Manifold aligned density estimation

Xiaoxia Wang
School of Computer Science
University of Birmingham

A thesis submitted for the degree of
Doctor of Philosophy

June 15, 2010

— to my family —

My husband Ping Sun

My parents Renquan Wang and Yufang Liu

Acknowledgements

First and foremost I am grateful to my supervisor Dr. Peter Tino. This thesis would never have been completed without his continuous help on the clarification of ideas and paper writings. I have benefited greatly from his encouragement, wide knowledge, and clarity of thought in conducting this research.

I would like to thank Dr. Somak Raychaudhury, Prof. Arif Babul, Dr. Mark A. Fardal and Prof. Puragra GuhaThakurta from astronomy community for their collaboration, inspiring talking and provision of the astronomical simulation datasets used in the thesis. I would also like to thank Dr. Prakash N. Patil for his support on theoretical analysis in Chapter 2.

Finally, I would like to thank my thesis group members, Dr Richard Durdon, and Dr Ata Kaban for taking time to read my RSMG reports and giving many valuable comments.

Abstract

With the advent of the information technology, the amount of data we are facing today is growing in both the scale and the dimensionality dramatically. It thus raises new challenges for some traditional machine learning tasks. This thesis is mainly concerned with manifold aligned density estimation problems. In particular, the work presented in this thesis includes efficiently learning the density distribution on very large-scale datasets and estimating the manifold aligned density through explicit manifold modeling.

First, we propose an efficient and sparse density estimator: Fast Parzen Windows (FPW) to represent the density of large-scale dataset by a mixture of locally fitted Gaussians components. The Gaussian components in the model are estimated in a "sloppy" way, which can avoid very time-consuming "global" optimizations, keep the simplicity of the density estimator and also assure the estimation accuracy. Preliminary theoretical work shows that the performance of the local fitted Gaussian components is related to the curvature of the true density and the characteristic of Gaussian model itself. A successful application of our FPW on principled calibrating the galaxy simulations is also demonstrated in the thesis.

Then, we investigate the problem of manifold (i.e., low dimensional structure) aligned density estimation through explicit manifold modeling, which aims to obtain the embedded manifold and the density distribution simultaneously. A new manifold learning algorithm is proposed to capture the non-linear low dimensional structure and provides an improved initialization to Generative Topographic Mapping (GTM) model. The GTM models are then employed in our proposed hierarchical mixture model to estimate the density of data aligned along multiple manifolds. Extensive experiments verified the effectiveness of the presented work.

Contents

List of Figures

List of Tables

Glossary

$\mathbf{x}_1, ..., \mathbf{x}_N$	N observations in the input space $\mathcal{R}^D$
$K_h(\cdot)$	Kernel function with smoothing parameter h
$\tilde{f}(\mathbf{x})$	Parzen window density esitmator
$\hat{f}(\mathbf{x})$	Estimated density function
$p(\mathbf{x})$	Density distribution of the observed data $\mathbf{x}$
$\hat{p}(\mathbf{x})$	Fast Parzen window density esitmator
$\mathcal{L}(\cdot)$	Log-likelihood
$\mathcal{K}(\mathbf{x}_i, \mathcal{D})$	K nearest neighbors around $\mathbf{x}_i$ in dataset $\mathcal{D}$
$\mathcal{N}(\mathbf{x}; \mathbf{m}, C)$	Gaussian kernel with the mean $\mathbf{m}$ and the covariance C
$\delta(\cdot)$	Delta function
$\mathcal{G}, \mathcal{G}^m$	Latent space and data space graphs
$\mathbf{M}, \tilde{\mathbf{M}}$	Projection matrix
$\Phi(\cdot)$	Column vector with K fixed basis function $\dot{\phi_j()}$
$\mathbf{W}$	Weight matrix of RBF network
β^{-1}	Variance of the Gaussian distribution
$\mathbf{U}$	Eigenvectors matrix
$\mathcal{U}(a, b)$	Uniform distribution in the interval (a, b)
$\boldsymbol{\tau}$	Unobserved data point in latent space $\mathcal{R}^d$
$M31, M32$	Galaxy names
ALL	Average log likelihood
EM	Expectation-Maximization algorithm
GMM	Gaussian mixture models
GTM	Generative Topographic Mapping
KDE	Kernel density estimation
KL	Kullback-Leibler divergence

$\mathcal{PO}$	Pseudo-observation
PW	Parzen window density esitmator
FPW	Fast Parzen window density esitmator, -H: hard version, -S: soft version
$RSDE$	Reduced Set Density Estimator
SMM	Simplifying Mixture Models

Chapter 1

Introduction

Density estimation and dimensionality reduction are two widely discussed topics in unsupervised learning [2]. Both of them aim to discover hidden structures from the data samples without any labeled information. In density estimation, the hidden structure is represented as some stationary process from which the data samples are assumed to have been generated. The process is usually described by a probabilistic model. After estimating the parameters of the model from the current observations, the novelty of future observations could be predicted or assessed by the model. In dimensionality reduction, the hidden structure refers to a more useful and compact representation of the information in the dataset. In this case, we say that the original data points with high dimensionality actually lie along an unobserved low dimensional structure (i.e, manifold) embedded in the high dimensional space. It appears that density estimation and dimensionality reduction are two different problems but these can be closely related. By taking advantage of underlying manifold structure in the data, the performance of classical density estimator can be significantly improved [3]. On the other hand, by learning a probability density over the data through a latent variable model, manifolds embedded can be represented explicitly [4].

In this thesis, we estimate the density distribution of data points distributed in high dimension space but aligned along the low dimensional structures. The presented work is motivated from a joint project on studying galaxy evolutions with astronomy colleagues. There emerge two challenges from modeling the astronomical data for simulating galaxy evolutions. The first one is that the size of dataset becomes *very large*, which makes many density estimators computationally prohibitive. The second one is that *more than one* low dimensional structure are embedded in the data and

these are non-linear in some circumstances. We address these two challenges by presenting sparse density estimation techniques and a novel multiple manifolds learning framework in this thesis.

In this chapter, we introduce both the classical density estimators and manifold learning algorithms. Then, we outline the thesis structure in Section 3 and highlight main contributions of the author in Section 4.

1.1 Density estimation

For a given dataset, density estimation is to construct an estimation of unobservable, underlying unknown density function based on observed data [5, 6].

Parametric density estimators [7] assume a particular shape to describe the density distribution of the observed data samples. The shape is formulated by a function with parameters (e.g. unknown expectation and variance of Normal distribution), which are estimated by fitting the function to the observed dataset with optimization. For parametric density estimators, an inaccurate assumption on the density distribution will lead to an incorrect density estimation.

