaluate their performance in terms of precision. This section is divided into two subsections. We will first approach the presented algorithms from its qualitative behavior with regard to increasing ID and noise of the datasets in subsection 2.4.1. In the following subsection 2.4.2 we will test the presented heuristics on a large set of synthetic datasets and one real dataset and compare its performance to a variety of other ID heuristics mentioned in the review section 2.2.

For the implementation of the algorithm [vL18], Python and its NumPy and SciPy packages [JOP+01] were used, as well as scikit-learn [PVG+11] (mainly for nearest-neighbors-computations) and Matplotlib [Hun07] for all plots and visualizations. The weighted kernel density estimation was performed with the code at [Hof14]. Sparse data structures were employed whenever possible.

2.4.1. Increasing ID and Noise

We conduct two experiments to learn about the qualitative behavior of `GAP` and `GAV` for different intrinsic dimensions d and noise levels ρ of the datasets.

For the first experiment, we create datasets with 1000 data points uniformly sampled from the d-dimensional unit hyperspheres and -cubes for $d = 1, \ldots, 20$, embed them in $\mathbb{R}^{50}$ and rotate them randomly around the origin. We choose the sphere for the reason that it poses a difficulty for an ID heuristic because of its curvature, and the cube because of its boundaries and the fact that its mass is concentrated in the corners. We perform the ID estimations with $\texttt{GAV}_{\texttt{mean}}$ (the results of `GAV` and `GAP` are very similar in practice, while the mean variants perform better than the majority vote variants, which we will observe in subsection 2.4.2). The results are depicted in figure 2.5.

One observes a good fit with the identity line, which would be ideal estimates. The ID estimate of the sphere is always about 2 dimensions too high, which is an acceptable error in higher dimensions. Interestingly however, the heuristic fails to estimate the ID correctly just for the case $d = 2$, which is an outlier (which high statistical error)

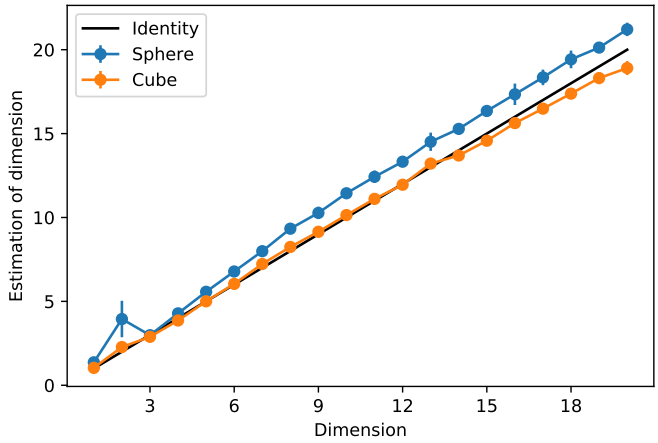


Figure 2.5.: Mean value of $\texttt{GAV}_{\texttt{mean}}$ ID estimates of the d-dimensional cube and sphere embedded in $\mathbb{R}^{50}$ respectively, for $d = 1, \ldots, 20$. The datasets consist of 1000 data points and no noise was added. Error bars are indicated (the datasets were sampled multiple times), as well as the identity line in black, on which perfect estimates would lie on.

for the low dimensions. The ID estimate of the cube on the other hand is relatively accurate. One may therefore conclude that datasets with intrinsic dimensions in the order of magnitude 10^1 can in principle be well handled by the Gaussian annulus approach.

As a second experiment, we choose the same unit sphere- and cube-datasets again, but with fixed $d = 10$. Additionally, we add Gaussian noise with standard deviation $\rho = 0, 0.05, 0.1$ and 0.2 to the datasets. Figure 2.6 shows the results.

Apparently, the ID estimation quickly tends to the extrinsic dimension $D = 50$ for increasing noise. A possible explanation is that the (full-dimensional) noise is interpreted as part of the manifold. Since the noise is full-dimensional, i.e. there is 50 times one-dimensional Gaussian noise with variance ρ, these distances quickly accumulate. One may therefore conclude that the Gaussian annulus approach (qualitatively) fails to estimate ID correctly for noisy datasets.

2.4.2. Benchmark Comparison

In this subsection, we want to evaluate the precision of all Gaussian annulus variants $\texttt{GAV}_{\texttt{mean}}$, $\texttt{GAV}_{\texttt{vote}}$, $\texttt{GAP}_{\texttt{mean}}$ and $\texttt{GAP}_{\texttt{vote}}$ in more detail. Moreover, we want to compare it to ID heuristics from the literature.

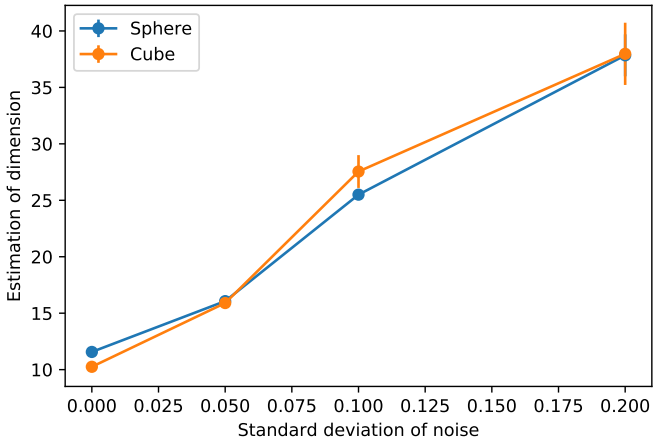


Figure 2.6.: Mean value of $\texttt{GAV}_{\texttt{mean}}$ ID estimates of the 10-dimensional cube and sphere embedded in $\mathbb{R}^{50}$ respectively, for different noise levels $\rho = 0, 0.05, 0.1, 0.2$. The datasets consist of 1000 data points. Error bars are indicated (the datasets were sampled multiple times).

Datasets

As a benchmark test set, we use all 21 synthetic datasets and one real-world dataset from the benchmark proposal of [CCCR15]. The synthetic datasets all consist of $n = 2500$ data points and are generated by randomly sampling from some manifold of known ID and embedding it linearly or non-linearly into some higher dimensional space. They do not contain any noise. See table 2.1 for an overview.

The datasets $\mathcal{M}_1$ up to $\mathcal{M}_{13}$, which are proposed by [HA05], are generated with a publicly available MATLAB tool [HA] of these authors. Moreover, six other datasets were generated with a tool also available as MATLAB code at [Lom] provided by the authors of [CCCR15]. This overall benchmark set also contains datasets with data sampled from a non-uniform probability density function. For more details on the datasets, see [CCCR15].

Moreover, a real-world dataset, namely the ISOMAP face database [TDSL00], was tested. It is denoted by $\mathcal{M}_{\text{faces}}$ in table 2.1. It consists of 698 gray-level images of size 64×64 depicting the face of a sculpture. This dataset has three degrees of freedom: two for the pose and one for the lighting direction. Therefore $d = 3, D = 64 \cdot 64 = 4096$. The ISOMAP face database is also used in the numerical experiments of chapter 3; the reader is referred to figure 3.16 for some exemplary images from this dataset.

Algorithms

On the datasets described above, additionally to the presented algorithms $\texttt{GAV}_{\texttt{mean}}$, $\texttt{GAV}_{\texttt{vote}}$, $\texttt{GAP}_{\texttt{mean}}$ and $\texttt{GAV}_{\texttt{vote}}$, we apply the following heuristics for a comparison: `Hein`,

Name	d	D	Description
$\mathcal{M}_1$	10	11	Uniformly sampled sphere linearly embedded.
$\mathcal{M}_2$	3	5	Affine space.
$\mathcal{M}_3$	4	6	Concentrated figure, confusable with a $3d$ one.
$\mathcal{M}_4$	4	8	Nonlinear manifold.
$\mathcal{M}_5$	2	3	$2d$-helix.
$\mathcal{M}_6$	6	36	Nonlinear manifold.
$\mathcal{M}_7$	2	3	Swiss roll.
$\mathcal{M}_9$	20	20	Affine space.
$\mathcal{M}_{10a}$	10	11	Unformly sampled hypercube.
$\mathcal{M}_{10b}$	17	18	Unformly sampled hypercube.
$\mathcal{M}_{10c}$	24	25	Unformly sampled hypercube.
$\mathcal{M}_{10d}$	70	71	Unformly sampled hypercube.
$\mathcal{M}_{11}$	2	3	Möbius band 10 times twisted.
$\mathcal{M}_{12}$	20	20	Isotropic multivariate Gaussian.
$\mathcal{M}_{13}$	1	13	$1d$-helix.
$\mathcal{M}_{13}$	24	96	Nonlinear manifold.
$\mathcal{M}_{N1}$	18	72	Nonlinear manifold.
$\mathcal{M}_{N2}$	24	96	Nonlinear manifold.
$\mathcal{M}_{\text{beta}}$	10	40	Beta distribution nonlinearly embedded.
$\mathcal{M}_{P3}$	3	12	Paraboloid, nonlinearly embedded.
$\mathcal{M}_{P6}$	6	21	Paraboloid, nonlinearly embedded.
$\mathcal{M}_{P9}$	9	30	Paraboloid, nonlinearly embedded.
$\mathcal{M}_{\text{faces}}$	3	4096	64×64-pixel grayscale sculpture images.

Table 2.1.: Descriptions of the manifolds of the datasets tested in this benchmark comparison. Up to $\mathcal{M}_{13}$, the table is copied from [CBR+14], table 2.

CD, Takens, MLSVD, MLE, MiND_{MLi}, MiND_{KL}, DANCo and DANCoFit.

In order to perform the tests, MATLAB code provided by the groups of the authors was used, which is provided at [HA] (Hein, CD, Takens), [Mag] (MLSVD) and [Lom] (MLE, MiND_{MLi}, MiND_{KL}, DANCo and DANCoFit).

Note that DANCoFit was not mentioned in the literature review section 2.2. DANCoFit is the name chosen in the package [Lom]. However, the author of this thesis assumes that this heuristic is the heuristic called FastDANCo in the literature, since FastDANCo is in fact DANCo with fitted functions to speed up the computation. The name DANCoFit suggests this and in fact, training DANCoFit first is necessary in the implementation. Moreover, it achieves much lower runtimes as reported in the description of the package available at [Lom] (which was the aim of FastDANCo).

See table 2.2 for the parameters used, which are always the default parameters provided in the code of the other authors. Note that we do not term an input variable as a "parameter", if it is clearly always favorable to choose a higher (or lower) value at the price of a higher computation time (like increasing m_2 just means trying more different σ-candidates). k always is the number of nearest neighbors of each data point used in the computations. For the GAV and GAP heuristics, however, k stands for the number of neighbor distances computed for each data point that were fed into the ε-nearest-neighbor-routine. For DANCoFit, the pre-trained parameters were used that are provided with the code. Finally, the minCorrectionDim-parameter for DANCo and DANCoFit is the minimum dimension, for which these heuristics are actually used. For lower dimensions, the output of the algorithm is just a pre-computed estimate with some other heuristic (which is otherwise used later in the algorithm). The default precomputation is done with MiND_{MLk}. It is indeed observed that DANCo and DANCoFit always agree with MiND_{MLi} (the integer variant of MiND_{MLk}), whenever the estimated dimension is $(\text{minCorrectionDim} - 1) = 4$ or lower.

Results

The results of all ID heuristics applied to all described datasets can be read off table 2.3.

Regarding the heuristics presented in this thesis, GAV and GAP seem to produce about equal results, but the way $\hat{d}$ is calculated from all single point estimators is important. Namely, the vote approach estimates ID lower than the mean approach. This indicates a skew in the distribution of the estimators, i.e. the distribution seems to have its density peak lower than its expected value.

One also notes that the heuristic is not naturally bounded by the extrinsic dimension D: E.g. for $\mathcal{M}_1$, i.e. the 10-dimensional sphere in 11-dimensional space, the estimates are higher than D. When this happens, this could easily be corrected, however.

Moreover, $\mathcal{M}_{\text{faces}}$ is an interesting case. Though under noise GAV and GAP are expected to perform badly and we have $D = 4096$, the rounded estimate is still 4 instead of 3 or 3 for GAP_{vote}. This observation can be made for all heuristics, however. Interestingly, the bad heuristics according to the error metrics (see below) perform better than all good heuristics for this real-world dataset.

Heuristic	Parameters
$\texttt{GAV}_{\texttt{mean}}$	$k = 100, r = 3.0, \alpha = 0.5$
$\texttt{GAV}_{\texttt{vote}}$	$k = 100, r = 3.0, \alpha = 0.5$
$\texttt{GAP}_{\texttt{mean}}$	$k = 100, r = 3.0, \alpha = 0.5$
$\texttt{GAP}_{\texttt{vote}}$	$k = 100, r = 3.0, \alpha = 0.5$
`Hein`	*None*
`CD`	*None*
`Takens`	*None*
`MLSVD`	#trails = 5
`MLE`	$k = 10$
$\texttt{MiND}_{\texttt{ML}}$	$k = 10$
$\texttt{MiND}_{\texttt{KL}}$	$k = 10$
`DANCo`	$k = 10$, minCorrectionDim = 5
`DANCoFit`	$k = 10$, minCorrectionDim = 5, provided train-file

Table 2.2.: Parameters of all used ID heuristics in the benchmark comparison. The parameters are further explained in the text.

We observe that the `DANCo`-heuristics achieve very promising results, even for high ID. We group them together in the following, because `DANCoFit` just tries to approximate `DANCo` to achieve a better runtime and we indeed just observe three datasets, where `DANCoFit` produces a different result, which is only one dimension off in these cases.

`MLSVD` also has an interesting behavior. If it does not fail completely as observed in a few cases, most often it is just exact. This is also the case for the high dimensional datasets, though `MLSVD` is not expected to work in these regimes, as it fails to estimate the dimension of a sphere incorrectly already for $d = 13$ as reported in [LLJM09]. Moreover, despite the original idea is looking for a "spectral gap" and hence this is not a fractal estimator (at least locally), the algorithm outputs a fractional number in exactly one case (the one with highest ID).

We also employ several error metrics to combine the results over the range of datasets into a single number for every algorithm. Let $d_{\mathcal{M}_i}$ be the ID of the dataset $\mathcal{M}_i$ and $\hat{d}_{\mathcal{M}_i}$ its estimation of some algorithm. Frequently used is the *mean percentage error* `MeanPE`, defined as

$$\texttt{MeanPE} := \frac{100}{\#\{\mathcal{M}_i\}} \sum_{i=1}^{\#\{\mathcal{M}_i\}} \frac{|\hat{d}_{\mathcal{M}_i} - d_{\mathcal{M}_i}|}{d_{\mathcal{M}_i}}$$

measuring the average difference of estimation and the true ID relative to the true ID. However, often it is more important to get the order of magnitude right. As an example, take a manifold $\mathcal{M}$ with $d_{\mathcal{M}} = 30$ and two estimations $\hat{d}_{\mathcal{M},1} = 60$ and $\hat{d}_{\mathcal{M},2} = 1.0$. While $\hat{d}_{\mathcal{M},1}$ is a somewhat mediocre estimation (wrong by a factor of 2),

name	N	d	D	$\texttt{GAV}_{\texttt{mean}}$	$\texttt{GAV}_{\texttt{vote}}$	$\texttt{GAP}_{\texttt{mean}}$	$\texttt{GAP}_{\texttt{vote}}$	Hein	CD	Takens	MLSVD	MLE	$\texttt{MiND}_{\texttt{ML}}$	$\texttt{MiND}_{\texttt{KL}}$	DANCo	DANCoFit
$\mathcal{M}_1$	2500	10	11	11.15	9.96	11.52	9.90	9	9.23	9.46	**10.0**	9.24	9	11	11	11
$\mathcal{M}_2$	2500	3	5	2.85	2.64	2.86	2.28	**3**	2.99	2.97	**3.0**	2.86	**3**	**3**	**3**	**3**
$\mathcal{M}_3$	2500	4	6	4.19	3.53	**3.99**	2.91	**4**	3.82	3.81	2.4	3.85	**4**	**4**	**4**	**4**
$\mathcal{M}_4$	2500	4	8	4.48	3.87	4.36	3.19	**4**	3.82	3.83	8.0	**3.91**	**4**	**4**	**4**	**4**
$\mathcal{M}_5$	2500	2	3	2.21	**2.00**	2.27	1.67	**2**	1.98	2.03	**2.0**	1.99	**2**	**2**	**2**	**2**
$\mathcal{M}_6$	2500	6	36	7.89	6.76	7.82	7.07	**6**	**5.73**	5.70	12.0	6.28	**6**	7	7	7
$\mathcal{M}_7$	2500	2	3	1.97	**2.00**	2.01	1.63	**2**	1.93	1.91	**2.0**	1.94	**2**	**2**	**2**	**2**
$\mathcal{M}_9$	2500	20	20	19.76	17.98	20.74	15.62	16	15.13	15.23	**20.0**	14.66	15	**20**	**20**	19
$\mathcal{M}_{10a}$	2500	10	11	10.16	8.79	10.34	8.39	9	8.78	8.64	**10.0**	8.21	8	11	**10**	**10**
$\mathcal{M}_{10b}$	2500	17	18	16.79	14.66	17.57	13.05	13	12.36	13.02	**17.0**	12.75	13	14	16	16
$\mathcal{M}_{10c}$	2500	24	25	22.87	21.08	24.01	17.70	17	17.06	17.25	**24.0**	16.95	17	**24**	**24**	23
$\mathcal{M}_{10d}$	2500	70	71	54.65	52.02	56.25	42.92	38	36.89	37.10	**70.2**	36.59	38	62	71	71
$\mathcal{M}_{11}$	2500	2	3	2.04	**1.99**	2.04	1.62	**2**	1.91	1.93	1.0	1.93	**2**	**2**	**2**	**2**
$\mathcal{M}_{12}$	2500	20	20	21.79	19.91	22.64	16.73	15	14.44	14.45	**20.0**	16.15	16	18	**20**	**20**
$\mathcal{M}_{13}$	2500	1	13	4.32	1.69	2.45	1.39	**1**	1.02	1.02	**1.0**	**1.00**	**1**	**1**	**1**	**1**
$\mathcal{M}_{N1}$	2500	18	72	18.34	17.36	19.03	13.92	14	13.66	13.78	**18.0**	13.75	14	15	**18**	**18**
$\mathcal{M}_{N2}$	2500	24	96	24.05	22.03	24.82	18.83	18	17.12	17.20	**24.0**	17.41	18	26	**24**	**24**
$\mathcal{M}_{\text{beta}}$	2500	10	40	7.64	6.04	7.77	5.01	4	3.51	3.56	**10.0**	6.01	6	7	7	7
$\mathcal{M}_{P3}$	2500	3	12	2.81	2.67	**2.93**	2.31	2	2.04	2.10	1.0	2.91	**3**	**3**	**3**	**3**
$\mathcal{M}_{P6}$	2500	6	21	**5.78**	4.74	5.70	4.15	2	2.31	2.31	1.0	4.96	5	5	**6**	5
$\mathcal{M}_{P9}$	2500	9	30	**7.95**	6.79	7.84	5.83	2	2.33	2.20	1.0	6.39	7	**8**	**8**	**8**
$\mathcal{M}_{\text{faces}}$	2500	3	784	4.46	4.13	4.31	**2.98**	**3**	3.37	3.43	1.0	3.71	4	4	4	4

Table 2.3.: ID estimations of several ID heuristics on the benchmark test set described above (rounded to two decimal places after comma). In each row, the best estimate is indicated in bold. If the best estimate was produced by an algorithm only estimating whole numbers, also the best continuous estimate is highlighted (since continuous heuristics have a much lower change of attaining the correct ID exactly). Additionally, the best heuristic of the four heuristics presented in this thesis are highlighted in red.