Semi-parametric density estimator [7] mixes a set of probability density functions from a parametric family to form a complex distribution of the data. If choosing the component as Gaussian distribution, we get a Gaussian mixture model [8]. Since the label information indicating from which component of the mixture the data points were generated is unknown, it is not straightforward to estimating the parameters of a semi-parametric density estimator (mixture model). The Expectation-Maximization (EM) [9] algorithm solves this problem by introducing an Expectation (E) step to compute the responsibility of the component to each data points. Then it calculates the parameters in the Maximization (M) step by maximizing the expected complete-data (including the observed data and the unobserved label information) log likelihood function using the responsibility obtained in E step.

Both parametric density and semi-parametric density estimators are based on the assumption of the density distribution of the observed data. Prior knowledge for making this kind of assumption may not be available in many cases. It might be because the density distribution is very complex itself or the data is new to the analyst that

no prior knowledge of data properties could be used. Non-parametric density estimations [7, 10] therefore were proposed to show the structure of the distribution from the data itself like simple histogram [11, 12] nonparametric density estimator. Another widely used non-parametric density estimator is Kernel Density Estimator, or KDE [11, 13]. It provides a much smoother representation of the density distribution compared to histogram.

In Kernel Density Estimator [11, 14], each observed data point is associated with a smooth kernel. The overall density is represented by the sum of these kernels. The accuracy of the kernel density estimator depends mainly on the width of the smooth kernel. With a too small value of the width, kernel density estimator gives a density distribution over-fitting to the observed data, but if the width is too large, some fine structure of the density distribution may not be captured. Variable kernel density estimator [11, 15, 16] was proposed to improve the accuracy of the classical kernel density estimator by replacing the single kernel width by N various values (N is the size of the dataset). So a varied of degrees of smoothing are considered at different data points. One example of the variable kernel density estimator is manifold Parzen Windows estimator [3]. It uses Gaussians as the kernels and assumes that the high dimensional data points lie on a low dimensional structure (which is called "manifold"). Instead of using the same covariance for all the Gaussian kernels, manifold Parzen Windows estimates the covariance of the Gaussian on top of each data point according to its local manifold patch. By employing the Gaussian kernel aligned along the manifold, the method of manifold Parzen Windows gives more weights on the data points aligned along the manifolds and demonstrates the improved estimation accuracy on high dimensional data lying on low dimensional manifold.

For estimating the density distribution of a real dataset by kernel density estimator, the main concern is its computational complexity. Since all the observed data points are used to form the density distribution, the basic form of this method can be computationally prohibitive when the dataset is very large. This situation will get worse when dealing with dataset of high dimensionality, because kernel density estimator requires more observed data points to estimate the density distribution accurately. To reduce the computational cost in kernel density estimator, many different strategies have been presented in the past. In the case of Gaussian kernels, Fast Gauss Transform

(FGT) [17, 18, 19] and Improved FGT (IFGT) [20] can be used to speed up the summations of Gaussians, which dominates the overall cost of kernel density estimator. For a wider range of isotropic kernels, the computational efficiency could be significantly improved by arranging the dataset in a tree-type multiresolution data structure such as Dual Tree [21] (kd-tree [22] and Anchors Hierarchy data structure). However the practical performance of these techniques (generally called N-body approach [23]) are relatively sensitive to the choices of critical hyperparameters and the dimensionality of the data [24, 25]. Another class of approaches to break down the computational complexity are reducing the amount of kernels (or components) directly in the density estimation, which will be detailed in the next chapter.

1.2 Dimensionality reduction/manifold learning algorithms

Given a set of N high-dimensional data points $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ with $\mathbf{x}_i \in \mathcal{R}^D$, $i = 1, ..., N$, the problem of dimensionality reduction/manifold learning can be represented by the following function [26]:

$$\mathbf{x}_i = F(\boldsymbol{\tau}_i) + \epsilon_i, \quad i \in \{1, \ldots, N\}, \tag{1.1}$$

where we assume that $\mathbf{x}_i$ is sampled possibly with the noise ϵ_i from the manifold F parametrized by $\boldsymbol{\tau}_i \in \mathcal{R}^d$ ($d < D$). By *dimensionality reduction* we refer to the estimation of the low dimensional coordinates $\boldsymbol{\tau}_i$'s from the observations $\mathbf{x}_i$'s with a underlying relationship (1.1). By *manifold learning*, we mean the reconstruction of $F(\cdot)$ from the data. In general, we call these algorithms manifold learning algorithms in this thesis.

Many different approaches were proposed to find low-dimensional structures among high-dimensional data points and we can categorize them into different groups according to different classification criteria. In a simple case, we make a distinction between linear and nonlinear methods. Linear manifold learning algorithms assume that the function $F(\cdot)$ is a linear mapping. Examples of these algorithms are Principal Components Analysis (PCA), Projection Pursuit (PP) [27, 28], factor analysis (FA) [29] and classical scaling [30]. On the other hand, most of the manifold learning algorithms are designed to recover a non-linear relation. This kind of algorithms include Principle curves(surface) [31], Self-organizing Mapping (SOM) [32],Generative Topographic Mapping (GTM) [4], Isomap [33], LLE [34] and many more. We can also di-

vide manifold learning algorithms into convex and non-convex techniques based on the convexity of the objective function in the optimization [35].

In this work, we are particularly interested in latent variable based manifold learning algorithm (e.g., GTM) because it builds an explicit probabilistic model of the embedding function $F(\cdot)$. In the probabilistic framework, the latent variable models could be easily mixed to represent more complex dataset. Accordingly, we can also represent more than one manifold in the data by mixing the generative models. In line with that, the following brief review classifies various manifold learning algorithms into two types: techniques without probabilistic setting and latent variable model based ones. We only introduce most common approaches here and refer readers to other related work.

1.2.1 Manifold learning algorithms without probabilistic setting

Principal Components Analysis (PCA) [36] is by far the most popular linear technique and it seeks a low-dimensional representation of the data by maximizing the amount of variance in the data. In mathematics, PCA can be formulated as finding the first d principle eigenvalues and associated eigenvectors of the covariance matrix. To cope with complex nonlinear data, some nonlinear generalizations of PCA have been developed. For example, principal curves (surfaces) [31, 37] attempts to provide a smooth curved approximation to data points in a least-squares sense. By employing popular kernel trick in supervised learning [38], a kernel version of PCA (KPCA) [39, 40] have been derived to capture nonlinear structures embedded in high-dimensional data.

Self-organizing mapping, or SOM [32, 41] is another best-known learning algorithm for dimensionality reduction. It creates a set of prototype vectors representing the data set and carries out a topology preserving projection of the prototypes from the high-dimensional input space onto a low-dimensional grid. The ordered grid can be used for a convenient visualization tool for high-dimensional data [42]. For original SOM, the inter-neuron distances are not visible or measurable on the grid map such that the structures of high-dimensional data points may not be kept. To remedy the drawbacks a visualization-induced SOM (ViSOM) [43] was proposed. A comprehensive overview on the developments of SOM for visualization can be found in [44].

Two other nonprobabilistic methods (e.g., Locally linear embedding, or LLE [34] and Isomap [33]) for dimensionality reduction are worthy of mention as they largely

renewed research interests in developing efficient nonlinear manifold learning algorithms in the last few years. Basically LLE [34, 45] is motivated from the assumption that each data point and its neighbors in the data should lie on or close to a locally linear patch of the manifold. It begins by computing the set of coefficients that best reconstructs each data point from its neighbors. Then LLE uses an eigenvector-based optimization technique to find the low-dimensional embedding of points while preserving these neighborhood coefficients. By following the same line of research of preserving local properties of the data, a number of new manifold learning algorithms were presented including Laplacian Eigenmaps [46], Hessian LLE [47], Locality preserving projections [48] and Local Tangent Space Alignment[26].

The Isomap algorithm [33] uses classical multidimensional scaling (MDS) [30] to recover the low dimensional representation, but seeks to preserve the intrinsic geometry of the data, which is captured by geodesic manifold distances between all pairs of data points. Compared to LLE, Isomap attempts to preserve geometry globally at all scales and the properties could be better understood theoretically [49]. Following the study of Isomap, some work has been recently developed to overcome the drawbacks of original algorithm [49, 50, 51].

1.2.2 Generative model based manifold learning algorithms

The generative model used in manifold learning is a latent variable model [52]. The representation of the distribution $p(\mathbf{x})$ of the observed data in terms of latent variable is computed as:

$$p(\mathbf{x}) = \int p(\mathbf{x}, \boldsymbol{\tau}) d\boldsymbol{\tau} = \int p(\mathbf{x}|\boldsymbol{\tau}) p(\boldsymbol{\tau}) d\boldsymbol{\tau} \tag{1.2}$$

where $p(\mathbf{x}, \boldsymbol{\tau})$ is the joint distribution and it can be decomposed into the product of the marginal distribution $p(\boldsymbol{\tau})$ of the latent variable and the conditional distribution $p(\mathbf{x}|\boldsymbol{\tau})$. The mapping from latent variable to data variable is represented in the conditional distribution $p(\mathbf{x}|\boldsymbol{\tau})$.

One example of the linear latent variable model based manifold learning algorithm is Probabilistic Principal Component Analysis (PPCA)[53]. It assumes that the distribution of the latent variable is Gaussian and the mapping between the latent variable and observed variable is linear. The linear mapping is obtained from the parameter of

the data distribution $p(\mathbf{x})$. It can be also viewed as a probabilistic version of classical PCA.

To model the more complex data, which aligns along a non-linear low dimensional structure, mixtures of local linear generative models such as mixture of PPCA or mixture of FA [54] were proposed. Although these models provide local linear mapping between local latent spaces and data spaces, the local latent spaces are not comparable with each other. i.e. the coordinate systems of neighboring components in mixture of PPCA might be differently oriented. To remedy this, the work of [55, 56] proposed to learn a "global" low dimensional coordinate system for the data. Similar works were presented in [57, 58] and [59], non-linear manifold was learnt by coordinating or aligning the mixture of local linear models. Different from aligning local linear models, Generative Topographic Mapping (GTM) [4] algorithm defines in terms of a mapping from the pre-defined latent space into the data space and uses EM algorithm to optimize the mapping between the latent space and data space. Several extensions to the original GTM are reported in [60].

Among these generative model based manifold learning methods, there are a couple of disadvantages: 1) The parameters are learnt from Expectation Maximization (or similar) optimization process, which is sensitive to the initialization and slow. 2) Most of the algorithms are proposed with the assumption that the datasets are generated from a single low dimensional structure, which may not be true in practice.

1.3 Thesis organization

The structure of the thesis is organized as follows:

In Chapter 1, we introduced some preliminaries for subsequent chapters: density estimation, manifold learning and probability density estimation along the underlying manifold. We also outlined the structure of the thesis and summarized the main contributions of the author.

Chapter 2 proposes an efficient and sparse density estimator accompanied by some theoretical analysis. First, we review the literature on the techniques of reducing the computational cost for kernel density estimators. Then we describe the proposed Fast Parzen Windows (FPW) and analyze the algorithm theoretically. The effectiveness

and efficiency of the proposed algorithm are verified on both synthetic and galaxy simulation datasets.

Chapter 3 demonstrates an application of our proposed FPW density estimation on large-scale astronomy data. After introducing the astronomical problem of calibrating galaxy disruption simulations, we present a methodology to address this problem based on the FPW density estimator. Experimental results show that FPW provides an efficient and effective way to estimate the density for large-scale astronomy datasets and the likelihood based principled calibration is much more reliable than the classical chi-square test based calibration.

Chapter 4 starts with a review of one generative model based manifold learning algorithm GTM and an illustration of the limitation of classical initialization for the parameters optimization process. Then a novel algorithm is proposed to learn the low dimensional structure, which is used to initialize the GTM model. After that, we verify our novel manifold learning algorithm on both synthetic datasets generated from non-linear manifolds and the galaxy evolution simulation dataset.

In Chapter 5, we extend the generative model for data generated from one manifold to a mixture model for describing the dataset generated from more than one manifold of varying dimensionality. To construct the model, we propose a multiple manifolds learning framework. The chapter finished with the demonstration of the proposed mixture model on both complex synthetic data and astronomical simulations data.

Chapter 6 summarizes this work and describes several future studies for future research.

1.4 Thesis contributions and publications

The significant contributions of the author include the following:

1. An efficient and sparse density estimator: Fast Parzen Windows (FPW) is proposed for large scale density estimation problems. It keeps the simplicity of the non-parametric model and avoids computationally expensive model construction procedures, but provides comparable performances to other density estimators on high dimensional data with low dimensional structures. Loosely

speaking, the presented FPW can be considered as a "sloppy Gaussian mixture model" (chapter2).