	$\texttt{GAV}_{\texttt{mean}}$	$\texttt{GAV}_{\texttt{vote}}$	$\texttt{GAP}_{\texttt{mean}}$	$\texttt{GAP}_{\texttt{vote}}$	Hein	CD	Takens	MLSVD	MLE	$\texttt{MiND}_{\texttt{ML}}$	$\texttt{MiND}_{\texttt{KL}}$	DANCo	DANCoFit
MeanPE	24.85	14.97	16.31	23.13	20.38	22.62	22.40	27.08	16.37	15.13	8.72	**4.93**	6.10
MeanGE	1.17	1.16	1.14	1.30	1.34	1.36	1.36	1.49	1.21	1.19	1.09	**1.05**	1.06
MedPE	5.64	11.77	5.34	22.28	15.00	18.29	18.93	**0.00**	17.65	18.33	9.17	**0.00**	**0.00**
MedGE	1.07	1.13	1.06	1.29	1.25	1.32	1.31	**1.00**	1.22	1.25	1.10	**1.00**	**1.00**
Exact	13	9	10	6	9	9	9	14	9	9	10	**15**	12

Table 2.4.: Error metrics described above for algorithms and datasets in the benchmark comparison in table 2.3. The overall lowest error in each row is printed in bold, the lowest error among the heuristics presented in this thesis in red.

$\hat{d}_{\mathcal{M},2}$ gives a completely wrong impression of the dataset, hence failed here and is not the preferred estimate in this case. Nevertheless, the percentage error would still favor $\hat{d}_{\mathcal{M},2}$. Therefore, we introduce the *mean geometric error* `MeanGE` as

$$\texttt{MeanGE} := \sqrt[\#\{\mathcal{M}_i\}]{\prod_{i=1}^{\#\{\mathcal{M}_i\}} \max\left\{\frac{\hat{d}_{\mathcal{M}_i}}{d_{\mathcal{M}_i}}, \frac{d_{\mathcal{M}_i}}{\hat{d}_{\mathcal{M}_i}}\right\}}$$

measuring the geometric mean of the ratio of true and estimated ID.

Another problem of the proposed error measures is the high sensitivity of these measures to outliers. Obviously, `MLSVD` sometimes completely fails the estimation, but when it is right, the estimation is often exact. To take this into account, we also use median variants `MedPE` and `MedGE` of the metrics above:

$$\texttt{MedPE} := \text{median}_i \left\{100 \cdot \frac{|\hat{d}_{\mathcal{M}_i} - d_{\mathcal{M}_i}|}{d_{\mathcal{M}_i}}\right\}$$

and

$$\texttt{MedPE} := \text{median}_i \left\{\max\left\{\frac{\hat{d}_{\mathcal{M}_i}}{d_{\mathcal{M}_i}}, \frac{d_{\mathcal{M}_i}}{\hat{d}_{\mathcal{M}_i}}\right\}\right\}.$$

Finally, let us also look at how many correct estimations the heuristics produced, when rounding all estimations to the closest whole number, so we define

$$\texttt{Exact} := \#\left\{i : \text{round}\left(\hat{d}_{\mathcal{M}_i}\right) = d_{\mathcal{M}_i}\right\}.$$

The error metrics for the benchmark datasets can be found in table 2.4.

We note that the `mean`-variants of `GAV` and `GAP` outperform most heuristics, except for the `DANCo`-heuristics and `MLSVD`, in all metrics except for the mean percentage error (which, as mentioned above, is sensitive to outliers and a geometric error is more sensible in a lot of cases) and $\texttt{MiND}_{\texttt{KL}}$ for the `MeanGE`-error. Further, the heuristics provided at [HA] (`Hein`, `CD` and `Takens`) are very similar in precision, with `Hein` slightly outperforming the other two, in particular its ball step function analog `CD`. The group of heuristics contained in the code [Lom] comes out above that group,

where one may approximately order them as MLE and $\texttt{MiND}_{\texttt{MLi}}$, then $\texttt{MiND}_{\texttt{KL}}$, then DANCoFit, then DANCo (in descending order with respect to error). MLSVD makes up an own category, being exact in the median – which is otherwise only achieved by the DANCo-heuristics – but having bad mean errors, since several complete failures worsen the average.

2.5. Discussion and Further Research

In this chapter, the problem of intrinsic dimension estimation was discussed. After presenting different approaches to assign an intrinsic dimension to a dataset X or just to single data points $x \in X$, the Gaussian annulus approach was motivated, leading to two concrete ID estimation strategies. Importance sampling served as the theoretical justification for Strategy 2. New heuristics for choosing the scale parameters of the strategies were presented, as well as a Kullback-Leibler-divergence-based optimization of the estimate. This was all merged into implementable algorithms, of which the computational complexity was analyzed. Moreover, experiments with qualitative as well as quantitative results (in subsections 2.4.1 and 2.4.2, respectively) were conducted.

Several problems remain unsolved and questions unanswered, both for GAV and GAP, as well as all other heuristics. Firstly, there is not one heuristic to this day fulfilling all desired properties of an ideal ID estimator, as mentioned in the beginning of this chapter. In the case of GAV and GAP, one could conclude that the presented estimators are *computationally feasible*, *accurate*, and *robust to high dimensionality*, but not *robust to multiscaling* and there was also no *operative range* established. More generally, there are multiple heuristics that are precise on noise-free data sets, especially the DANCo-heuristics. However, these are not robust to multiscaling, as they fail on noisy datasets, which is reported in subsection 2.4.1 or [CBR+14], respectively. At the very least, they have an inferior performance compared to MLSVD, which is very robust to multiscaling, since it is part of the construction of the algorithm (see [LLJM09]). The DANCo-heuristics also do not provide an operative range, i.e. a range where these estimators yield reliable results or concrete estimates on the errors. Using some regularity assumptions of the data, it should be possible to establish such estimates for GAV, however, possibly in a manner like the proof of 2.4.1. An analysis of the qualitative behavior of the number of data points needed for increasing dimensionality would also be of importance.

On the other hand, MLSVD has the problem that it is not robust to high intrinsic dimensionality, which the authors report themselves in [LLJM09]. Moreover, MLSVD fails on all real-world datasets in the benchmark test in [CCCR15].

Techniques on norm concentration alone will, however, most likely face overestimation problems in the presence of noise, since the noise is accumulated for all directions. Hence, trying to combine a multiscaling strategy like MLSVD, which is able to "split up" the noise into its single directions (the corresponding sigular values) with such a concentration technique would be interesting.

Another unsolved problem in this field is the ill-posedness of the problem with

regard to the acceptable noise level and the multiscaling problem. More theoretical and practical work would be required in order to be able to assign mathematically justified ID values to datasets or classify the cases where this is not possible.

Additionally, there is a number of possibilities one could try to further work with `GAV` and `GAP`. As mentioned in section 2.3.3, up to now there is no mathematically sound justification for the heuristic of the kernel scale σ. One may also explore other ways to choose a good scale σ for the estimators, e.g. for the reason that this heuristic is computationally demanding. A case where this choice would not even be necessary, would be the existence of a plateau in the function $\hat{d}(\sigma)$ of the ID estimation depending on the chosen scale. Something like this is observed in particular cases in [FdRL17]. Another possibility would be to choose the kernel scale using the Berry-Harlim-heuristic in the locally adaptive way described in [BH16] that we used as an inspiration for the global ε-heuristic for the nearest-neighbors-cutoff in this thesis. What would be necessary in terms of consistency would be to assume in the algorithm that the cutoff radius ε is always large enough with respect to the choice of σ so that the Gaussian annulus is "fully contained" in the ε-ball (see the analysis in subsection 2.3.2).

A promising try would be to combine the norm-based techniques in this thesis with an angle-based technique like `DANCo` does, in order to achieve even more precise results. Note that $\texttt{GAV}_{\texttt{mean}}$ and $\texttt{GAP}_{\texttt{mean}}$ had slightly better median error metrics in the benchmark test in section 2.4 than $\texttt{MiND}_{\texttt{KL}}$, which is just the "norm-part" of `DANCo`.

Since the Gaussian annulus approach in some sense also exploits that the volume of d-dimensional balls B_r grows like r^d, any equivalence statements with respect to other fractal estimators would be of theoretical interest.

What was assumed in this thesis is that the sample density q on the manifold $\mathcal{M}$ is locally well approximated by a uniform distribution in a linear subspace. More work would be required to be able to relax these conditions. For instance, one may try to adaptively reduce the reliability of estimators, where the curvature of the manifold is high. One would therefore need to employ curvature estimators of point clouds, for which there exist several ideas in the literature like in [Tau95] or [LP05] or by using pointwise distances to manifold approximation methods like the one in [Lev16]. To compensate for rough sample densities q, one could try to use an approach used in diffusion maps, where one divides out the density locally. This will be presented in section 3.2.

One remark to be made is that the author also tried to estimate ID by minimizing the $\mathcal{KL}$-divergence of the importance sampled Gaussian distribution to the annulus bump function ξ. This approach did not achieve superior results compared to `GAV` and `GAP`, however.

Finally, one may view the presented method using importance sampling as a more general framework. Importance sampling widens the range of statistics one may use on a dataset tremendously. It would be interesting, if different importance distributions could lead to new heuristics or yield improvements for other algorithms from the literature.

3. The Higher Harmonics Problem of Diffusion Maps

In this chapter, we will turn to manifold learning, i.e. the calculation of concrete low-dimensional parametrizations of our data with a particular algorithm, diffusion maps. We assume again that we are given data $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^D$, which lies on a compact d-dimensional manifold $\mathcal{M}$ and is sampled i.i.d. according to a density $q : \mathbb{R}^d \to \mathbb{R}$ with $\operatorname{supp}(q) \subset \mathcal{M}$. Neither $\mathcal{M}$, nor the density q, is known. Our task is to find a feature map $\Phi : \mathbb{R}^D \to \mathbb{R}^d$ (often $d \ll D$), such that

- Φ parametrizes $\mathcal{M}$
- $\|\Phi(x) - \Phi(y)\|_{\mathbb{R}^d}$ is an intrinsic distance of $\mathcal{M}$.

The diffusion maps algorithm is well suited to achieve this task and widely used in the literature. It approximates the Laplace-Beltrami-operator $\Delta_{\mathcal{M}}$ on the manifold (the generalization of the Laplace operator Δ in Euclidean space to general Riemannian manifolds) and yields its eigenfunctions as embedding coordinates. But for anisotropic data, i.e. data that extends more in some directions than others, we face a problem, which we will call the *higher harmonics problem* in the following. It occurs, because the eigenfunctions of the Laplace-Beltrami-operator are cosine-functions in this case, plus its higher harmonics. The latter are redundant coordinates, since our aim is to find a parametrization of our data of minimal dimension, termed as a *minimal diffusion maps embedding* in this thesis. An iterative approach to project out previously found coordinates will be presented.

This chapter is organized as follows. First, previous works regarding this problem are mentioned in section 3.1. In section 3.2 we will introduce the standard diffusion maps algorithm. We will then investigate the higher harmonics problem in more detail in section 3.3 where also a standard example case is shown. A novel approach of solving the higher harmonics problem is presented in section 3.4. The algorithm is tested and evaluated on synthetic as well as real world data in section 3.5. We conclude this chapter with section 3.6 by investigating possible directions of further research on the higher harmonics problem and its solutions.

Note the minor notation changes in this chapter: μ a probability measure now, p is used for a probability transition kernel, which will be introduced, and the letter d will be used for several distance metrics. Moreover, depending on the context, the notation for D will be slightly abused, using it not only for the extrinsic dimension of the data, but D_t or D_x will be either the *diffusion distance* with time parameter t (D_t) or a matrix containing the difference vectors $y_i - x$ for all neighbors $y_i \in N_x$ row by row (denoted by D_x).

3.1. Relevant Work

There is only very few work on the higher harmonics problem of diffusion maps, let alone attempts to resolve that issue.

Firstly, although this does not directly address the higher harmonics problem, the original diffusion maps paper [CL06a] describes a kernel that produces fast and slow directions of the diffusion. The authors use this kernel in order to find a function g which shall be constant on the level sets of a given empirical function f on the manifold, but less oscillatory than f. (For more details, see the motivation in section 3.4.) If one puts this strategy in terms of the higher harmonics problem, they try to fix it by transforming a higher harmonic to the lowest harmonic.

In [NLCK06], a paper from the same year, where the same authors also contribute, a diffusion maps framework is set up to compute reaction coordinates of dynamical systems. The higher harmonics problem is present in one of their synthetic examples but is neither treated as a problem, nor is it tried to resolve.

In [CGD+14], the higher harmonics problem is mentioned in the theoretical part, but not resolved. In the presented results of their application, the higher harmonics problem does not seem to appear.

As an ad hoc solution, in a conference talk [Kev17] is was proposed to study the set $\{(\psi_i(x), \psi_j(x)) : x \in X\}$ where ψ_i, ψ_j are diffusion maps coordinates on the data. That is, one e.g. shall plot two potentially dependent coordinates against each other. One should then check whether the graph is similar to the graph of a parabola like in figure 3.2(a). This would indicate a pair of a coordinate and a higher harmonic with twice the frequency, which we will see in section 3.3. If this is observed, the coordinate with higher frequency is discarded. However, this approach has the problem that for even higher harmonics than the ones with just twice the frequency, the graph looks different again. Additionally, if the intrinsic dimension of the data is high, one has to compare every new coordinate with all previously computed ones, which requires a lot of computation time.

For these reasons, an iterative approach is presented in this thesis, trying to completely remove already computed coordinates from the given manifold learning problem and computing a new coordinate.

3.2. Diffusion Maps

Diffusion maps is a manifold learning technique that aims to parametrize the data by constructing an $n \times n$ Markov chain transition matrix, which approximates a differential operator on the manifold. One then tries to extract the geometric information about $\mathcal{M}$ by studying the eigenfunctions of that operator.

Let $d\mu(y) = q(y)dy$ be our integration measure on $\mathcal{M}$. Then the standard diffusion maps construction works as follows.

1. Fix an isotropic, rotationally invariant kernel $k_\sigma(x, y) = g\left(\|x - y\|^2/\sigma\right)$. A common choice for g is $g(z) \propto \exp(-z)$, such that k_σ becomes the heat kernel. To

achieve compact support, e.g. for computational reasons (to obtain a sparse kernel matrix), we might also choose $g(z) \propto \exp(-z)\mathbb{1}_{\{z \leq \varepsilon\}}$.

2. Fix $\alpha \in \mathbb{R}$, let

$$q_\sigma(x) = \int_{\mathcal{M}} k_\sigma(x,y) d\mu(y)$$

be a smoothed approximation of the density $q(x)$. Assuming $q_\sigma(x), q_\sigma(y) > 0$, form the density normalized kernel

$$k_\sigma^{(\alpha)}(x,y) = \frac{k_\sigma(x,y)}{q_\sigma^\alpha(x) q_\sigma^\alpha(y)}.$$

3. Set

$$d_\sigma^{(\alpha)}(x) = \int_{\mathcal{M}} k_\sigma^{(\alpha)}(x,y) d\mu(y)$$

and normalize the kernel $k_\sigma^{(\alpha)}$ by defining the anisotropic transition kernel

$$p_{\sigma,\alpha}(x,y) = \frac{k_\sigma^{(\alpha)}(x,y)}{d_\sigma^{(\alpha)}(x)}$$

such that $\int_{\mathcal{M}} p_{\sigma,\alpha}(x,y) d\mu(y) = 1$, i.e. $p_{\sigma,\alpha}(x,y)$ is in fact a Markov transition kernel. That is, we can define a sequence of random variables $(X_l)_{l \in \mathbb{N}}$ that take values in $\mathcal{M}$ with transition probabilities

$$\mathbb{P}[X_{l+1} \in A | X_l = x] = \int_A p_{\sigma,\alpha}(x,y) d\mu(y).$$

4. Define the *transfer operator* $P = P^{(\alpha)}$ of the constructed Markov process to be

$$P^{(\alpha)} f(x) = \mathbb{E}[f(X_{l+1}) | X_l = x] = \int_{\mathcal{M}} p_{\sigma,\alpha}(x,y) f(y) d\mu(y).$$

Compute the first $m+1$ eigenvalues λ_k (i.e. $1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$) and eigenfunctions ϕ_k of $P = P^{(\alpha)}$ and study the diffusion maps embedding

$$\Phi(x) = (\lambda_1 \phi_1(x), \ldots, \lambda_m \phi_m(x)).$$

Note that we use $\lambda_0 = 1$ and ϕ_0 as a notation for the first pair of eigenvalue and eigenfunction of P here and discard them in the diffusion maps embedding. This is because we will assume a choice of the kernel k_σ so that $p_{\sigma,\alpha}$ is the transition kernel of a *reversible* and *irreducible* Markov process (X_l) (reversible follows from a symmetric

kernel k_σ). This implies that we will always have $\phi_0 \equiv 1$ with eigenvalue $\lambda_0 = 1$ since

$$P\phi_0(x) = \int_{\mathcal{M}} p_{\sigma,\alpha}(x,y) \cdot 1 \, dy = 1 = 1 \cdot \mathbb{1}(x) = \lambda_0 \phi_0(x) \quad \forall x \in \mathcal{M}$$

and, most importantly, one can show that $1 = \lambda_0 > \lambda_1 \geq \lambda_2 \geq \dots$ for the reversible and irreducible Markov chain. In the following, we will therefore call ϕ_1 or ψ_1 respectively the *first* eigenvector, eigenfunction, coordinate, etc. instead of ϕ_0 or ψ_0 and will ignore ϕ_0 and ψ_0 for the most part.

Finite Data

The approximation with data is straightforward. To obtain a sparse kernel matrix (corresponding to a kernel cutoff), one first performs an ε-nearest-neighbor search (see appendix A for more details). Since

$$z = \frac{\|x-y\|^2}{\sigma} \overset{!}{\leq} r \quad \Leftrightarrow \quad \|x-y\| \leq \sqrt{\sigma r},$$

one chooses $\varepsilon := \sqrt{\sigma r}$ as cutoff radius. All non-neighbor kernel values are set to zero. This corresponds to exchanging the ordinary Gaussian kernel for a Gaussian kernel with cutoff by multiplying with the step function $\mathbb{1}_{B_\varepsilon(x)}$, compare section 2.3.

Then the construction of the transition kernel p is slightly modified:

1. Form $K_{ij} = k(x_i, x_j)$ and compute the densities $d_i = \sum_{j=1}^n K_{ij}$
2. Choose $\alpha \in [0,1]$ and compute the normalized kernel matrix
$$K_{ij}^{(\alpha)} = \frac{K_{ij}}{d_i^\alpha d_j^\alpha}$$
3. Obtain a transition matrix $P^{(\alpha)}$ by row-normalization:
$$P_{ij}^{(\alpha)} = \frac{K_{ij}^{(\alpha)}}{d_i^{(\alpha)}}, \quad \text{where } d_i^{(\alpha)} = \sum_{j=1}^n K_{ij}^{(\alpha)}$$
4. Compute $\lambda_1, \dots, \lambda_m$ and corresponding eigenvectors $\psi_1, \dots, \psi_m$ of $P = P^{(\alpha)}$ and study the diffusion maps embedding
$$\Psi(x) = (\lambda_1(\psi_1)(x), \dots, \lambda_m(\psi_m)(x)) \quad \forall x \in X.$$
In other words, $\psi_1, \dots, \psi_m$ are the computed coordinates for the data points. That is, they can be viewed as real-valued functions $\psi : X \to \mathbb{R}$ defined on the data points that parametrize X.

Reasoning Behind the Diffusion Maps Construction

So far, we have established the diffusion maps procedure as the construction of a Markov chain on the dataset, where we chose the transition probabilities somehow related to the distances between data points. But does this Markov chain and its eigenfunctions, i.e. the embedding coordinates, encode any structure about $\mathcal{M}$?

Two answers can be given. The first one is that distances in the embedding correspond to a distance metric in the original Euclidean space called *diffusion distance*. From the construction one can show the existence of a stationary distribution π of $(X_l)_{l\in\mathbb{N}}$, so that for $l \to \infty$ the probability distribution of (X_l) converges to π. Then we may define:

Definition 3.1: *For $x, y \in \mathcal{M}$ and $t \in \mathbb{N}$, the* diffusion distance $D_t(x,y)$ *is*

$$D_t^2(x,y) = \|p_t(x,\cdot) - p_t(y,\cdot)\|^2_{L_2(\mathcal{M},\pi^{-1}d\mu)} = \int_{\mathcal{M}} (p_t(x,\cdot) - p_t(y,\cdot))^2 \frac{d\mu(u)}{\pi(u)}$$

where $p_t(x,y)$ is the transition kernel of P^t, i.e.

$$P^t f(x) = \int_{\mathcal{M}} p_t(x,y) f(y) d\mu(y).$$

The diffusion distance measures the difference between the probability distributions $p_t(x,\cdot)$ and $p_t(y,\cdot)$. Essentially, this is the overlap of the probability distributions of x and y when starting the Markov chain in x or in y respectively and running the chain for time t.

If P has an orthonormal basis (ONB) $\{\phi_k\}_{k=0}^{\infty}$ of eigenfunctions with corresponding eigenvalues $\{\lambda_k\}_{k=0}^{\infty}$ (which can be already derived from square-integrability of k), by some calculations (which can be found in [Ban17]), one finds

$$D_t^2(x,y) = \sum_{k=1}^{\infty} \lambda_k^{2t} (\phi_k(x) - \phi_k(y))^2.$$

We let $D_t^{(d)}(x,y)$ be the approximation of the diffusion distance by cutting of the infinite sum:

$$(D_t^{(d)})^2(x,y) := \sum_{k=1}^{d} \lambda_k^{2t} (\phi_k(x) - \phi_k(y))^2.$$

Therefore, we come full circle and arrive at diffusion maps again, since this shows that the algorithm is constructed in a way, so that

$$\|\Phi(x) - \Phi(y)\|_{\mathbb{R}^d} = D_t^{(d)}(x,y)$$

i.e. the distances in the embedding are the diffusion distances between the points on the manifold.

Hence we have related the construction of the Markov transition kernel in the dif-

fusion maps procedure to the diffusion distance. However, there is another deep connection between the coordinates of the diffusion maps embedding and the Laplace-Beltrami-operator $\Delta_{\mathcal{M}}$, which is the generalization of the standard Laplace operator Δ in Euclidean space to manifolds.

We recall that the transition matrix $P^{(\alpha)}$ is self-adjoint and the normalization condition implies that $P^{(\alpha)}\mathbb{1} = \mathbb{1}$. We now form the *graph Laplacian*

$$L^{(\alpha)} = \sigma^{-1}\left(P^{(\alpha)} - I\right).$$

Then one can show that the graph Laplacian indeed approximates the continuous Laplace-Beltrami-operator:

Theorem 3.2: *For an infinite amount of data and the kernel bandwidth σ going to 0, the graph Laplacian converges to the Laplace-Beltrami-operator, that is*

$$\lim_{\sigma\to 0}\lim_{n\to\infty} L_{ij}^{(\alpha)} f(x_j) = \Delta_{\mathcal{M}} f + (2-2\alpha)\nabla f \cdot \frac{\nabla q}{q}.$$

Note that if q is the uniform distribution or if we choose $\alpha = 1$, the second term vanishes (i.e. the density has no influence) and we get

$$\lim_{\sigma\to 0}\lim_{n\to\infty} L_{ij}^{(\alpha)} f(x_j) = \Delta_{\mathcal{M}} f.$$

Therefore, we can also state the convergence of the constructed Markov chain to Brownian motion or heat diffusion on the manifold, since the infinitesimal generator of Brownian motion is the Laplace operator. The infinitesimal generator of of a time-homogeneous Markov process shall not be exactly defined here, but can be thought of as the "expected time derivative" of that process, i.e. it describes the movement of the stochastic process in an infinitesimal time interval. Let $e^{-t\Delta_{\mathcal{M}}}$ denote the Neumann heat kernel, where $\Delta_{\mathcal{M}}$ has eigenfunctions ϕ_k. That is, ϕ_k verifies Neumann boundary conditions $\partial\phi_k = 0$ on the boundary $\partial\mathcal{M}$ of $\mathcal{M}$. Then the following holds:

Proposition 3.3: For any $t > 0$, the Neumann heat kernel $e^{-t\Delta_{\mathcal{M}}}$ can be approximated on $L^2(\mathcal{M})$ by $P_{\sigma,1}^{\frac{t}{\sigma}}$:

$$\lim_{\sigma\to 0} P_{\sigma,1}^{\frac{t}{\sigma}} = e^{-t\Delta_{\mathcal{M}}}.$$

For the proof, which is beyond the scope of this thesis, we refer to [CL06a] and to [HAVL05] for a more general treatment.

Therefore, intuitively, we study the structure of the manifold by constructing Brownian motion on it and then studying the eigenfunctions of the generator of this process. Furthermore, the Markov chain is constructed in a way such that no probability flows away from the "boundary" of the dataset. In particular, the mentioned eigenfunctions are the solutions to the Laplace equation $\Delta_{\mathcal{M}}\phi = 0$ with homogeneous Neumann boundary conditions, since the normal derivative at the "boundary" must be zero.

An interpretation why this strategy works on nonlinear manifolds (e.g. compared to linear methods like Principal Component Analysis or methods that use all distances between the data points like Multidimensional Scaling) is that only local distances are trusted and used in the algorithm because of the Gaussian kernel. Global (Euclidean) distances may be inappropriate to use and can be very different from e.g. geodesic distances which are more sensible, since the underlying manifold $\mathcal{M}$ of the data is nonlinear or curved in general.

3.3. The Higher Harmonics Problem

Diffusion maps works well in a lot of situations and yields good coordinates for a given dataset. However, if very anisotropic data is given, diffusion maps produces redundant coordinates because of higher harmonics. An example of this phenomenon is shown in figure 3.1, where the standard diffusion maps algorithm is applied to a typical synthetic dataset in manifold learning called *swiss roll*. This is a rectangle, winded up in space (so that two close points in Euclidean distance are not necessarily close in geodesic distance).

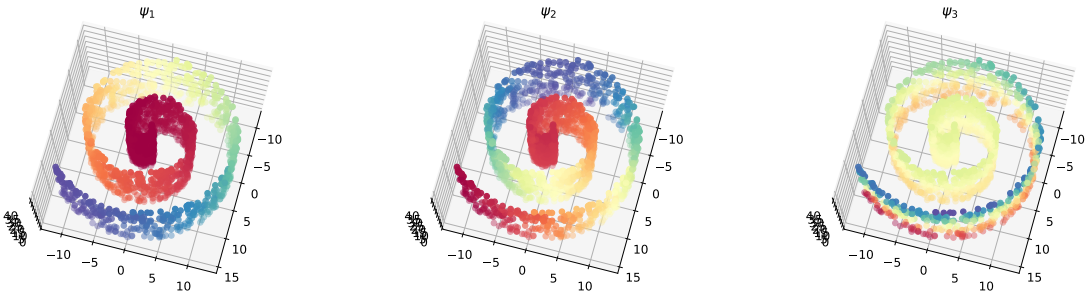
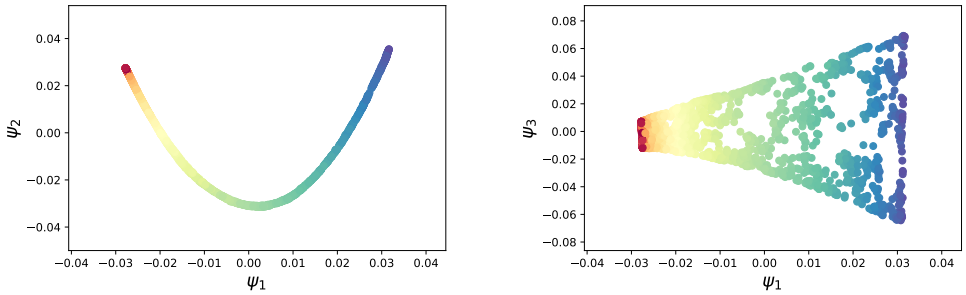


Figure 3.1.: Standard diffusion maps applied to a swiss roll dataset with 2000 data points, corrupted by Gaussian noise with standard deviation 0.1. The depicted first three coordinates are found by the standard diffusion maps algorithm with parameters $\sigma = 6.0$, $r = 5.0$, $\alpha = 1.0$. The color map indicates the function values of $\psi_i, i = 1, 2, 3$.

In this example, the first coordinate ψ_1 found by diffusion maps (i.e. $\lambda_1 \geq \lambda_2 \geq \dots$) nicely parametrizes the long direction of the rectangle, that is, the angular coordinate of the swiss roll. However, although ψ_1 is very different from ψ_2, the second coordinate ψ_2 parametrizes the same direction as ψ_1. This is because the wound-up rectangle is more than twice as long as wide. Therefore, this coordinate is effectively redundant.

Figure 3.2(a) shows the embedding of the swiss roll for $m = 2$ obtained from ψ_1 and ψ_2. Since these two coordinates parametrize the same direction, the embedded manifold is effectively one-dimensional, although the original manifold is two-dimensional. ψ_3, however, parametrizes the z-direction orthogonal to the angular direction of the

swiss roll. As a result, the combination of ψ_1 and ψ_3 yields a two-dimensional manifold, as shown in figure 3.2(b).



(a) ψ_1 plotted against ψ_2.

(b) ψ_1 plotted against ψ_3.

Figure 3.2.: Embeddings of the swiss roll with Gaussian noise (standard deviation 0.1) in $\mathbb{R}^2$. In 3.2(a), ψ_1 and ψ_2 computed by diffusion maps with parameters mentioned in figure 3.1 are chosen as embedding coordinates. Since they parametrize the same direction on the manifold, one obtains a one-dimensional manifold in $\mathbb{R}^2$. Choosing ψ_3 instead of ψ_2 as the second embedding coordinate yields a two-dimensional manifold.

We term this the *problem of higher harmonics*. Let us understand this effect, which also clarifies the choice of the name of this issue. As shown in theorem 3.2 and proposition 3.3, the constructed Markov process approximates Brownian motion on the manifold and the eigenfunctions computed by diffusion maps are the eigenfunctions of the Laplace operator with homogeneous Neumann boundary conditions.

If the domain is a rectangle (which would be the "unrolled case" of figure 3.1), these eigenfunctions ϕ of the Laplacian are well known (see e.g. [GN13]). Let $\Omega = [0, a_x] \times [0, a_y] \subset \mathbb{R}^2, l_{x,y} > 0$, then separation of variables yields

$$\phi_{k,l}(z_x, z_y) = \phi_k^{(x)}(z_x)\phi_l^{(y)}(z_x), \quad \lambda_{k,l} = \lambda_k^{(x)}(z_x) + \lambda_l^{(y)}(z_y)$$

where $\phi^{(x)}, \phi^{(y)}$ and $\lambda^{(x)}, \lambda^{(y)}$ correspond to the eigenfunctions in the one-dimensional case. Since we have Neumann boundary conditions, the ϕ's are cosine-functions:

$$\phi_k^{(z)}(z) = \cos(\pi k z / a_{(z)}) \tag{3.1}$$

This means that for very anisotropic data (without loss of generality, let the x-direction be the long one), the few first eigenfunctions are the constant 1-function for $k = 0$, the base cosine $\cos(\pi x / a_x)$ for $k = 1$ in x-direction (multiplied by the constant 1-function for the y-direction) and some *higher harmonics* of the x-cosine for a few higher k. That is, the cosine has more oscillations for higher k. Only when $\lambda_l^{(x)} < \lambda_1^{(y)}$ for the first time for some l, the first eigenfunction parametrizing the y-direction is found.

At this point we are also in shape to explain why the curve in figure 3.2(a) looks like a quadratic parabola arc. We know from equation (3.1) that in our example we are basically computing $\cos(x)$ and $\cos(2x)$ as the first two eigenfunctions. By addition theorems of the trigonometric functions, $\cos(2x) = 2(\cos x)^2 - 1$. Hence, setting $\cos(x) =: y$, an arc of a quadratic parabola is parametrized by the set

$$\{(y, 2y^2 - 1) : y \in [-1, 1]\} = \{(\cos x, \cos(2x)) : x \in \mathbb{R}\}.$$

Though we will not show this result for more general domains than the rectangle, we will in fact encounter this parabola arc a few times again in the numerical experiments, e.g. for a domain of an ellipsoid and even for a real-world dataset.