2. By analyzing the relation between main contributed component in FPW and the local parametric density estimator, we get some theoretical results on the performance of our proposed FPW (chapter 2).

3. A successful application of the proposed Fast Parzen Window in a density based comparison scheme for a real astronomical problem. Its efficiency of probability density estimator on large-scale dataset makes the proposed comparison scheme possible (chapter 3).

4. A new GTM initialization scheme is proposed. It captures the non-linear low dimensional structure in data space and leads to a significant improved result over classical GTM initialization (chapter 4).

5. A novel mixture model based framework is presented to learn multiple manifolds of possibly different intrinsic dimensionalities (chapter 5).

The work resulting from these investigations has been published in a couple of papers:

Publication list

- X. Wang, P. Tiňo and M. A. Fardal, *Multiple manifolds learning framework based on hierarchical mixture density model,* In proceeding of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), 2008. (Oral presentation)

- X.Wang, P. Tiňo, M. A. Fardal, S. Raychaudhury and A. Babul, *Fast Parzen Window Density Estimator*, International joint conference on Neural Network (IJCNN) 2009. (Oral presentation)

Chapter 2

Efficient and Sparse density estimator (Fast Parzen Windows)

In this chapter, we address the problem of learning kernel density estimation on large-scale dataset and present an efficient and sparse density estimator: Fast Parzen Windows (FPW) algorithm. We start with reviewing previous work on reducing the computational cost for classical kernel density estimator in Section 1. Then, we describe our proposed Fast Parzen Windows. In Section 3, we investigate our efficient and sparse density estimator (FPW) theoretically. Experimental results on two synthetic data sets, as well as on data sets derived from galaxy disruption simulations are reported in Section 4.

2.1 Efficient kernel density estimators

Kernel density estimator (KDE) [11], also known as Parzen Windows [14], is the most popular non-parametric density estimator. It estimates the density distribution as follows:

$$\tilde{f}(\mathbf{x}) = \frac{1}{N} \sum_{j=1}^{N} K_h(\mathbf{x}_j - \mathbf{x}) \tag{2.1}$$

where N is the number of the data samples in dataset $\mathcal{D} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, $K_h(\cdot)$ is the kernel function associated with each data point, h is the smoothing parameter. Although the distribution of the data samples is revealed from dataset itself, classical kernel density estimator is very computationally expensive when the sample size becomes very large. The computational efficiency can be improved by reducing the amount of kernels in the density estimators. Strategies proposed in the literature include data

reduction methods [61, 62], the binned kernel density estimators [63, 64] and model simplifying algorithms [65, 66, 67, 68].

2.1.1 Data Reduction Methods

The basic idea of data reduction methods is reducing a given large data set to a small representative subset on which data analysis can be carried out. The algorithms are usually proposed independent of data analysis tasks and could be used as a pre-processing step to many machine learning problems such as classification, clustering and density estimation. The simplest methods of data reduction are to draw the desired number of samples in a random way[69]. These can be easily implemented and just lead to negligible computational cost. However, a stable performance is not guaranteed because of the randomness nature in these methods. A more principled technique of data reduction is classical vector quantization methods using a set of codebook vectors which minimize the quantization error [61]. Recently, [62] introduced a density-based data reduction method that pruning samples in a multiresolution manner rather than with uniform resolution. The computational efficiency of density-based method can be further enhanced by using some entropy-based criteria [70]. The characteristic of data reduction methods is that they can be broadly used for tackling a wider range of data analysis tasks as we mentioned above. If the accuracy of density estimation is a major concern, data reduction methods might not be the first choice compared to those approaches specially designed for density estimation tasks.

2.1.2 Binned kernel density estimators

Another class of approaches to reduce the computational complexity of KDE is the binned kernel density estimator [71]. The idea is to approximate the kernel density estimated in Eq. (2.1) by the density obtained over a mesh of M grid points $\{b_1, b_2, \ldots, b_M\}$, and binned kernel density reads:

$$\tilde{f}(\mathbf{x}|H) = \frac{1}{H} \sum_{j=1}^{M} \frac{c_j}{N} K_H \left(\mathbf{x} - b_j\right) \tag{2.2}$$

where b_j is equally spaced grid node, c_j is the grid counts based on the neighboring observations and H is the smoothing parameter of the kernel centered on the grid nodes (It is usually different from the smoothing parameter h in Eq. (2.1)). Binned

kernel density estimators reduces the number of kernels in the estimator from the number of the data points N to the number the grid points M.

The accuracy of the binned kernel density estimator is theoretically analyzed by Hall in [72], Scott and Sheather in [63]. Hall [64] also analyzed the case when associating each bin center with different binning kernel. Instead of using the equally spaced grid binning, the work of [73] introduced other non-uniform binning schemes and demonstrated better results in terms of estimation accuracy.

However it is computationally prohibitive for binning in a high-dimensional input space and most algorithms along this line are only focused on univariate data.

2.1.3 Sparse kernel density estimators

In the last decade, sparse approximation techniques [74, 75, 76] have became very popular to tackle large scale machine learning problems including density estimation. If we denote the classical kernel density as

$$\tilde{f}(\mathbf{x}) = \sum_{j=1}^{N} \pi_j \varphi_j(\mathbf{x}), \tag{2.3}$$

where $\pi_j = \frac{1}{N}$ and $\varphi_j(\mathbf{x}) = K_h(\mathbf{x} - \mathbf{x}_j)$. The goal of sparse methods is to approximate $\tilde{f}(\mathbf{x})$ by a simplified model of M components ($M < N$),

$$\hat{g}(\mathbf{x}) = \sum_{i=1}^{M} w_i g_i(\mathbf{x}). \tag{2.4}$$

To obtain $\hat{g}(\mathbf{x})$, several algorithms were proposed to estimate the mixing coefficient w_i with the constraint $\sum_{i=1}^{M} w_i = 1$ and select the components $\{g_i(\mathbf{x}), i = 1, \ldots, M\}$.

Girolami and He [65] proposed *Reduced Set Density Estimator* (RSDE) to give a sparse estimator by directly minimizing the integrated squared error (ISE) between the simplified model and the true density. They showed that a direct minimization of ISE for a general kernel density estimator yields a sparse representation in the weighting coefficients.