Summing up, this is an undesired effect. Ideally, the first k chosen coordinates should yield a full-dimensional manifold in $\mathbb{R}^k$. The higher harmonics problem, however, produces redundant coordinates. This either unnecessarily blows up our embedding, i.e. the the diffusion maps embedding is not *minimal*, or, if we choose to just compute $m = d$ coordinates, lets diffusion maps fail to parametrize the manifold. Unfortunately, it is a priori not clear how to detect higher harmonics automatically in all cases or even eliminate them completely.

Moreover, because of the different sampling densities in ψ_1-direction, we get a distorted embedding, when choosing the "right" coordinate ψ_3. This is apparent in figure 3.2(b): If ψ_1 and ψ_3 would be independent, we would get a rectangle, but we receive a trapezoid instead. This is because the data points are much more dense on the inside than on the outside, since they were sampled so that equal angles get an equal amount of data points, independent of the distance to the center axis. In theory, this should be compensated by dividing out the densities in the kernel, i.e. by using $K_{ij}^{(\alpha)} = \frac{K_{ij}}{d_i^\alpha d_j^\alpha}$ as entries of the kernel matrix. However, theorem 3.2 is a convergence result, i.e. the Laplace-Beltrami-operator is only exactly recovered for an infinite dataset.

This is also undesired, since even when choosing ψ_3 as the second diffusion maps coordinate, it is not independent from the first one.

Last but not least, even though there are good ID heuristics, it would be nice to have some stopping criterion intrinsic to the algorithm. Since diffusion maps is not expected to work for all possible datasets, this could be used to check, whether the whole dimension reduction procedure worked or not.

3.4. Minimal Diffusion Maps

In this section, a new approach for solving the higher harmonics problem is presented. The idea lies in subtracting the parts of the data points' difference vectors, which are already parametrized by the currently already computed part of the embedding, and then iteratively finding new parameters.

Motivation

Though with slightly different intentions, an anisotropic diffusion kernel already is introduced in [CL06a], namely

$$k_\sigma(x,y) = \exp\left(-\frac{\|x-y\|^2}{\sigma} - \frac{\langle \nabla f, x-y\rangle^2}{\sigma^2}\right),$$

where f is some empirical function on X, which is more oscillatory than desired. This kernel will favor points which are constant on the same level sets of f. It is shown in [Laf04] that by normalizing this to a Markov kernel p_σ, for $\sigma \to 0$ one obtains a diffusion which is constant along the level sets of f. Consequently, its first eigenfunction ϕ_1 is constant along the level sets and has few oscillations, since it is at the beginning of the spectrum.

Let us use this idea in the context of the higher harmonics problem. Suppose we already know the first nontrivial eigenfunction ϕ_1. We could now want to speed up the diffusion in the direction which is already parametrized by ϕ_1 and thereby change the spectrum of the Laplacian so that the first nontrivial eigenfunction of the old diffusion is *not* the first in the new diffusion. Therefore, by just calculating the first nontrivial eigenfunction $\phi_1^{(1)}$ of the new diffusion, we can also not get any higher harmonic of ϕ_1, since the higher harmonics come later in the spectrum and therefore we obtain a new non-redundant coordinate of the data.

Subtracting the Gradient Part

This speedup of the diffusion shall be achieved by updating the transition kernel appropriately after we know ϕ_1. In particular, we modify the distances between the data points by diminishing the part of the difference vectors $x-y$, which is already "explained" by ϕ_1. That is, we set

$$d^{(0)}_{\text{subt}}(x,y) = \left\| (x-y) - \left\langle x-y, \frac{\nabla_x \phi_1}{\|\nabla_x \phi_1\|} \right\rangle \phi_1 \right\|_2 \tag{3.2}$$

where $\nabla_x \phi_1$ denotes the gradient of ϕ_1 evaluated at x. This corresponds to making the diffusion in the direction parametrized by ϕ_1 infinitely fast, i.e. in expectation it takes no time to move in direction $\nabla\phi_1$. If x and y are not neighbors, we set $d^{(0)}_{\text{subt}}(x,y) = \infty$.

However, there is one more problem with this approach. In practice, we only have a finite dataset and need to calculate the gradient $\nabla_x \psi_1$ where ψ_1 is now just a pointwise defined real-valued function (as a coordinate of the data points). That is, we would like to compute infinitesimal changes of a function which is not defined on a continuous set. To circumvent this problem, we use the Nyström extension (see appendix A) to extend the pointwise defined function ψ_1 to a continuous one. Then we can estimate the gradient $\nabla_x \psi_1$ by just using a standard finite difference scheme (see appendix A).

Because of the noise in the data, curvature etc., we do not always have parallel gradients for neighboring points, i.e. for $y \in N_x$ in general $\nabla_x \psi_1 \nparallel \nabla_y \psi_1$. This

would cause $d(x,y) \neq d(y,x)$, which is undesirable (for example, symmetry is a useful assumption in correctness proofs of diffusion maps). To resolve this issue, we just symmetrize the distances by $d^{(0)}_{\text{loc}}(x,y) = d^{(0)}_{\text{loc}}(y,x) := \frac{1}{2}(d^{(0)}_{\text{subt}}(x,y) + d^{(0)}_{\text{subt}}(y,x))$.

Patching Together

After we have updated the local distances to take out the part already parametrized by ψ_1, we have one more issue to solve. Since we want to globally decrease distances in $\nabla\psi_1$-direction, it is possible that two points that are not neighbors now should have distance below the cutoff distance, i.e. we desire that $d^{(1)}(x,y) < \varepsilon$ (recall that $\varepsilon = \sqrt{\sigma r}$). But because we only compute and update the *local* distances, i.e. those of neighbors, for now we would still end up with $d^{(1)}(x,y) = \infty$. That is, we could obtain a new transition matrix P^α with e.g. $P^\alpha_{x,y} = 0$, i.e. x, y have transition probability zero, although they may lie on the same parameter line given by ψ_1.

A new distance metric that would not have this issue fulfills a triangle inequality in the sense that if $d^{(1)}(x,y) \leq \varepsilon$ and $d^{(1)}(y,z) \leq \varepsilon$, then $d^{(1)}(x,z) \leq 2\varepsilon$. This means that we need to assign finite distances to pairs of points which are not neighbors. Therefore, to glue the different neighborhoods together, we employ Dijkstra's algorithm with a cutoff. That is, we construct the weighted graph G from our local distance $d^{(0)}_{\text{loc}}(x,y)$ and compute all shortest paths between all pairs of points that have length $\leq \varepsilon$ where the length of a path is meant to be the sum of the weights of its edges (see appendix A for more details). Very useful is that it is possible to interrupt Dijkstra's algorithm, when the shortest paths emanating from one point x get longer than ε. This is because Dijkstra's algorithm computes the shortest paths to x's neighbors in ascending order (with respect to the lengths of the paths).

Since we are decreasing distances, the size of the neighborhoods will increase, as more point-pairs fulfill $d^{(0)}_{\text{loc}}(x,y) \leq \varepsilon$ than $d^{(0)}(x,y) \leq \varepsilon$. This unnecessarily increases the computational complexity, so we decrease ε to compensate for this in the following way. For any pair x, y we compute the ratio $d^{(0)}_{\text{loc}}(x,y)/d^{(0)}(x,y) \leq 1$, by which their distance decreases. The old cutoff distance $\varepsilon^{(0)}$ is then multiplied by the mean *ratio*, i.e. by the *geometric* mean: $\varepsilon^{(1)} = \varepsilon^{(0)} \cdot \sqrt[N]{\prod_{x\sim y\in X} d^{(0)}_{\text{loc}}(x,y)/d^{(0)}(x,y)}$ (where $\sim$ is the original neighbor relation of the data points). This accounts for the extent the distances have been decreased.

Finally, we choose the shortest-path-distances computed by Dijkstra's algorithm as the new "global" distance metric $d^{(1)}$ for our data points and apply the standard diffusion maps procedure again. In other words, to compute the next coordinate ψ_2, we form the kernel and transition matrix from the distance $d^{(1)}$, compute the first non-trivial eigenfunction, and iterate.

Residuals

In the minimal diffusion maps algorithm, we are successively subtracting distances between the points, which are parametrized by the computed diffusion maps coordinates. These distances are already "explained" by the parametrization of the data

up to that point. Therefore, analogously to the idea of *explained variation*, we may define the residual r_k currently left "unexplained" by our parametrization, as

$$r_0 := \sum_{x \sim y \in X} \|x - y\|_2$$
$$r_k := \sum_{x \sim y \in X} d_{\text{loc}}^{(k-1)}(x, y), \quad k \geq 1$$

where $\sim$ again denotes the original neighbor relation between the data points.

The hope is that if we choose to compute $m \geq d$ coordinates, we get $r_d \approx 0$ (having explained all distances in the data) or at least "convergence" of r_k for $k \geq d$ to some constant value (having explained all distances in the data possible by the minimal diffusion maps algorithm). We would then use these residuals as a natural stopping criterion. In section 3.5, it is investigated, how well this criterion works in numerical experiments.

Algorithm

Let us now summarize the derivation above and write down a concise algorithm. Since we have finite data, we can store the distance metrics, difference vectors etc. in matrices. We use the $\pm$ sign for a shorthand notation meaning both + and − (i.e. if we write $\overline{K^{\pm}}$ then there is a matrix $\overline{K^{+}}$ and a matrix $\overline{K^{-}}$). Let

- e_l denote the unit vector in dimension $1 \leq l \leq D$,
- the overline denote something related to a Nyström extension (see appendix A for more details). For instance, $\overline{\psi_k}(x \pm h e_l)$ denotes the extension of ψ_k to $x \pm h e_l$ (where $x \in X$),
- $\nabla \psi_k(x)$ denote the estimated gradient using central difference,
- S denote the matrix storing the current distances between the points,
- K denote the kernel and Π the diagonal matrix of estimated densities of the data points,
- D_x denote the matrix storing the difference vectors $y_i - x$ row-wise. As a preparation, note that $(D_x \cdot \nabla \psi_k(x)) \nabla \psi_k(x)^T$ is an extrinsic vector product (a matrix). For $\|\nabla \psi_k(x)\| = 1$, the rows of that matrix are exactly the gradient parts to be subtracted from the rows of D_x in the update step, compare (3.2). This vectorization speeds up the implementation.

We use $g(z) = \exp(-z)\mathbb{1}_{\{z \leq \varepsilon\}}$ as our symmetric positive definite kernel function again. Note that because of this choice, K, Π, S have few nonzero entries because of the cutoff and can be implemented efficiently using sparse matrices.

After that preparation of notation, a full overview on the Minimal Diffusion Maps algorithm `MDM` is given via the pseudocode below. Note that everything outside the

if-clause is just the vectorized version of the standard diffusion maps algorithm already described in section 3.2. Moreover, we use the notation $\Psi = (\lambda_1\psi_1, \ldots, \lambda_m\psi_m)$ for the *minimal diffusion maps embedding*, which is in general *not* the same as the standard diffusion map (m dominant eigenfunctions of the transition kernel).

Algorithm 2 Minimal Diffusion Maps (`MDM`)

procedure MINIMAL DIFFUSION MAPS($X, h, m, \alpha, \sigma, r$)
  Perform ε-nearest-neighbor-search to compute neighbors y_i of each point x
  $\forall x \in X$, initialize difference vector matrices D_x, distances $\|x - y_i\|_2$
  $\varepsilon^{(0)} \leftarrow \sqrt{\sigma r}$
  for all $k = 0, \ldots, m-1$ **do**
    if $k > 0$ **then** ▷ Compute gradients, update distances
      for all $x \in X$ **do**
        $\overline{K_x^{\pm}} \leftarrow \left(k(x \pm he_l, y_j)/(d_{x\pm e_l})^{\alpha}/(d_{y_j})^{\alpha}\right)_{\substack{l=1,\ldots,D \\ j=1,\ldots,n_x}}$, with $d_z = \sum_{z' \in N_z} k(z, z')$
        $\overline{\psi_k}(x \pm he_l) \leftarrow \overline{K_x^{\pm}} \cdot \psi_k \cdot \mathrm{diag}(\lambda_k^{-1})$
        $\nabla\psi_k(x) \leftarrow \frac{1}{2h}(\overline{\psi_k}(x + he_l) - \overline{\psi_k}(x - he_l))_{l=1,\ldots,D}^{T}$ ▷ Finite difference scheme
        $\nabla\psi_k(x) \leftarrow \frac{\nabla\psi_k(x)}{\|\nabla\psi_k(x)\|_2}$ ▷ Normalize gradient vector
        $D_x \leftarrow D_x - (D_x \cdot \nabla\psi_{k-1}(x))\nabla\psi_{k-1}(x)^T$ ▷ Update difference vectors
      end for
      $S \leftarrow (d_{\mathrm{subt}}^{(k-1)}(x_i, x_j))_{i,j=1,\ldots,n}$ with $d_{\mathrm{subt}}^{(k-1)}(x_i, x_j) = \|(D_{x_i})_{j,:}\|_2$ ▷ Local distances
      $S \leftarrow (d_{\mathrm{loc}}^{(k-1)}(x_i, x_j))_{i,j=1,\ldots,n} = \frac{1}{2}\left(S + S^T\right)$ ▷ Symmetrize local distances
      $\varepsilon^{(k)} \leftarrow \varepsilon^{(k-1)} \cdot \sqrt[N]{\prod_{x_i \sim x_j \in X} d_{\mathrm{loc}}^{(k-1)}(x_i, x_j)/d^{(k-1)}(x_i, x_j)}$ ▷ New cutoff distance
      $S \leftarrow (d^{(k)}(x_i, x_j))_{i,j}$ = Dijkstra lengths from $d_{\mathrm{loc}}^{(k-1)}$-weighted graph, cutoff $\varepsilon^{(k)}$
    end if
    $K \leftarrow (K_{ij})_{i,j}$ with $K_{ij} = g(d^{(k)}(x_i, x_j))$ ▷ Compute kernel
    $\Pi \leftarrow \mathrm{diag}(d_i^{\alpha})$ with $d_i = \sum_{j=1}^{n_i} K_{ij}$
    $K \leftarrow \Pi^{-1} K \Pi^{-1}$ ▷ Change influence of sample density
    $K_{ij} \leftarrow K_{ij} / \sum_{j=1}^{n_i} K_{ij}$ ▷ Row-normalize
    Solve $Kv = \lambda v$ for largest $\lambda < 1$
    Store $\psi_{k+1} \leftarrow v$, $\lambda_{k+1} \leftarrow \lambda$
  end for
  return $\Psi = (\lambda_1\psi_1, \ldots, \lambda_m\psi_m)$ ▷ Minimal diffusion maps embedding
end procedure

Computational Complexity

Let us again investigate the overall computational complexity of the `MDM` algorithm, which is always meant in terms of the optimal implementation possible.

Recall that the number of data points is denoted by n, the number of pairs of neighbors is N, the (extrinsic) dimension of the data is D and the number of coordinates to compute is m. Nearest-neighbor-algorithms building on k-d-trees have time complexity $\mathcal{O}(n \log n)$ and initialization of the distances costs $\mathcal{O}(N \cdot D)$. Then the same loop is iterated m times. For each point, we are extending the eigenvector to $2D$ points for the finite difference scheme, each requiring a kernel evaluation to the

neighbors, so there are $2DN = \mathcal{O}(N \cdot D)$ evaluations to be done. Then we do some matrix- and vector-multiplications, which is at most worth $\mathcal{O}(N \cdot D)$ when we use implementations for sparse matrices, which is also the amount of operations to calculate the distances from the updated difference vectors. Running Dijkstra's algorithm once using Fibonacci heaps [FT87] costs $\mathcal{O}(N + n \log n)$ here, and in one loop, we do this for each data point, which yields $\mathcal{O}(nN + n^2 \log n)$ operations (Dijkstra is done with a cutoff, so picking the last point before cutoff as the target node attains this time, all other neighbors are reached along the way). This is the most expensive part of the whole algorithm. Calculating the first two eigenvectors of a sparse matrix using matrix-vector-products can in principle be achieved in $\mathcal{O}(N)$ time. Therefore, the overall time complexity of the algorithm is $\mathcal{O}(mNn + mn^2 \log n + mND)$, which, for $n \geq D$ simplifies to $\mathcal{O}(mNn + mn^2 \log n) = \mathcal{O}(mn(N + n \log n))$.

The storage complexity, on the other hand, is $\mathcal{O}(N \cdot D)$, where the most expensive part is storing N difference vectors of dimension D.

3.5. Numerical Experiments

We will now evaluate the proposed minimal diffusion maps algorithm `MDM` for some synthetic and real-world datasets. We will always compare the results to the standard diffusion maps algorithm.

Like in chapter 2, Python was used for the implementation [vL18], as well as the NumPy and SciPy packages [JOP+01], scikit-learn [PVG+11] (mainly for nearest-neighbors-computations), Matplotlib [Hun07] for all plots and visualizations and the networkx package [HSSC08] for the implementation of Dijkstra's algorithm. Sparse data structures were used whenever possible.

3.5.1. Synthetic Data

The following synthetic datasets are constructed in a way to test the possibilities the algorithm provides, as well as to explore its weaknesses.

Swiss roll

First, we come back to the swiss roll from example from section 3.3. This dataset has become a standard benchmark dataset in the manifold learning community, since it is constructed so that small Euclidean distances do not necessarily mean small geodesic (i.e. intrinsic) distances because of the winding.

The results of the standard diffusion maps algorithm are depicted in figure 3.1, whereas figure 3.3 now shows the first 6 computed coordinates of the same dataset with `MDM`. The second coordinate ψ_2 now nicely parametrizes the short direction of the rectangle. Moreover, the embedding produced by `MDM` is not a trapezoid, but rather something much closer to a rectangle (see figure 3.4). It is very slightly sheared, but opposite sides have equal lengths.

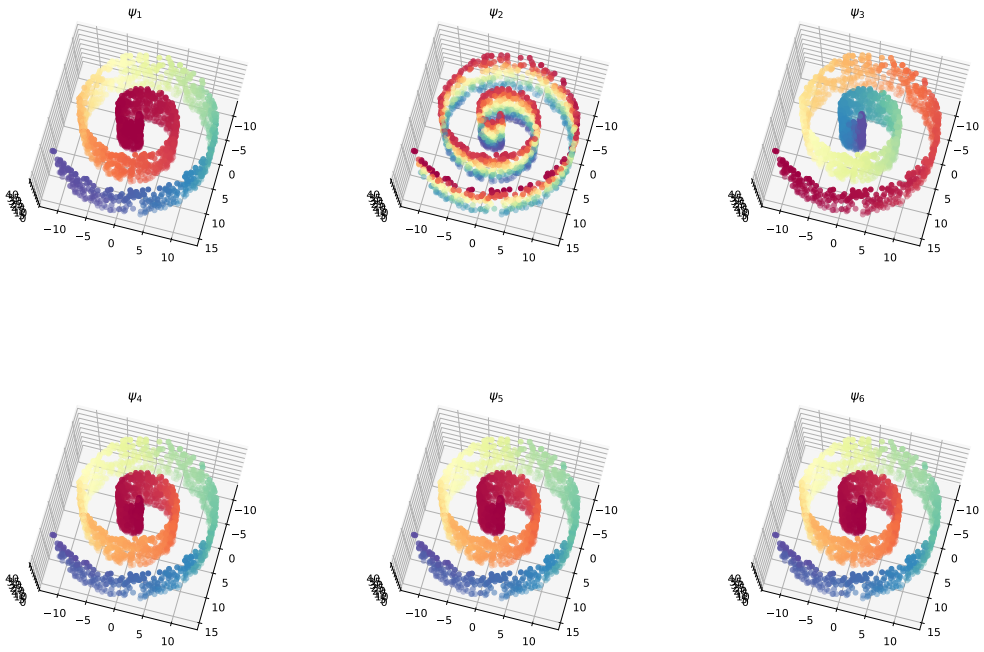


Figure 3.3.: The first six coordinates of the same swiss roll dataset with 2000 data points from section 3.3 computed with `MDM`. The same hyperparameters $\sigma = 6.0$, $r = 5.0$, $\alpha = 1.0$ were used, as well as $h = 0.05$ as a step size for the gradient computation. ψ_2 now parametrizes a new direction on the manifold with regard to ψ_1.

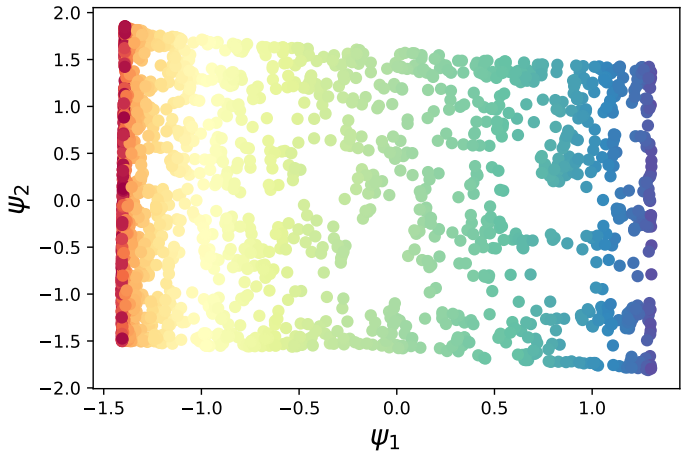


Figure 3.4.: Two-dimensional embedding of the swiss roll using the coordinates ψ_1 and ψ_2 depicted in figure 3.3.

We also see that unlike diffusion maps, we have some convergence behavior: Except from the different sign for ψ_3, all coordinates (except ψ_2) look completely the same. This is because `MDM` locally uses linear projections

$$\begin{aligned} P_b : \mathbb{R}^D &\to \mathbb{R}^D \\ z &\mapsto z - \langle b, z \rangle b \end{aligned} \tag{3.3}$$

projecting onto the hyperplane spanned by the normal vector b with $\|b\|_2 = 1$. In this case, b is the normalized gradient vector $b = \frac{\nabla_x \psi_k}{\|\nabla_x \psi_k\|_2}$. Once we have a distance $d^{(k)}$, such that the first diffusion maps coordinate computed from this distance is equal to some already existing coordinate, the situation will not change anymore, since projections do nothing when applied again by definition.

Let us additionally take a deeper look, how the global distances $d^{(k)}$ and the neighborhoods evolve during the steps of the algorithm. Figure 3.5 shows, how the neighborhoods and the global distances $d^{(k)}(x, \cdot)$ of one exemplary point x change for different k.

The neighborhoods behave as expected. In the $(k = 1)$-step of `MDM`, the distances $d^{(k)}(x, \cdot)$ are obviously decreased in the direction of the computed gradient $\nabla_x \psi_1$, which is the direction parametrized by the first coordinate, as one can see in figure 3.3. This results in an increase of the neighborhood into that direction.

Let us further look at the residuals r_k (see figure 3.6). We see that they do not decrease to zero but at least "converge". They unfortunately still change a bit after computation of the third coordinate ψ_3, but not afterwards. Therefore, the residuals may still serve as an indication for where to stop iterating `MDM` here.

A possible explanation, why the residuals fail to decrease to zero completely is because of the curvature of the swiss roll. Since we use the linear projections (3.3) locally, but the manifold has curvature, we do not resolve the local distances between the points as well as we would by collapsing the parameter lines of ψ_k to a single point. Once we compute a coordinate, which is already present in the current embedding, the algorithm stays stationary for the reasons described above and we get convergence of the residuals.

Narrow swiss roll

This dataset is basically the same as the previous one, except that the short direction is even shorter (25 instead of 40 length units). Therefore, the swiss roll is very anisotropic. We compare the standard diffusion maps algorithm and `MDM`.

The case of standard diffusion maps is handled by the algorithm as expected: The second direction of the swiss roll is not found as the second coordinate ψ_2, but only at ψ_6. Unfortunately, however, `MDM` completely fails to find another direction now. This dataset is too anisotropic even for `MDM`.

A possible explanation is that the distances in direction of ψ_1 can not be completely removed. Take two points $x \nsim y \in X$ that are not neighbors originally, i.e. that lie on different charts in the beginning, but that have $d^{(k)}(x, y) < \varepsilon^{(k)}$ for some $k \leq m$.

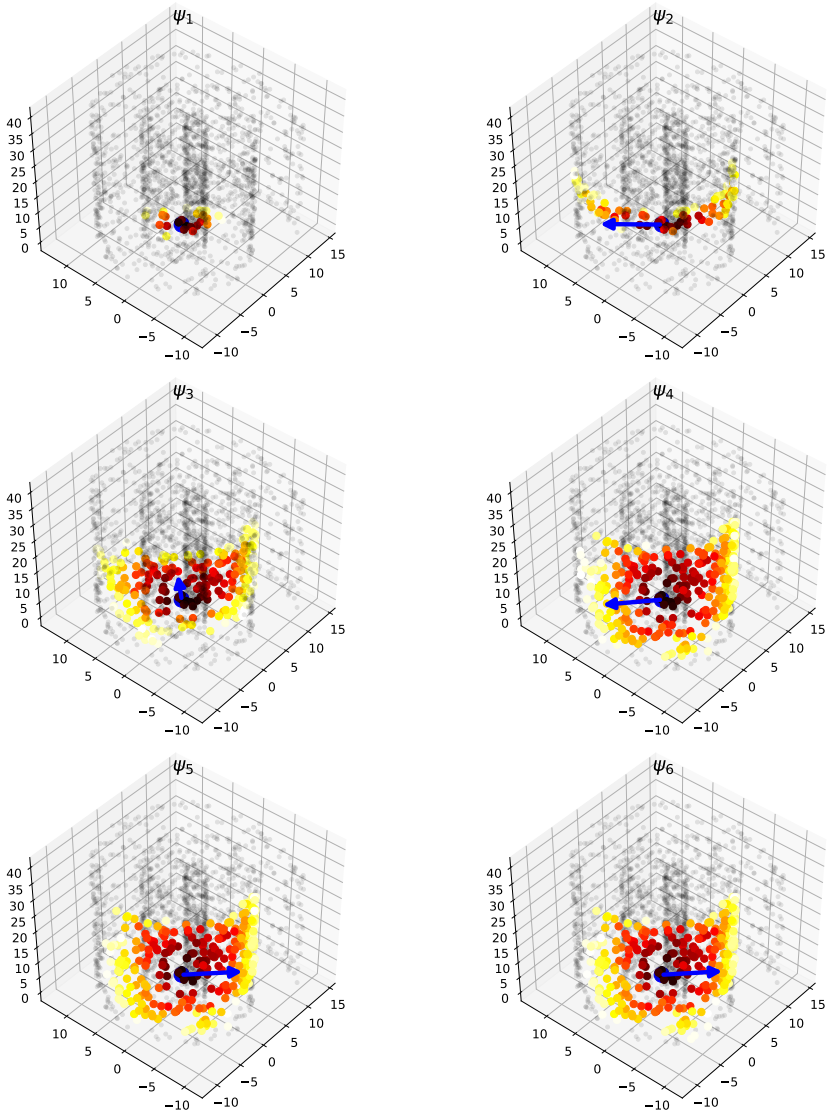


Figure 3.5.: Global distances $d^{(k)}$, neighborhoods and gradient of a single point for several steps of the `MDM` algorithm applied to the swiss roll dataset. The exemplary point is plotted very thick in blue, neighbors are plotted in color and thick, and all other points are plotted thin and black transparent. Small distances are indicated by dark colors, large distances by light colors. The first diffusion maps coordinate is computed from the distances in the first subplot etc. From the second plot on, the blue arrow indicates the gradient in the blue point from that previously computed coordinate. One clearly sees, how the neighborhoods extend into these gradient directions in the first steps, since the corresponding distances are decreased by `MDM`. They stay stationary after some steps.

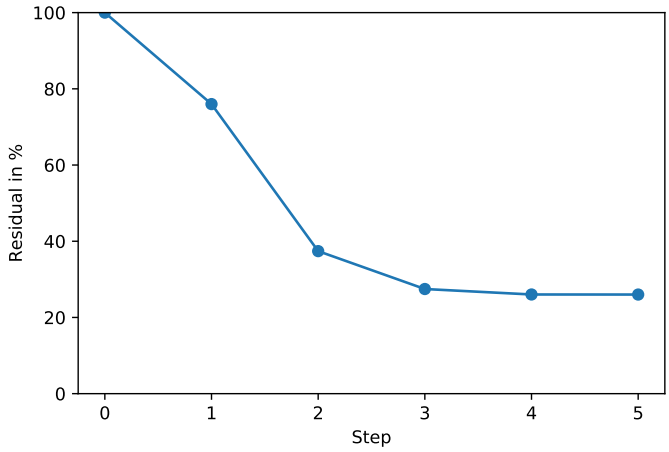


Figure 3.6.: Residuals of the `MDM` algorithm on the swiss roll in % of the sum of original distances. The residuals converge to a constant value.

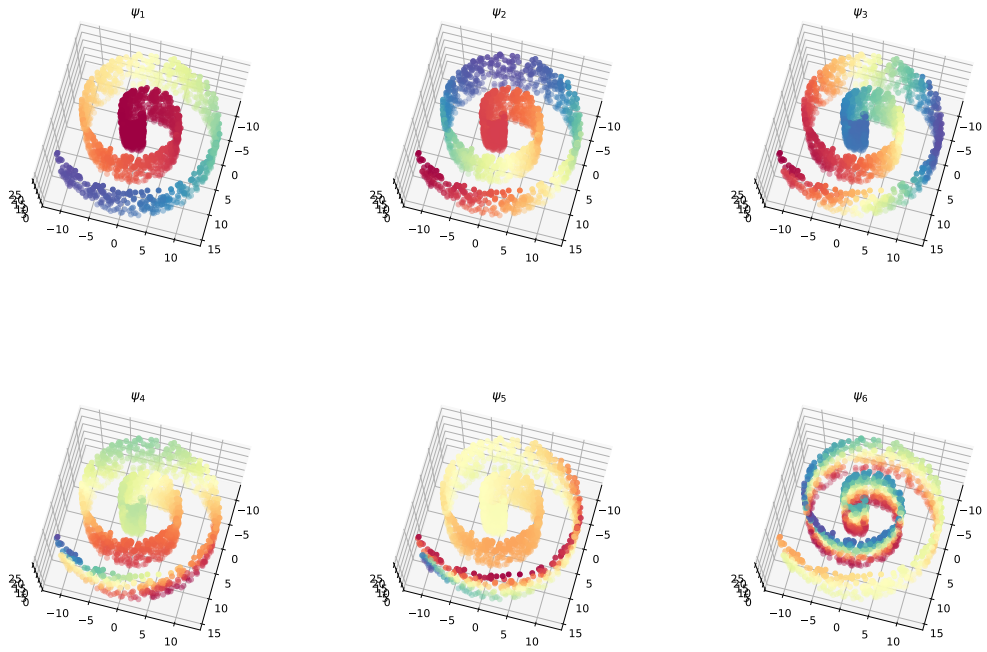


Figure 3.7.: Standard diffusion maps coordinates of the narrow swiss roll. The second direction in the manifold is now found even later than before.

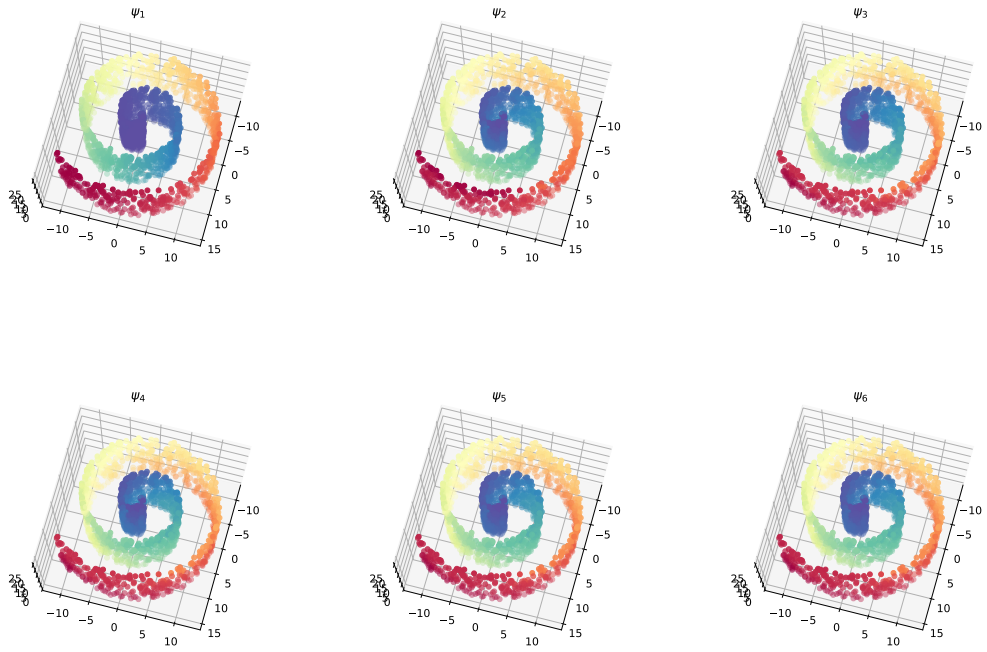


Figure 3.8.: `MDM` coordinates of the narrow swiss roll. `MDM` fails to find the second direction in the manifold at all.

Their distance $d^{(k)}(x, y)$ is the length of their shortest connecting path (in terms of weight). Since we only have finite data, the single segments of the path will, with probability 1, not point exactly in the direction $\nabla_x \psi_{k-1}$. Therefore, e.g. even if x and y might have the exact same intrinsic second coordinate (z-coordinate in this case), they are assigned a positive distance $d^{(k)}(x, y) > 0$ by `MDM`, since their connecting path also slightly extends into directions orthogonal to $\nabla_x \psi_{k-1}$, even though the path returns to the same parameter line. These extensions off the parameter lines count as distances in `MDM` and are accumulated over the path segments into the distance $d^{(k)}(x, y)$.

The longer these Dijkstra-paths get, the more of these "fake distances" are added. At some point this happens to the extent that $d^{(k)}(x, y') \geq \varepsilon^{(k)}$ for some $y' \in X$. Therefore, the neighborhood N_x of x is not extended along the whole parameter line of x given by ψ_k but is rather just stretched along the parameter line – a lot, but to some finite extent. This is already observed for the case of the first swiss roll in figure 3.5, where the second neighborhood is not a stripe along the whole swiss roll, but rather a long ellipse.