Rather than RSDE, the method of *Simplified Mixture Model* was proposed in [66]. It is based on clustering components of the complex mixture model. First, it groups the mixture components $\{\varphi_j\}_{j=1}^{N}$ into disjoint clusters $\{S_1, \ldots, S_M\}$, then the model

simplification error ε can be upper bounded as

$$\begin{aligned}\varepsilon =& \int \left(\sum_{i=1}^{M} w_i g_i(\mathbf{x}) - \sum_{j=1}^{N} \pi_j \varphi_j(\mathbf{x}) \right)^2 d\mathbf{x} \\ \leq & M \sum_{i=1}^{M} \int \left(w_i g_i(\mathbf{x}) - \sum_{j \in S_i} \pi_j \varphi_j(\mathbf{x}) \right)^2 d\mathbf{x} \\ =& M \sum_{i=1}^{M} \bar{\varepsilon}_i.\end{aligned} \tag{2.5}$$

We can see that minimizing the upper bound of ε is equivalent to minimizing each component $\bar{\varepsilon}_i$. Consequently, the $w_i g_i(\mathbf{x})$ can be obtained by minimizing the local error $\bar{\varepsilon}_i$,

$$w_i g_i(\mathbf{x}) = \arg \min_{w_i g_i(\mathbf{x})} \int \left(w_i g_i(\mathbf{x}) - \sum_{j \in S_i} \pi_j \varphi_j(\mathbf{x}) \right) d\mathbf{x}.$$

An earlier work along the same line was done by Goldberger and Roweis [67]. They use a different distance measure between $\hat{f}$ and $\hat{g}$ as

$$d(\tilde{f}, \hat{g}) = \sum_{j=1}^{N} \pi_j \min_{i=1,\ldots,M} KL(\varphi_j || g_i)$$

where $KL(\cdot)$ is the Kullback-Leibler divergence [77]. The simplified density model (mixture of Gaussian) $\hat{g}$ can be obtained as

$$\hat{g} = \arg \min_{\hat{g}} d(\tilde{f}, \hat{g}).$$

It is proved that the optimal density $\hat{g}$ is a mixture of Gaussians obtained from grouping the components of $\tilde{f}$ into clusters and collapsing all Gaussians within a cluster into a single Gaussian.

In [68], the relation between the model $\hat{g}(\mathbf{x})$ and $\hat{f}(\mathbf{x})$ is written in the regression form as follows

$$\begin{aligned}\hat{g}(\mathbf{x}) =& \tilde{f}(\mathbf{x}) + \epsilon(\mathbf{x}) \\ =& \sum_{j=1}^{N} \pi_j K_h(\mathbf{x} - \mathbf{x}_j) + \epsilon(\mathbf{x})\end{aligned} \tag{2.6}$$

where $\epsilon(\mathbf{x})$ is the modeling error at $\mathbf{x}$ between the sparse kernel density estimator $\hat{g}(\mathbf{x})$ and the PW density estimator $\tilde{f}(\mathbf{x})$. They construct the sparse kernel density $\hat{g}(\mathbf{x})$ by using a forward constrained regression [78]. The algorithm selects the component

g_i of the sparse kernel density estimator sequentially from the kernels formed from dataset. The learning procedure terminates when the accuracy of the sparse kernel density estimator $\hat{g}(\mathbf{x})$ is sufficiently close to that of the PW density estimator $\tilde{f}(\mathbf{x})$.

For the algorithms discussed above, although the complex model is replaced by an optimized simple sparse one, the model construction process (i.e., evaluating $\tilde{f}(\mathbf{x})$ in (2.3)) must first be accomplished, which is still a problem for large scale dataset. In order to keep the simplicity of the nonparametric model and avoid the complex model construction procedures, we propose a new efficient and sparse density estimator: Fast Parzen Window density estimator to reduce the computational cost of PW. The idea is to cover the entire data space by a set of hyper-discs of fixed radii. With carefully chosen radii, the density of the partitioned disk cells can be described by a full covariance Gaussian kernel and the local data structure could be well preserved. The global model is formed by a mixture of such locally fitted Gaussians with appropriately mixing weights.

2.2 Fast Parzen Windows density estimator

The basic idea of Fast Parzen Window (FPW) is to segment the whole data set into hyper-discs with fixed radii first. Then it fits a full covariance Gaussian kernel to the data points in each hyper-disc and updates the weights of the mixing kernels. Since we employ Gaussians as the local data-adaptive kernels [3], our density estimator reads:

$$\hat{p}(\mathbf{x}; r) = \sum_{j=1}^{M} P(\mathcal{S}_j)\mathcal{N}(\mathbf{x}; \mathbf{m}_j, C_j) \tag{2.7}$$

where r is the radii used for segmenting the dataset, M is the number of the hyper-discs obtained after the segmentation, $P(\mathcal{S}_j)$ is the mixing coefficients of the segment $\mathcal{S}_j$ with constraint $\sum_j P(\mathcal{S}_j) = 1$, $\mathcal{N}(\mathbf{x}; \mathbf{m}_j, C_j)$ is a multivariate Gaussian density with mean vector $\mathbf{m}_j$ and covariance matrix C_j:

$$\mathcal{N}(\mathbf{x}; \mathbf{m}_j, C_j) = \frac{1}{\sqrt{(2\pi)^D ||C_j||}} \exp\{-\frac{1}{2}(\mathbf{x} - \mathbf{m}_j)^T C_j^{-1}(\mathbf{x} - \mathbf{m}_j)\}. \tag{2.8}$$

Here D is the dimension of the data point, $||C_j||$ denotes the determinant of C_j. Note that $\mathcal{N}(\mathbf{x}; \mathbf{m}_j, C_j)$ is designed to fit the local distribution of the segment (hyper-disc) S_j.

2.2.1 Partitioning the Dataset

The first step of FPW density estimation is partitioning the data space with hyper-discs of fixed radii r, so that the data distribution could be approximated by local densities fitted within the hyper-discs. We use the following algorithm to position the hyper-discs:

1. Set $\mathcal{S} = \emptyset$ (set of hyper-disc centers), $\zeta = \emptyset$ (set of already processed data points from $\mathcal{D}$).

2. Randomly select a data point $\mathbf{x} \in \mathcal{D}$ and declare it to be the first selected center $\mathbf{s}_1$, $\mathbf{s}_1 = \mathbf{x}$.

   - Add $\mathbf{s}_1$ to the partition set $\mathcal{S}$: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{s}_1\}$.
   - Initialize the corresponding partition $\mathcal{S}_1$: $\mathcal{S}_1 = \{\mathbf{x}\}$.
   - Record that $\mathbf{x}$ has been processed: $\zeta \leftarrow \zeta \cup \{\mathbf{x}\}$.
   - Set the center counter: $k = 1$.

3. While $\mathcal{D} \setminus \zeta \neq \emptyset$ (there are data points in $\mathcal{D}$ still to be processed), repeat:

   - Select a data point $\mathbf{x} \in \mathcal{D} \setminus \zeta$ and add it to the set ζ ($\zeta \leftarrow \zeta \cup \{\mathbf{x}\}$).
   - Compute the pairwise distances $D(\mathbf{x}, \mathbf{s}_j)$, $\mathbf{s}_j \in \mathcal{S}$, between $\mathbf{x}$ and the already selected centers in $\mathcal{S}$.
   - If for some j, $D(\mathbf{x}, \mathbf{s}_j) \leq r$, where $r > 0$ is a predefined threshold, assign $\mathbf{x}$ to the corresponding partition $\mathcal{S}_j$: $\mathcal{S}_j \leftarrow \mathcal{S}_j \cup \{\mathbf{x}\}$.
   - If $D(\mathbf{x}, \mathbf{s}_j) > r$ for all $\mathbf{s}_j \in \mathcal{S}$, add $\mathbf{x}$ as a new member of the partition set $\mathcal{S}$:
     - Increase the center counter: $k \leftarrow k + 1$. Set $\mathbf{s}_k = \mathbf{x}$.
     - Add $\mathbf{s}_k$ to the partition set $\mathcal{S}$: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{s}_k\}$.
     - Initialize the corresponding partition $\mathcal{S}_k$: $\mathcal{S}_k = \{\mathbf{x}\}$.

We thus obtain a set of partition centers $\mathcal{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_M\}$ ($M \leq N$) representing the partitions $\{\mathcal{S}_1, \ldots, \mathcal{S}_M\}$ of the data set $\mathcal{D}$.

Note that, unlike in vector quantization, the partition elements $\mathcal{S}_i$ cover the same volume but are populated according to the local density of data points. Compared

with vector quantization based downsampling algorithms, relatively more data samples are chosen from sparse regions by our proposed procedure.

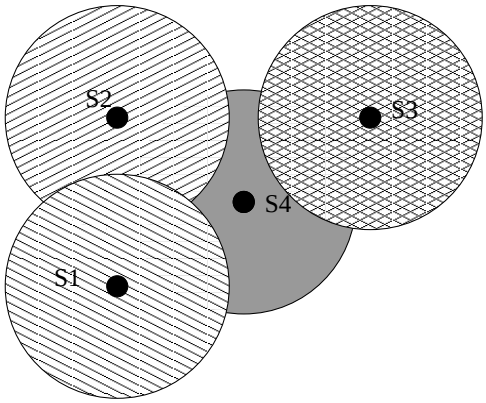
In Figure 2.1, we use s_1, s_2, s_3 and s_4, four partition centers selected sequentially to illustrate a partition example. Figure 2.1 (a) shows the original partitions for selected centers s_1, s_2, s_3 and s_4 by the partition algorithm proposed above. Due to the overlapping among the hyper-discs, very few neighboring points were left for the center s_4 and they may not be able to represent the local distribution of s_4 accurately. To cope with the overlapping, we developed two strategies: hard and soft partitions. Since the hard version can be seen as a special case of the soft version, we introduce the soft one first. In the soft manner, data points are allowed to be assigned to more than one partition. The importance of the data points on representing the local distribution of the centers is evaluated by a weighting kernel (e.g. Normal kernel in Figure 2.1 (e)) on the centers. The partition result of the soft version of the Fast Parzen Window (FPW-S) is illustrated in Figure 2.1 (d). For the hard version, data points that fell in the overlapping regions are assigned to its nearest partition center. Figure 2.1 (b) illustrates the hard version partition. It can be known that the hard version is a simplified soft one by replacing the weighting kernel with a simple uniform kernel shown in Figure 2.1 (c).

2.2.2 Estimation of the locally fitted components

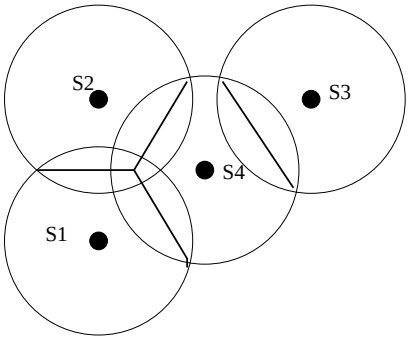
Some kernel-based estimators use diagonal Gaussians as mixture components. However, if the true density that we would like to model is actually 'close to' a lower dimensional manifold embedded in high dimensional input space, spherical Gaussians will spread their density mass equally along all input space directions, thus giving too much probability mass to irrelevant regions of the data space. Therefore, we employ a full covariance Gaussian to model local distribution instead.

With different weighting kernels, we implement the Fast Parzen Window algorithm in both hard and soft versions:

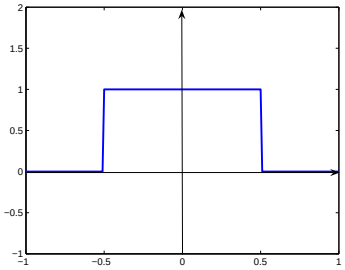
- **Hard version:**



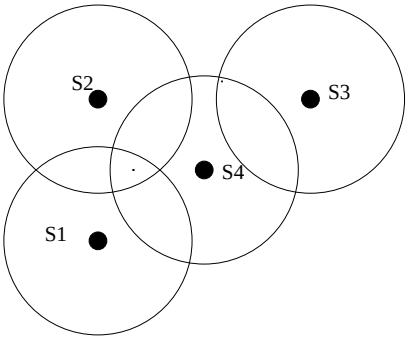
(a) Original partition from the proposed algorithm



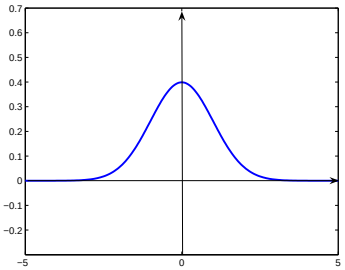
(b) Hard version partition of the efficient and sparse density estimator



(c) Simple weight Kernel



(d) Soft version partition of the efficient and sparse density estimator



(e) Normal weight Kernel

Figure 2.1: Illustration of the hard and soft partitioning strategies used in FPW density estimator.