Therefore, the distance metric $d^{(k)}$ is not completely collapsed to zero in the gradient direction, but rather shortened a lot. If the dataset is so anisotropic that in the distance $d^{(k)}$ the shortened direction of the dataset is still longer than the others, because of the "fake distances", the first eigenvector ψ_k for this new distance will still parametrize the same direction as in the last step. If this happens, `MDM` is stuck and will compute the same coordinate again and again for the stationarity reasons

described above in the normal swiss roll case. This is why the stationarity described in the previous paragraph of the algorithm can also be a disadvantage; the standard diffusion maps algorithm at least finds the correct next coordinate for *some* k.

Note that, at least in the limit of a countably infinite number of data points or a continuous manifold, this does not happen. For more and more data, the "fake distances" will eventually decrease to zero, since there are points lying closer to the parameter line along the manifold. In other words, for an increasing amount of data, it should be possible to successfully compute a desired embedding with `MDM` for a more and more anisotropic manifold $\mathcal{M}$.

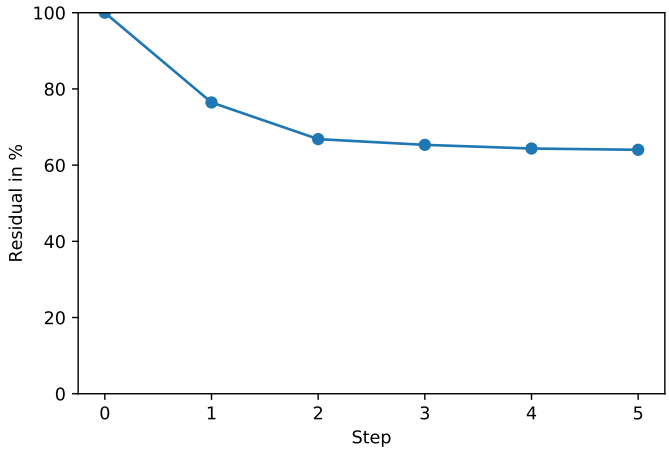


Figure 3.9.: Residuals r_k of `MDM` for the narrow swiss roll dataset. The convergence to a high value of above 60% indicates that the algorithm could not resolve a lot of distances in the dataset.

Moreover, we see in figure 3.9 that the residuals in this case converge only to a very high value, at least indicating that a lot of distances are not explained and something went wrong.

Ellipsoid

Next, we try a dataset with a different topology, namely a sphere with Gaussian noise with standard deviation 0.1. Moreover, we scale the sphere by a factor of 4 into the z-direction to get an anisotropic ellipsoid (note the scale on the z-axis compared to x and y in figure 3.12). The coordinates diffusion maps should find are called *spherical harmonics*.

Apparently, the standard diffusion maps algorithm again is unable to find a new coordinate in the data with the second eigenvector ψ_2, as shown in figure 3.10: ψ_2 again parametrizes the up-down-coordinate with twice the frequency, as expected. If we use ψ_1 and ψ_2 for a two-dimensional embedding, we get the familiar one-dimensional parabola arc (see figure 3.11) that we already encountered in section 3.3.

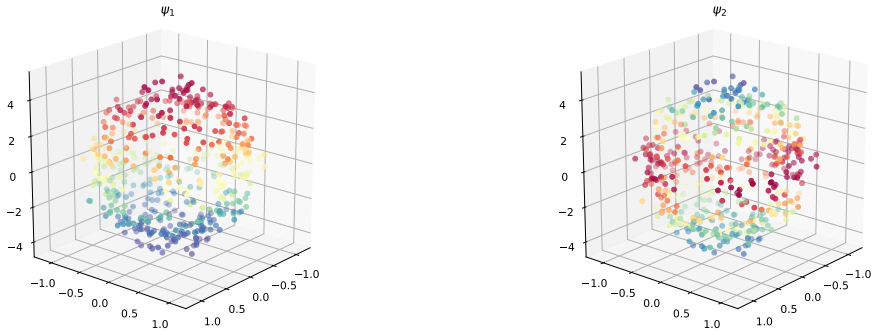


Figure 3.10.: First two diffusion maps coordinates ψ_1, ψ_2 for ellipsoid with $\alpha = 1.0$, $r = 3.0$, $\sigma = 1.0$. Note the scale of the z-axis. ψ_2 parametrizes the same coordinate like ψ_1.

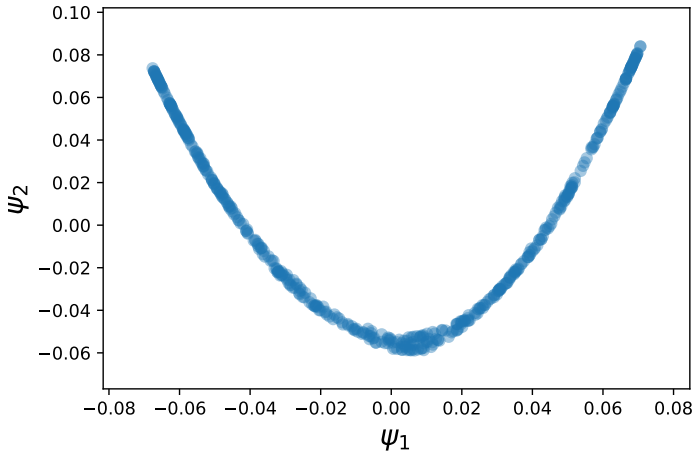


Figure 3.11.: Two-dimensional standard diffusion maps embedding of the ellipsoid. A one-dimensional parabola arc is obtained instead of something two-dimensional.

MDM instead produces coordinates like the spherical harmonics with the first three coordinates ψ_1, ψ_2, ψ_3, which parametrize orthogonal directions (see figure 3.12).

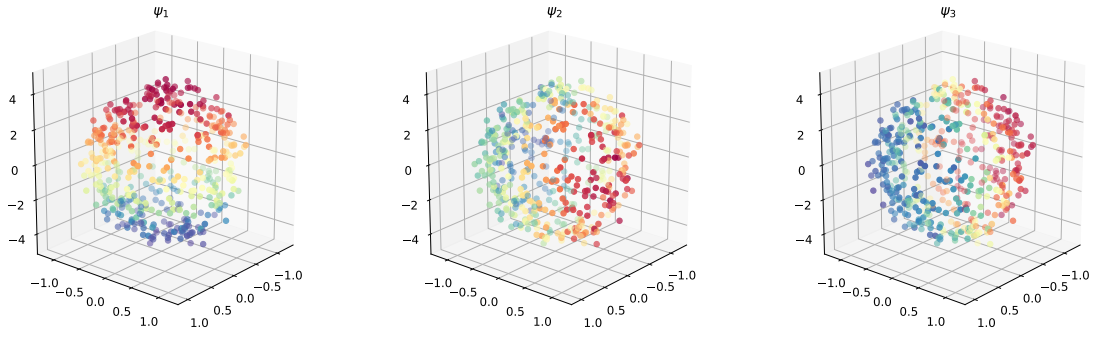


Figure 3.12.: First three coordinates of the ellipsoid produced by MDM. They parametrize the whole ellipsoid as desired.

As a result, the ellipsoid embedded in two dimensions is two-dimensional, and the embedding using ψ_1, ψ_2 and ψ_3 reproduces the topology of the original manifold (see figure 3.13(b)).

Let us take a look at the residuals in figure 3.14. They converge to a low constant value again. This time, the convergence is after the third step, although the ellipsoid is a two-dimensional manifold. This is because there is no embedding of the two-dimensional unit sphere in two dimensions, one needs at least three. Just using two coordinates yields a plane (see subfigure 3.13(a)).

5-dimensional sphere

To see if the MDM-procedure still works as automatically as in the ellipsoid case in slightly higher dimensions, we embed a 5-dimensional sphere in 10-dimensional space, randomly rotate it and see, if MDM can cope with that. Since the resulting coordinates are hard to visualize, we just investigate the residuals in figure 3.15.

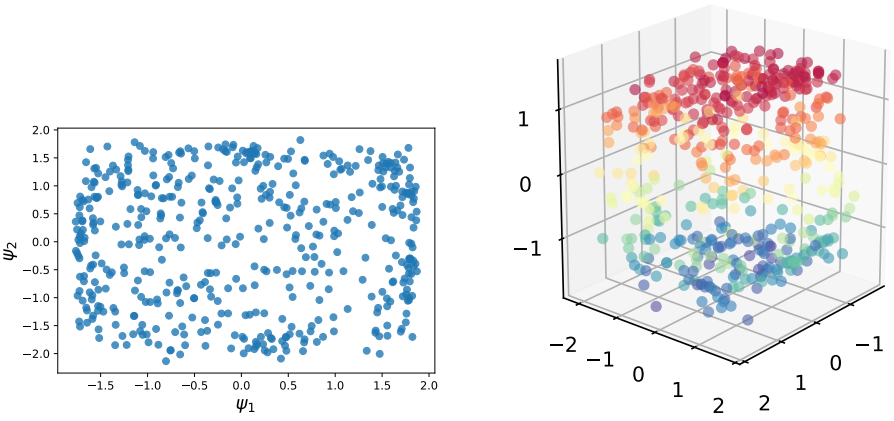
As mentioned in the previous paragraph, since there is no embedding of the d-dimensional sphere in $\mathbb{R}^d$, there are $5 + 1 = 6$ dimensions needed for a topology-preserving embedding of the sphere. Indeed, the residuals converge after the sixth step.

3.5.2. Real-World Data

In this subsection standard diffusion maps and MDM are applied to two real-world datasets.

The ISOMAP face database

We again consider the ISOMAP face database [TDSL00] that was already used in section 2.4. We recall that it has intrinsic dimension equal to three. Since every



(a) ψ_1 plotted against ψ_2.

(b) ψ_1, ψ_2, ψ_3 plotted against each other.

Figure 3.13.: Two- and three-dimensional embeddings of the ellipsoid. From subfigure 3.13(a) it is clear that ψ_1 and ψ_2 are independent. In subfigure 3.13(b), the three coordinates of the ellipsoid are plotted against each other. The topology of the ellipsoid is recovered. However, since the data points are again more concentrated on the boundary, the embedding looks more like a rounded cuboid than an ellipsoid. The colormap in 3.13(b) indicates the first coordinate ψ_1.

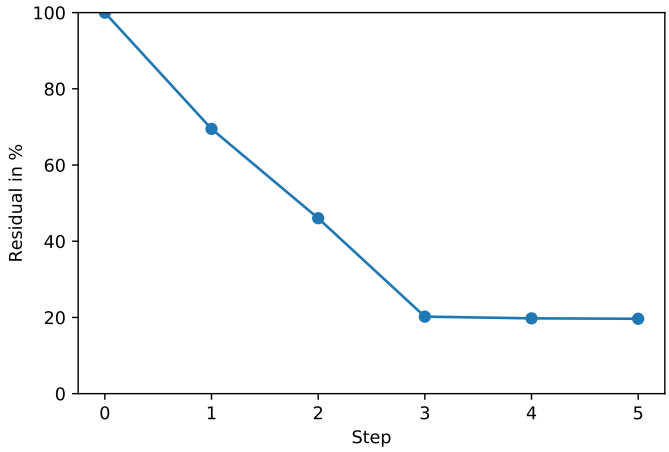


Figure 3.14.: Residuals when applying `MDM` to the ellipsoid. `MDM` converges after the third coordinate is computed.

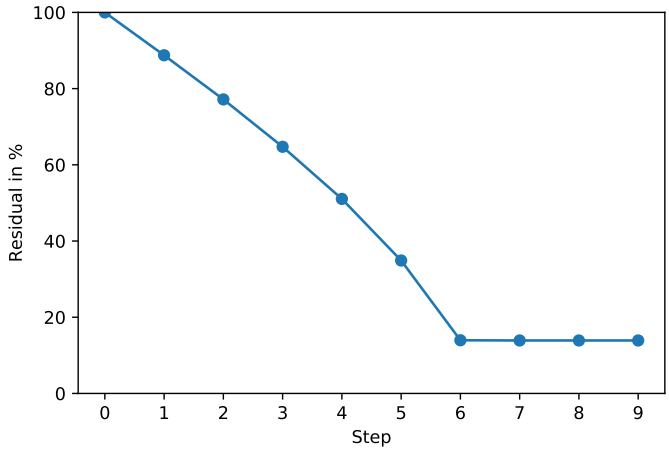


Figure 3.15.: Residuals when applying `MDM` to a 5-dimensional sphere embedded in $\mathbb{R}^{10}$. `MDM` converges after the sixth coordinate is computed.

data point has a concrete meaning (the image associated to it), the results are hard to visualize. Therefore, we just try to embed the datasets in two dimensions using standard diffusion maps and `MDM`.

Moreover, since the Nyström extension and finite differences for the gradient computation was computationally intractable in $64 \times 64 = 4096$ dimensions (extrinsic dimension D of the dataset), the images were downscaled to $32 \times 32 = 1024$ pixels by averaging all distinct 2×2-blocks of them into one pixel each. The results of standard diffusion maps and `MDM` are compared in figure 3.16.

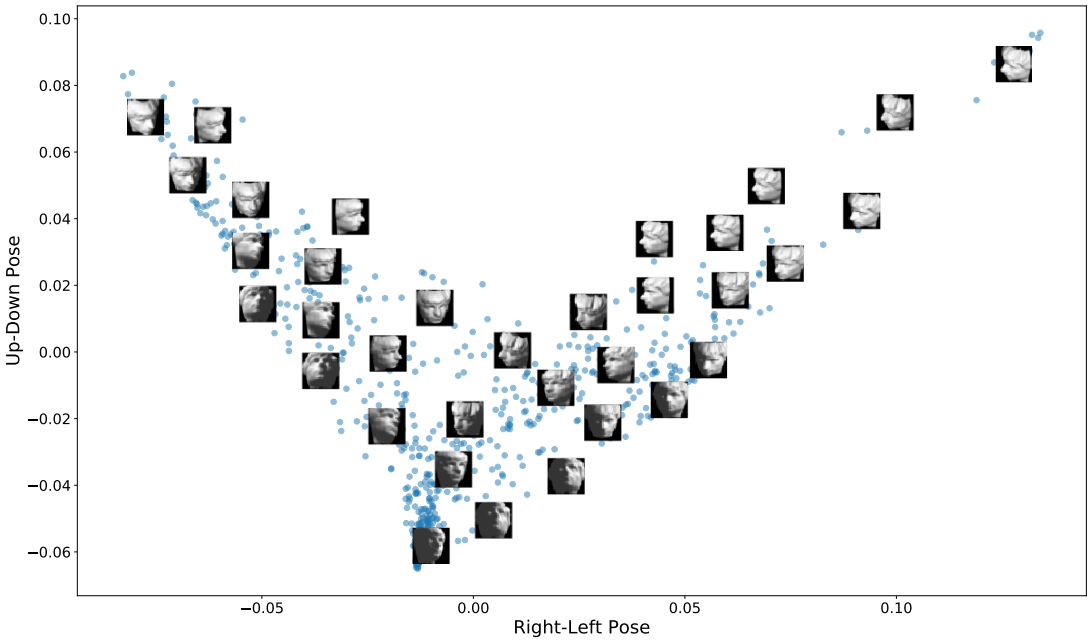
We see that `MDM` is again able to produce a reasonable embedding, somewhat parametrizing the two different angles of rotation of the images. This is also achieved by the standard diffusion maps algorithm.

Improvements over standard diffusion maps are only slightly made in terms of how widely spread out into two dimensions the data points are. Both embeddings have a bulk of data points concentrated on the bottom that seem to represent darker pictures. But some error in the embedding is expected, since we cannot embed the three-dimensional face dataset in two dimensions.

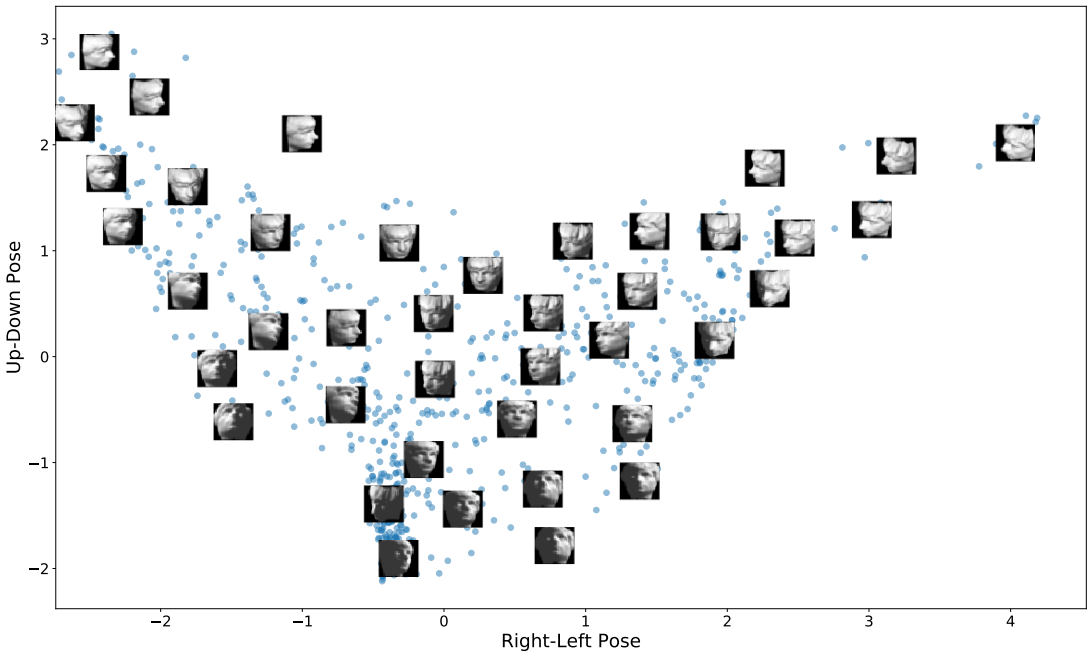
The digits dataset

As a second real-world example, we employ the digits dataset from [PVG+11], which is the test set part of the NIST digits dataset from the UCI Machine Learning repository [DKT17]. It consists of gray-scale images of handwritten digits with $8 \times 8 = 64$ pixels each. The whole dataset contains 1797 images in total (≈ 180 for every digit), where each pixel only attains values in $\{0, \ldots, 16\}$.