In the hard partition version (FPW-H), we determine the mean and the covariance matrix of each Gaussian kernel using

$$\mathbf{m}_j = \frac{1}{|\mathcal{S}_j|} \sum_{\mathbf{x}_i \in \mathcal{S}_j} \mathbf{x}_i \tag{2.9}$$

$$C_j = \frac{1}{|\mathcal{S}_j|} \sum_{\mathbf{x}_i \in \mathcal{S}_j} (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T, \tag{2.10}$$

where $|\mathcal{S}_j|$ denotes the size of $\mathcal{S}_j$. The weights of the partition: $P(\mathcal{S}_j)$ for the hard version are computed as

$$P(\mathcal{S}_j) = \frac{|\mathcal{S}_j|}{N}. \tag{2.11}$$

- **Soft version:**

  The soft partition of $\mathcal{S}$ associates an influence weight $\kappa_h(\mathbf{x}_i|\mathbf{s}_j)$ in the partition $\mathbf{s}_j$ to any point $\mathbf{x}_i$ by the neighborhood kernel:

$$\kappa_h(\mathbf{x}_i|\mathbf{s}_j) = \frac{K_h(\mathbf{x}_i - \mathbf{s}_j)}{\sum_{n=1}^{N} K_h(\mathbf{x}_n - \mathbf{s}_j)}. \tag{2.12}$$

  where $h > 0$ is the kernel scale parameter. In our experiments, in order to avoid adding more parameters, we set h to be equal to the hyper-ball radius r.

  Then the weighted means and covariance matrices of the soft FPW version (FPW-S) are computed as follows (For clarity, we put derivations in Section 2.2.3):

$$\mathbf{m}_j = \sum_{i=1}^{N} \kappa_h(\mathbf{x}_i|\mathbf{s}_j)\mathbf{x}_i, \tag{2.13}$$

$$C_j = \sum_{i=1}^{N} \kappa_h(\mathbf{x}_i|\mathbf{s}_j)(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T. \tag{2.14}$$

  As points can be assigned to more than one partition with different weights, the following $P(\mathcal{S}_i)$ is used for soft version:

$$P(\mathcal{S}_j) = \frac{\sum_{n=1}^{N} K_h(\mathbf{x}_n - \mathbf{s}_j)}{\sum_{i=1}^{M}\sum_{n=1}^{N} K_h(\mathbf{x}_n - \mathbf{s}_i)}. \tag{2.15}$$

Even though the sums in Eqs. (2.12)-(2.14) run through all data points, in practice we only consider only a small fraction of points where the corresponding kernel values or responsibilities $\kappa_h(\mathbf{x}_i|\mathbf{s}_j)$ are larger than some small predefined threshold value (in

our experiments 0.00001).

It should be noted that calculating the **inverse** of the covariance matrices (in Eq. (2.8)) may be complicated as C_j's may be ill-conditioned. A common way to deal with this problem is to add a small isotropic (spherical) Gaussian noise of variance γ^2 in all directions, which is done by simply adding γ^2 to the diagonal of the covariance matrix: $C_j = C_j + \gamma^2 I$, where I is an identity matrix. In our experiments $\gamma^2 = 0.00001$.

Since the number of kernels has been reduced to the number M of the partitions, fitting each kernel to its local distribution improves the performance of the Fast Parzen Windows density estimator without increasing the time and memory costs significantly.

2.2.3 Derivation of FPW-S's parameters

To estimate the parameters $\mathbf{m}_j$, C_j and $P(\mathcal{S}_j)$ of the j-th mixture component in the FPW-S model, we introduce a component specific local log-likelihood of j-th mixture component $\mathcal{L}_j$. The sample points are weighted through kernel $K_h(\mathbf{x} - \mathbf{s}_j)$ in the component specific local log-likelihood and it is written as:

$$\mathcal{L}_j(\theta) = \sum_{i=1}^{N} K_h(\mathbf{x}_i - \mathbf{s}_j) \ \log[P(\mathcal{S}_j)\ \mathcal{N}(\mathbf{x}; \mathbf{m}_j, C_j)], \tag{2.16}$$

where θ collects all the free parameters (means, covariances and mixture coefficients of the mixture components).

The optimal $\mathbf{m}_j$ and C_j are obtained by differentiating $\mathcal{L}_j$ with respect to $\mathbf{m}_j$, C_j and setting the derivatives to zero. We have

$$\begin{aligned} \mathbf{m}_j &= \frac{\sum_{i=1}^{N} K_h(\mathbf{x}_i - \mathbf{s}_j)\ \mathbf{x}_i}{\sum_{n=1}^{N} K_h(\mathbf{x}_n - \mathbf{s}_j)} \\ &= \sum_{i=1}^{N} \frac{K_h(\mathbf{x}_i - \mathbf{s}_j)}{\sum_{n=1}^{N} K_h(\mathbf{x}_n - \mathbf{s}_j)}\ \mathbf{x}_i \\ &= \sum_{i=1}^{N} \kappa_h(\mathbf{x}_i | \mathbf{s}_j)\ \mathbf{x}_i. \end{aligned} \tag{2.17}$$

Analogously, we obtain

$$\begin{aligned} C_j =& \frac{\sum_{i=1}^{N} K_h(\mathbf{x}_i - \mathbf{s}_j)(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T}{\sum_{q=1}^{N} K_h(\mathbf{x}_q - \mathbf{s}_j)} \\ =& \sum_{i=1}^{N} \frac{K_h(\mathbf{x}_i - \mathbf{s}_j)}{\sum_{q=1}^{N} K_h(\mathbf{x}_q - \mathbf{s}_j)} (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T \\ =& \sum_{i=1}^{N} \kappa_h(\mathbf{x}_i|\mathbf{s}_j)\ (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^T. \end{aligned} \tag{2.18}$$

When solving for mixing coefficients $P(\mathcal{S}_j)$, the local log-likelihood $\mathcal{L}_j$ need to be extended with Lagrange multiplier to ensure the mixing coefficients summing up to unity:

$$\tilde{\mathcal{L}}_j(\theta) = \mathcal{L}_j(\theta) + \lambda \left(\sum_{j=1}^{M} P(\mathcal{S}_j) - 1 \right). \tag{2.19}$$

Differentiating $\tilde{\mathcal{L}}_j$ w.r.t. $P(\mathcal{S}_j)$ and solving for the Lagrange multiplier λ we obtain

$$P(\mathcal{S}_j) = \frac{\sum_{i=1}^{N} K_h(\mathbf{x}_i - \mathbf{s}_j)}{\sum_{q=1}^{M} \sum_{i=1}^{N} K_h(\mathbf{x}_i - \mathbf{s}_q)}. \tag{2.20}$$

The mixture coefficients $P(\mathcal{S}_j)$ represent the effective weight of points in the corresponding regions of the data space, given the smoothing kernels $K_h(\mathbf{x} - \mathbf{s}_j)$. The hard version of FPW, FPW-H, can be viewed as a simple efficient approximation to FPW-S.