We just use the two sets of fours and fives in our case, since they allow for a good comparison of standard diffusion maps and `MDM` with respect to the higher harmonics



(a) Two-dimensional embedding computed with standard diffusion maps.



(b) Two-dimensional embedding computed with `MDM`.

Figure 3.16.: Two-dimensional embeddings of the downscaled ISOMAP face database using standard diffusion maps or `MDM`, respectively. Small thumbnails are shown for some of the images in the corresponding area of the embedding.

problem. For each of these, we compute two coordinates using standard diffusion maps and MDM, see figures 3.17 and 3.18, respectively.

In the case of the fours, we observe a more or less one-dimensional embedding for the standard diffusion maps algorithm apart from some outliers. MDM produces an embedding much more stretched out into two dimensions.

For the fives, we observe the parabola arc once again, which was observed a few times before when standard diffusion maps produced a redundant second coordinate ψ_2. MDM instead produces an embedding with ID equal to 2 using two coordinates.

3.6. Discussion and Further Research

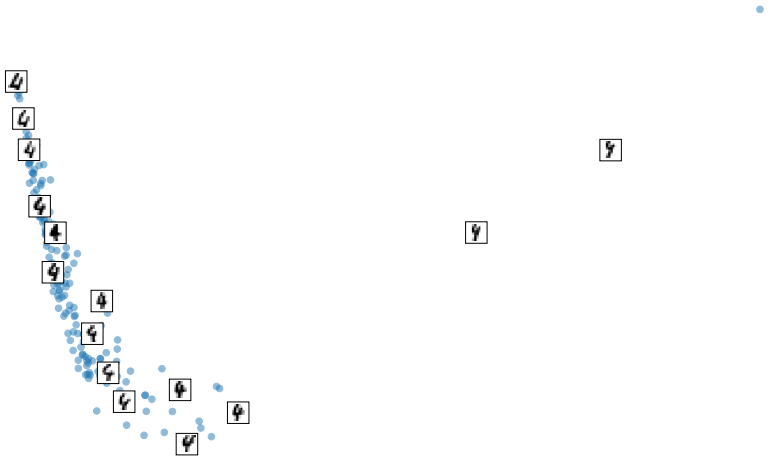
This chapter dealt with the higher harmonics problem of diffusion maps. After mentioning the few work on this subject, we first introduced the diffusion maps construction and considered the ideas and theory behind it. We could then discuss the higher harmonics problem in some more detail. Next, the Minimal Diffusion Maps algorithm MDM was presented after motivating the chosen general approach. Numerical experiments with both synthetic and real data yielded promising results. This was only with respect to the higher harmonics problem, which could often be resolved, but it also sometimes improved the quality of the embedding and the residuals.

There are several spots to improve upon and to extend the research to, however. In the case of the narrow swiss roll, using Dijkstra's algorithm like in MDM was problematic and the algorithm did not converge to a desired result. A closer theoretical analysis with regard to the eigenfunctions of the Laplace-Beltrami-operator $\Delta_{\mathcal{M}}$ in a similar manner like in section 3.3 could reveal more insights into the higher harmonics problem. For instance, in the swiss roll case one can imagine that there are at least as many harmonics of the long direction ψ_1 as the ratio $\frac{a_x}{a_y}$ of the lenghts of the rectangle. If something like this is true, there may also be results for more general domains.

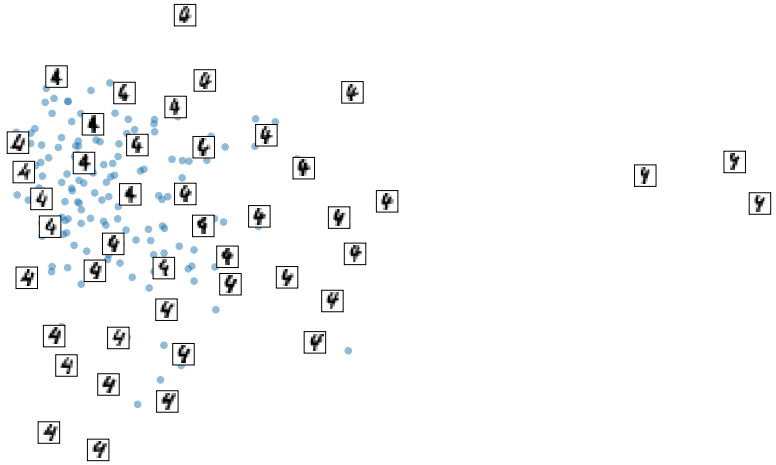
Also interesting would be other ways to remove a coordinate from the manifold, like performing a curvilinear projection along the parameter lines of the Nyström extended coordinate ψ_k. This would remove the "fake distances", which Dijkstra is adding in MDM and therefore increase overall precision, but also, most importantly, eliminate the problems with highly anisotropic data. This might also be a spot to speed up the algorithm, since the Dijkstra-part of MDM is computationally demanding.

Moreover, there may be better ways to compute gradients than using a Nyström extension and a finite difference scheme. E.g. one could investigate, whether this is possible by just using the values $\psi_k(y_i)$ of the neighbors $y_i \in N_x$ instead of extending ψ_k to other points.

The author is also curious what would happen if the idea of just iteratively computing one coordinate and projecting it out instead of trying to obtain all at once would be applied to other manifold learning algorithms. Possibly this strategy is advantageous, since it may remove nonlinear dependencies between the coordinates and allows for the presented way to measure residuals.

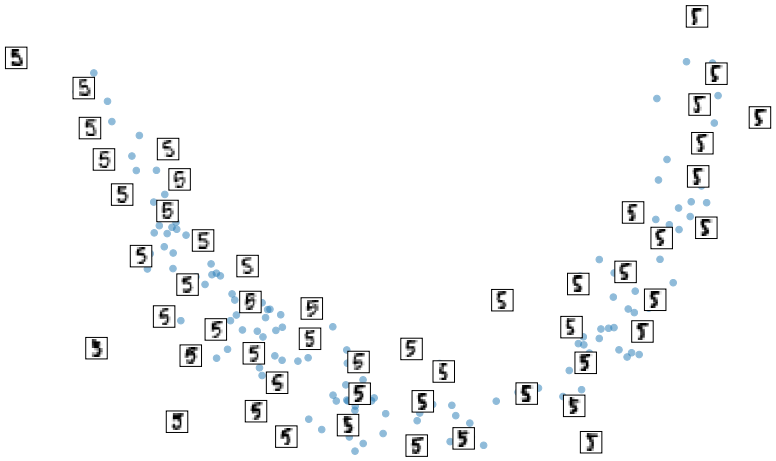


(a) Two-dimensional embedding of the fours computed with standard diffusion maps.

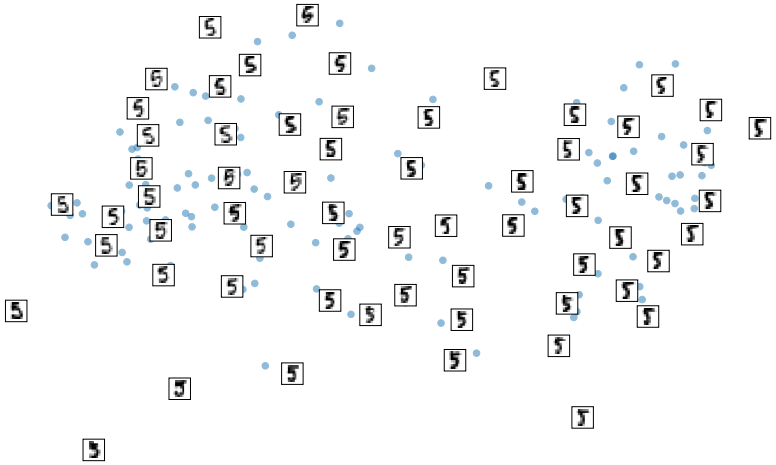


(b) Two-dimensional embedding of the fours computed with `MDM`.

Figure 3.17.: Two-dimensional embeddings of the fours in the digits dataset with standard diffusion maps and `MDM`. Small thumbnails are shown for some of the digits in the corresponding area of the embedding.



(a) Two-dimensional embedding of the fives computed with standard diffusion maps.



(b) Two-dimensional embedding of the fives computed with `MDM`.

Figure 3.18.: Two-dimensional embeddings of the fives in the digits dataset with standard diffusion maps and `MDM`. Small thumbnails are shown for some of the digits in the corresponding area of the embedding.

Finally, after having performed numerical experiments, one should prove a rigorous convergence theorem of `MDM` for $n \to \infty$ (which should be possible in a similar manner as for standard diffusion maps). Possibly, there is a theorem even for $n < \infty$ using some assumptions on the regularity and the available amount of the data.

A. Preliminaries

Some prerequired mathematical methods that were used in this thesis are described below.

Importance Sampling

Importance sampling can be seen as a way of studying one distribution while sampling from another. In our special case, we want to study the Gaussian distribution and assume that we (locally) have samples from a uniform distribution on a d-dimensional ball of radius r.

Suppose we want to know $\mu = \mathbb{E}_p[f(X)]$, where the index of $\mathbb{E}$ denotes the distribution of the random variable (i.e. in this case $X \sim p$, but analytical integration is not possible; we can only evaluate p at certain x in our sample space Ω. Even worse, we might just be able to compute an unnormalized $p^u = cp$ for some unknown constant $c \in \mathbb{R}$. Therefore, a Monte Carlo approach to approximate our quantity of interest $\mu = \mathbb{E}_p[f(X)]$ using a finite set of samples will be used.

To do so, it might be advantageous to not sample $X_i \sim p$ directly and estimate $\mathbb{E}[f(X)] \approx \frac{1}{n}\sum_{i=1}^n p(X_i)f(X_i)$ but to sample Y_i from some other distribution q. To account for this change of density, we reweigh with weights $w = p/q$ or, what we will consider here, unnormalized weights $w^u = p^u/q^u$ in the case where we are also just able to sample from an unnormalized density $q^u = bq$. More precisely, we use the *self-normalized importance sampling estimate*

$$\tilde{\mu} = \frac{\sum_{i=1}^n w^u(Y_i)f(Y_i)}{\sum_{i=1}^n w^u(Y_i)}. \tag{A.1}$$

A very frequent application of this procedure is for the case that p is close to zero in the interesting or important regions with regard to the quantity of interest f, such that one would need too many samples to make a Monte Carlo approach feasible. As an intuitive example, trying to estimate the expected outcome in a lottery well by drawing samples from the lottery numbers uniformly needs a lot of samples, because the jackpot is very unlikely to hit, but important for the expected value. Sampling $Y_i \sim q$, one increases the probabilities of sampling in the interesting regions and accounts for the change of density function with the reweighting. This is why q is called *importance distribution* and p is called *nominal distribution* in this context. In the lottery case, one would try to draw more samples from all classes winning prize money and reweighting accordingly later. In our case, it is even not possible to get samples from p (i.e. the Gaussian distribution), since we are given a fixed set X of samples from the very beginning (which we assume to be locally approximately uniformly distributed).

The following theorem from [Owe13], chapter 9, is a justification for this procedure.

Theorem A.1: *Let p be a probability density function on $\mathbb{R}^d$ and let $f : \Omega \to \mathbb{R}$ be a function such that $\mu = \mathbb{E}_p[f(X)] = \int_\Omega p(x)f(x)dx$ exists. Suppose that $q : \mathbb{R}^d \to \mathbb{R}$ is a probability density function with $q(x) > 0$ for all $x \in \Omega$. Let $Y_1, \ldots, Y_n \overset{i.i.d}{\sim} q$*

be independent and let $\tilde{\mu}$ *be the self-normalized importance sampling estimate* (A.1). *Then*

$$\mathbb{P}\left(\lim_{n\to\infty} \tilde{\mu} = \mu\right) = 1.$$

Proof. Substituting the definition of w^u into (A.1), we get

$$\begin{aligned}\tilde{\mu} &= \frac{\sum_{i=1}^n w^u(Y_i) f(Y_i)}{\sum_{i=1}^n w^u(Y_i)} \\ &= \frac{\sum_{i=1}^n cp(Y_i) f(Y_i)/(bq(Y_i))}{\sum_{i=1}^n cp(Y_i)/(bq(Y_i))} \\ &= \frac{\frac{1}{n}\sum_{i=1}^n p(Y_i) f(Y_i)/q(Y_i)}{\frac{1}{n}\sum_{i=1}^n p(Y_i)/q(Y_i)}. \end{aligned} \tag{A.2}$$

This is why $\tilde{\mu}$ is called *self-normalized*: The normalizing constants b, c cancel. Further note that for $Y \sim q$,

$$\begin{aligned}\mathbb{E}_q\left[\frac{p(Y)f(Y)}{q(Y)}\right] &= \int_{\mathbb{R}^d} \frac{p(x)f(x)}{q(x)} q(x)dx \\ &= \int_{\mathbb{R}^d} p(x)f(x)dx \\ &= \mathbb{E}_p[f(X)] \\ &= \mu.\end{aligned}$$

The denominator of (A.2) approximates this expectation value $\mathbb{E}_q\left[\frac{p(Y)f(Y)}{q(Y)}\right]$. Therefore, we can apply the strong law of large numbers, since the it exists and $Y_1, \ldots, Y_n \overset{\text{i.i.d}}{\sim} q$. This yields the convergence result

$$\mathbb{P}\left(\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^n \frac{p(Y_i)f(Y_i)}{q(Y_i)} = \mu\right) = 1.$$

The numerator of (A.2) converges to 1, since the same argument also applies when choosing $f \equiv 1$ so that $\mathbb{E}[f(X)] = 1$. Since $X_n \xrightarrow{a.s.} X$, $Y_n \xrightarrow{a.s.} Y$ for general random variables implies $X_n Y_n \xrightarrow{a.s.} XY$, we get

$$\mathbb{P}\left(\lim_{n\to\infty} \tilde{\mu} = \mu\right) = 1.$$

□

Note that the assumption $q(x) > 0$ for all $x \in \mathbb{R}^d$ can be weakened to $q(x) > 0$ for just all x where $p(x) > 0$. Proving this just requires to split the expectation integral in corresponding regions where the densities are equal to or greater than zero. This is omitted for simplicity here.

Nyström Extension

As an a priori remark, note that the following derivation is for finite data and matrices in order to stick to the situation of `MDM`. It can, however, be generalized to continuous eigenfunctions and kernels.

In this paragraph, we denote entries of matrices A defined on a particular set X or $\bar{X}$ of data points by A_{xy} for the row of x and the column of y. That is, we do not have an index for the data points in order to avoid double subscripts. The entry of a vector ψ for the data point x is denoted by $\psi(x)$, like in the main text.

The ordinary *Nyström extension* is an extension of eigenvectors (viewed as functionals on the data points) $\psi : X \to \mathbb{R}$ on the dataset X to eigenvectors $\bar{\psi} : \bar{X} \to \mathbb{R}$ on a bigger set of points $\bar{X}$. It is defined for symmetric matrices, in our case symmetric kernel matrices K. That is, we have a vector ψ with

$$\lambda\psi(x) = \sum_{y \in X} K_{xy}\psi(y) \quad \forall x \in X. \tag{A.3}$$

We observe that if we can also define $K_{\bar{x}y}$ for $\bar{x} \in \bar{X}$, $y \in X$ in some natural way, then in the manner of [CL06b] (Definition 2), after rearranging equation (A.3), it is possible to define the Nyström extension $\bar{\psi} : \bar{X} \to \mathbb{R}$ of $\psi : X \to \mathbb{R}$ by

$$\bar{\psi}(\bar{x}) := \frac{1}{\lambda} \sum_{y \in X} K_{\bar{x}y}\psi(y) \quad \forall \bar{x} \in \bar{X}.$$

In our case, K_{xy} is the density normalized kernel

$$k^{(\alpha)}(x, y) = \frac{k(x, y)}{d_x^\alpha \, d_y^\alpha}$$

with $d_x = \sum_{z \in X} k(x, z)$, $d_y = \sum_{z \in X} k(y, z)$. That we can indeed naturally extend to points $\bar{x} \in \bar{X}$ by

$$k^{(\alpha)}(\bar{x}, y) = \frac{k(\bar{x}, y)}{d_{\bar{x}}^\alpha \, d_y^\alpha}$$

with $d_{\bar{x}} = \sum_{z \in X} k(\bar{x}, z)$, since $k(\bar{x}, y)$ is also computable for $\bar{x} \notin X$ with our Gaussian kernel.

One more aspect has to be elucidated though: ψ is the eigenfunction of the transition matrix $P = P^{(\alpha)}$, which is *not* symmetric. Still, we would like to compute the Nyström extension in the same way like above, more precisely as

$$\bar{\psi}(\bar{x}) := \frac{1}{\lambda} \sum_{y \in X} P_{\bar{x}y}\psi(y) \tag{A.4}$$

with

$$P_{\bar{x}y} = P^{(\alpha)}_{\bar{x}y} = \frac{k^{(\alpha)}(\bar{x}, y)}{d^{(\alpha)}_{\bar{x}}}$$

for $d^{(\alpha)}_{\bar{x}} := \sum_{z\in X} k^{(\alpha)}(\bar{x}, z)$.

Luckily, this is in fact not a problem, because $P = P^{(\alpha)}$ is similar to another symmetric kernel matrix $G_{xy} = g(x, y)$, so that P shares the eigenvectors with G (except for a linear transformation related to the similarity transformation). This implies that the extension in the manner of (A.4) can be translated into the ordinary Nyström extension (for symmetric kernel matrices) of G. More precisely, we note that

$$P = D^{-1}K^{(\alpha)} \quad \text{with } D = \operatorname{diag}(D_{xx}) \text{ for } D_{xx} = d^{(\alpha)}_x = \sum_{z\in X} k^{(\alpha)}(x, z)$$

such that we can set

$$\begin{aligned} G &:= D^{-\frac{1}{2}}K^{(\alpha)}D^{-\frac{1}{2}} \quad \text{(which is a symmetric matrix)} \\ &= \left(D^{\frac{1}{2}}P\right)D^{-\frac{1}{2}}, \end{aligned}$$

i.e. P is similar to G. Therefore, if ξ is an eigenvector of G, we have

$$\begin{aligned} G\xi = \lambda\xi &\Leftrightarrow D^{\frac{1}{2}}PD^{-\frac{1}{2}}\xi = \lambda\xi \\ &\Leftrightarrow P\left(D^{-\frac{1}{2}}\xi\right) = \lambda\left(D^{-\frac{1}{2}}\xi\right) \\ &\Leftrightarrow D^{-\frac{1}{2}}\xi =: \psi \text{ is eigenvector of } P \end{aligned}$$

and vice versa. Moreover, as claimed, the Nyström extension also translates this way: Let $\bar{x} \in \bar{X}$, then

$$\begin{aligned} \xi(\bar{x}) &= \frac{1}{\lambda}\sum_{x\in X} G_{\bar{x}x}\xi(x) \quad \text{(ordinary Nyström extension for symmetric matrices)} \\ &= \frac{1}{\lambda}\sum_{x,y,z\in X} D^{\frac{1}{2}}_{\bar{x}z}P_{zy}\left(D^{-\frac{1}{2}}_{yx}\xi(x)\right) \\ &= \frac{1}{\lambda}\sum_{x,y\in X} D^{\frac{1}{2}}_{\bar{x}z}P_{zy}\psi(y) \end{aligned}$$

so by multiplying the equation $\xi = \frac{1}{\lambda}D^{\frac{1}{2}}P\psi$ with $D^{-\frac{1}{2}}$ from the left and again recalling that $\psi = D^{-\frac{1}{2}}\xi$, we get that

$$\bar{\psi}(\bar{x}) = \frac{1}{\lambda}\sum_{z\in X} P_{\bar{x}y}\psi(y).$$

In other words, the Nyström extension for our row-stochastic (non-symmetric) tran-

sition matrix is the same as an ordinary Nyström extension for symmetric matrices.

Finite Differences

As an approximation of the first derivative $(\nabla_x \psi)_l$ in the l-th component at x, we use the *central difference*

$$\frac{\psi(x + he_l) - \psi(x - he_l)}{2h} = \frac{(\psi)_i(x + h) - (\psi)_i(x - h)}{2h} = (\nabla_x \psi)_i + \mathcal{O}(h^2)$$

where e_l denotes the l-th unit vector. Using a Taylor expansion, this can be easily shown to approximate the first derivative up to second order terms.

Kernel Density Estimation

Suppose we are given a finite number of points $\{x_1, \ldots, x_n\} \subset R^D$ randomly sampled according to a probability density function p. Kernel density estimation (KDE) is a way to approximate the shape of p. Its *kernel density estimator* is defined as

$$\hat{p}_\sigma(x) := \frac{1}{n} \sum_{i=1}^{n} k_\sigma(x, x_i) = \frac{1}{n\sigma} \sum_{i=1}^{n} g\left(\frac{\|x - x_i\|}{\sigma}\right)$$

for some kernel $k_\sigma(y, z) = g(\|y - z\|/\sigma)$ with *bandwidth* σ. The latter acts as a smoothing parameter: Large σ will cause very smoothed out estimations. There is a range of different choices for the kernel function k, a common choice is the Gaussian kernel $k_\sigma(x, y) :\propto \exp\left(\|x - y\|_2^2/\sigma\right)$.

In section 2.3, a weighted variant of this is used, denoted by $\hat{p}_\sigma^w(x)$ here. We just scale every entry in the sum by its corresponding weight w_i and normalize by the sum of the weights:

$$\hat{p}_\sigma^w(x) := \frac{1}{\sum_j^n w_i} \sum_{i=1}^{n} w_i k_\sigma(x, x_i) = \frac{1}{\sigma \sum_j^n w_i} \sum_{i=1}^{n} w_i g\left(\frac{\|x - x_i\|}{\sigma}\right).$$

Nearest-Neighbor-Search

If the computation of all distances between all data points is not necessary or useful, one often reduces the number of distance computations by just calculating the distances $\|x - y_i\|$ of the *neighbors* y_i of all points x in the finite dataset X. The number of neighbors may e.g. be chosen to be a fixed integer k for all $x \in X$. Another way is to choose a neighborhood radius ε and require that the distances to all neighbors $y_i \in B_\varepsilon(x)$ in the ball around x with radius ε are calculated.

Computing the nearest neighbors for all data points in the naïve brute force way, however, requires to compute all $n(n + 1)/2 = \mathcal{O}(n^2)$ distances between the n data points in X. Since the goal is to avoid this, a variety of tree-based data indexing structures have been invented, reducing the computational cost to $\mathcal{O}(n \log n)$ or better. Common choices are KD-trees [Ben75] and ball trees [Omo89], where the latter shall

address the inefficiency of KD-trees in high dimensions. In essence, these methods exploit some triangle inequality in the following sense: if x_3 is very far from x_1 but x_2 is very close to x_1, then x_3 must also be far from x_2.

Dijkstra's Algorithm

Dijkstra's algorithm calculates the shortest path between two nodes $s, t \in V$ in a weighted undirected graph $G = (V, E)$ with non-negative edge weights. It starts at the source node $s \in V$ and iteratively extends a shortest path tree by the node which has minimal distance to s in the set of all currently unvisited nodes. It stops, when all neighbors of the target node $t \in S$ have been visited.

Hence, the algorithm is a *greedy* algorithm. The correctness proof of the Dijkstra's algorithm also uses fact: A possible shorter path from s to t would have been considered before termination of the algorithm, since it is always continued with the node with minimal distance to s.

This is also what comes in handy, if we want to only compute paths emanating from s, which have length shorter than some threshold ε, like in `MDM`. Dijkstra finds all nodes with distance shorter than ε, since it stops when there is no node left, which has distance shorter than ε to s.

For more details, see e.g. [CLRS01].

Bibliography

[AW10] Hervé Abdi and Lynne J Williams.
Principal component analysis.
Wiley interdisciplinary reviews: computational statistics, 2(4):433–459, 2010.

[Ban17] Ralf Banisch.
Mathematical aspects in machine learning.
`https://github.com/ralfbanisch/ml_sandbox/blob/master/script.pdf`, 2017.

[Ben75] Jon Louis Bentley.
Multidimensional binary search trees used for associative searching.
Communications of the ACM, 18(9):509–517, 1975.

[BH16] Tyrus Berry and John Harlim.
Variable bandwidth diffusion kernels.
Applied and Computational Harmonic Analysis, 40(1):68–96, 2016.

[BHK16] Avrim Blum, John Hopcroft, and Ravindran Kannan.
Foundations of data science.
Vorabversion eines Lehrbuchs, 2016.

[CBR+14] Claudio Ceruti, Simone Bassis, Alessandro Rozza, Gabriele Lombardi, Elena Casiraghi, and Paola Campadelli.
Danco: An intrinsic dimensionality estimator exploiting angle and norm concentration.
Pattern recognition, 47(8):2569–2581, 2014.

[CCCR15] P Campadelli, E Casiraghi, C Ceruti, and A Rozza.
Intrinsic dimension estimation: Relevant techniques and a benchmark framework.
Mathematical Problems in Engineering, 2015, 2015.

[CGD+14] Eliodoro Chiavazzo, Charles W Gear, Carmeline J Dsilva, Neta Rabin, and Ioannis G Kevrekidis.
Reduced models in chemical kinetics via nonlinear data-mining.
Processes, 2(1):112–140, 2014.

[CH06] Jose A Costa and Alfred O Hero.
Determining intrinsic dimension and entropy of high-dimensional shape spaces.
In *Statistics and Analysis of Shapes*, pages 231–252. Springer, 2006.

[CL06a] Ronald R Coifman and Stéphane Lafon.
Diffusion maps.
Applied and computational harmonic analysis, 21(1):5–30, 2006.

[CL06b] Ronald R Coifman and Stéphane Lafon.
Geometric harmonics: a novel tool for multiscale out-of-sample extension of empirical functions.
Applied and Computational Harmonic Analysis, 21(1):31–52, 2006.

[Clo16] IBM Marketing Cloud.
10 Key Marketing Trends for 2017.
`https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN`, 2016.
Accessed: 2018-04-20.

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein.
Introduction to Algorithms-Secund Edition.
McGraw-Hill, 2001.

[CS16] Francesco Camastra and Antonino Staiano.
Intrinsic dimension estimation: Advances and open problems.
Information Sciences, 328:26–41, 2016.

[CSSS08] Ronald R Coifman, Yoel Shkolnisky, Fred J Sigworth, and Amit Singer.
Graph laplacian tomography from unknown random projections.
IEEE Transactions on Image Processing, 17(10):1891–1899, 2008.

[DKT17] Dua Dheeru and Efi Karra Taniskidou.
UCI machine learning repository.
`http://archive.ics.uci.edu/ml`, 2017.
Accessed: 2018-04-20.

[FdRL17] Elena Facco, Maria d'Errico, Alex Rodriguez, and Alessandro Laio.
Estimating the intrinsic dimension of datasets by a minimal neighborhood information.
Scientific reports, 7(1):12140, 2017.

[FR83] Jerome H Friedman and Lawrence C Rafsky.
Graph-theoretic measures of multivariate association and prediction.
The Annals of Statistics, pages 377–391, 1983.

[FT87] Michael L Fredman and Robert Endre Tarjan.
Fibonacci heaps and their uses in improved network optimization algorithms.
Journal of the ACM (JACM), 34(3):596–615, 1987.

[GC16] Daniele Granata and Vincenzo Carnevale.
Accurate estimation of the intrinsic dimension using graph distances: Unraveling the geometric complexity of datasets.
Scientific reports, 6:31377, 2016.

[GN13] Denis S Grebenkov and B-T Nguyen.
Geometrical structure of laplacian eigenfunctions.
SIAM Review, 55(4):601–667, 2013.

[GP83] Peter Grassberger and Itamar Procaccia.
Measuring the strangeness of strange attractors.
Physica D: Nonlinear Phenomena, 9(1-2):189–208, 1983.

[HA] Matthias Hein and Jean-Yves Audibert.
Intrinsic dimensionality estimation code.
`http://www.ml.uni-saarland.de/code/IntDim/IntDim.htm`.
Accessed: 2018-04-03.

[HA05] Matthias Hein and Jean-Yves Audibert.
Intrinsic dimensionality estimation of submanifolds in $\mathbb{R}^d$.
In *Proceedings of the 22nd international conference on Machine learning*, pages 289–296. ACM, 2005.

[Hau18] Felix Hausdorff.
Dimension und äußeres maß.
Mathematische Annalen, 79(1-2):157–179, 1918.

[HAVL05] Matthias Hein, Jean-Yves Audibert, and Ulrike Von Luxburg.
From graphs to manifolds–weak and strong pointwise consistency of graph laplacians.
In *International Conference on Computational Learning Theory*, pages 470–485. Springer, 2005.

[Hof14] Till Hoffmann.
Weighted kernel density estimation.
`https://gist.github.com/tillahoffmann/f844bce2ec264c1c8cb5`, 2014.
Accessed: 2018-05-02.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult.
Exploring network structure, dynamics, and function using networkx.
Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[Hun07] J. D. Hunter.
Matplotlib: A 2d graphics environment.
Computing In Science & Engineering, 9(3):90–95, 2007.

[Jam99] Ioan Mackenzie James.
History of topology.
Elsevier, 1999.

[JOP+01] Eric Jones, Travis Oliphant, Pearu Peterson, et al.
SciPy: Open source scientific tools for Python.
`http://www.scipy.org/`, 2001.
Accessed: 2018-04-20.

[Kev17] Ioannis Kevrekidis.
No equations, no parameters no variables - data and the reconstruction of normal forms for parametrically dependent dynamical systems.
In *SIAM conference on Applications of Dynamical Systems*, May 21-25 2017.

[Kru64] Joseph B Kruskal.
Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis.
Psychometrika, 29(1):1–27, 1964.

[Laf04] Stéphane S Lafon.
Diffusion maps and geometric harmonics.
PhD thesis, Yale University PhD dissertation, 2004.

[LB05] Elizaveta Levina and Peter J Bickel.
Maximum likelihood estimation of intrinsic dimension.
In *Advances in neural information processing systems*, pages 777–784, 2005.

[LCC08] Gabriele Lombardi, Elena Casiraghi, and Paola Campadelli.
Curvature estimation and curve inference with tensor voting: a new approach.
In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 613–624. Springer, 2008.

[Lev16] Barak Sober David Levin.
Moving least-squares projective approximation of manifolds (mmls).
arXiv preprint arXiv:1606.07104, 2016.

[LLJM09] Anna V Little, Jason Lee, Yoon-Mo Jung, and Mauro Maggioni.
Estimation of intrinsic dimensionality of samples from noisy low-dimensional manifolds in high dimensions with multiscale svd.
In *Statistical Signal Processing, 2009. SSP'09. IEEE/SP 15th Workshop on*, pages 85–88. IEEE, 2009.

[Lom] Gabriele Lombardi.
Intrinsic dimensionality estimation techniques.
`https://de.mathworks.com/matlabcentral/fileexchange/40112-intrinsic-dimensionality-estimation-techniques`.
Accessed: 2018-04-03.

[LP05] Carsten Lange and Konrad Polthier.
Anisotropic smoothing of point sets.
Computer Aided Geometric Design, 22(7):680–692, 2005.

[Mag] Mauro Maggioni.
Multiscale svd code.
`http://www.math.jhu.edu/~mauro/#tab_MSVD`.
Accessed: 2018-04-09.

[MM04] Philippos Mordohai and Gérard Medioni.
Stereo using monocular cues within the tensor voting framework.
In *European Conference on Computer Vision*, pages 588–601. Springer, 2004.

[NLCK06] Boaz Nadler, Stéphane Lafon, Ronald R Coifman, and Ioannis G Kevrekidis.
Diffusion maps, spectral clustering and reaction coordinates of dynamical systems.
Applied and Computational Harmonic Analysis, 21(1):113–127, 2006.

[Omo89] Stephen M Omohundro.
Five balltree construction algorithms.
International Computer Science Institute Berkeley, 1989.

[Owe13] Art B. Owen.
Monte Carlo theory, methods and examples.
2013.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay.
Scikit-learn: Machine learning in Python.
Journal of Machine Learning Research, 12:2825–2830, 2011.

[PY01] Mathew D Penrose and Joseph E Yukich.
Central limit theorems for some graphs in computational geometry.
Annals of Applied probability, pages 1005–1041, 2001.

[RLC+12] Alessandro Rozza, Gabriele Lombardi, Claudio Ceruti, Elena Casiraghi, and Paola Campadelli.
Novel high intrinsic dimensionality estimators.
Machine learning, 89(1-2):37–65, 2012.

[RS00] Sam T Roweis and Lawrence K Saul.
Nonlinear dimensionality reduction by locally linear embedding.
science, 290(5500):2323–2326, 2000.

[Tak85] Floris Takens.
On the numerical determination of the dimension of an attractor.
In *Dynamical systems and bifurcations*, pages 99–106. Springer, 1985.

[Tau95] Gabriel Taubin.
Estimating the tensor of curvature of a surface from a polyhedral approximation.
In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 902–907. IEEE, 1995.

[TDSL00] Joshua B Tenenbaum, Vin De Silva, and John C Langford.
A global geometric framework for nonlinear dimensionality reduction.
science, 290(5500):2319–2323, 2000.

[vL18] Johannes von Lindheim.
Implementations and numerical experiments of the gaussian annulus id heuristics and minimal diffusion maps.
`http://www.zib.de/ext-data/manifold-learning/`, 2018.
Accessed: 2018-05-09.

[WRI] Wolfram Research, Inc.
Mathematica, Version 10.4.
Champaign, IL, 2014.

[You82] Lai-Sang Young.
Dimension, entropy and lyapunov exponents.
Ergodic theory and dynamical systems, 2(1):109–124, 1982.