2.3 Theoretical investigation of FPW-S

In this section, we theoretically analyze the performance of the FPW-S in univariate case by connecting it to a non-parametric density estimator $\hat{f}(x)$ (locally parametric density estimator) with parametric overtones proposed in [1]. Investigation of the properties of the latter estimator shows that it has approximately the same variance as the ordinary kernel method but potentially a smaller bias. For many situations it will be seen that [1]

$$\begin{aligned} \mathbf{E}\hat{f}(x) =& \quad f(x) + \frac{1}{2}\sigma_K^2 h^2 b(x) + O(h^4 + (Nh)^{-1}), \\ \mathbf{Var}\hat{f}(x) =& \quad R(K)(Nh)^{-1} f(x) - N^{-1} f(x)^2 + O(h/N) \end{aligned} \tag{2.21}$$

where $\mathbf{E}\hat{f}(x)$ and $\mathbf{Var}\hat{f}(x)$ are the expectation and the variance of the estimated density $\hat{f}(x)$, respectively, $\sigma_K^2 = \int zK(x)dz$ and $R(K) = \int K(z)^2 dz$. From (2.21), the bias of the estimated $\hat{f}(x)$ is written with a bias factor function $b(x)$ as follows:

$$\mathbf{E}\hat{f}(x) - f(x) = \frac{1}{2}\sigma_K^2 h^2 b(x) + O(h^4 + (Nh)^{-1}).$$

Different from the bias of classical kernel density estimator, the bias factor function $b(x)$ is related to $f''(x)$, characteristics of the parametric class and the weight functions used in estimating the density.

In [1], to estimate the locally parametric density $\hat{f}(x)$ for each given data x, the author defined a local kernel-smoothed likelihood function to estimate the best local parametric approximant to the true density. Around each given x, the local log likelihood [79] is defined as

$$\begin{aligned}\mathcal{L}_n(x,\theta) =& \int K_h(t-x)\left\{\log f(t,\theta)dF_N(t) - f(t,\theta)dt\right\} \\ =& \frac{1}{N}\sum_{i=1}^{N} K_h(x_i - x)\log f(x_i,\theta) - \int K_h(t-x)f(t,\theta)dt\end{aligned} \tag{2.22}$$

where F_N is the empirical distribution function, $f(\cdot,\theta)$ is the given parametric model with parameters θ.

In order to make the local parametric approximant $\hat{f}(x)$ comparable to our FPW-S, we choose the parametric model for the local model $f(t;\theta)$ with parameters a, μ and δ

$$f(t;a,\mu,\delta) = \frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2})\right\}$$

and Gaussian as the smooth kernel with the width h

$$K_h(x_i - x) = \frac{1}{\sqrt{2\pi}h}\exp\left\{-\frac{(x_i - x)^2}{2h^2}\right\}.$$

Then, we derive the density estimation $\hat{f}(x)$ from maximizing the local likelihood and write it in terms of the Parzen Windows estimation $\tilde{f}(x)$ as follows:

$$\hat{f}(x) = \tilde{f}(x)\,Q^{-1/2}\exp\left\{-\frac{1}{2}\left(\frac{\tilde{f}'(x)}{\tilde{f}(x)}\right)^2 Q^{-1}h^2\right\} \tag{2.23}$$

where $\tilde{f}(x)$ is the Parzen Windows estimation with kernel width h, i.e.,

$$\tilde{f}(x) = \frac{1}{N}\sum_{i=1}^{N} K_h(x_i - x)$$

and Q is learnt from the estimated variance δ as follows:

$$Q = \frac{\delta^2}{\delta^2 + h^2} = 1 - \frac{h^2}{\delta^2 + h^2} = 1 - h^2\left\{\left(\frac{\tilde{f}'(x)}{\tilde{f}(x)}\right)^2 - \frac{\tilde{f}''(x)}{\tilde{f}(x)}\right\}.$$

Furthermore, we obtain the estimated $\hat{f}(x)$ as

$$\begin{aligned}\hat{f}(x) =& \tilde{f}(x)\sqrt{\frac{\tilde{f}^2(x)}{\tilde{f}^2(x) + h^2\tilde{f}''(x)\tilde{f}(x) - h^2\tilde{f}'^2(x)}} \\ & \exp\left\{-\frac{1}{2}\frac{h^2\tilde{f}'^2(x)}{\tilde{f}^2(x) + h^2\tilde{f}''(x)\tilde{f}(x) - h^2\tilde{f}'^2(x)}\right\}.\end{aligned} \tag{2.24}$$

(see appendix .1 for detailed derivations).

2.3.1 Theoretical analysis of 1-D FPW-S

Here we investigate the performance of our proposed FPW-S in 1-D situation. The 1-D format of (2.7) is

$$\begin{aligned}\hat{p}(x;h) &= \sum_{j=1}^{M} P(\mathcal{S}_j)\mathcal{N}(x;\mu_j,\delta_j) \\ &= \sum_{j=1}^{M}\frac{\sum_{i=1}^{N} K_h(x_i - s_j)}{\sum_{q=1}^{M}\sum_{i=1}^{N} K_h(x_i - s_q)}\mathcal{N}(x;\mu_j,\delta_j) \\ &= \frac{1}{\sum_{q=1}^{M}\sum_{i=1}^{N} K_h(x_i - s_q)}\sum_{j=1}^{M}\sum_{i=1}^{N} K_h(x_i - s_j)\mathcal{N}(x;\mu_j,\delta_j)\end{aligned} \tag{2.25}$$

where μ_j and δ_j^2 are the mean and the variance of the jth 1-D Gaussian distribution.

Let $\alpha_j = \frac{1}{N}\sum_{i=1}^{N} K_h(x_i - s_j)$, we rewrite (2.25) in the following way:

$$\begin{aligned}\hat{p}(x;h) =& \frac{1}{\sum_{q=1}^{M}\alpha_q}\sum_{j=1}^{M}\alpha_j\mathcal{N}(x;\mu_j,\delta_j) \\ =& \sum_{j=1}^{M} g(x,\alpha_j,\mu_j,\delta_j).\end{aligned} \tag{2.26}$$

Note that here we defined

$$g(x, \alpha_j, \mu_j, \delta_j) = \frac{\alpha_j}{\sum_{q=1}^{M} \alpha_q} \frac{1}{\sqrt{2\pi}\delta_j} \exp\left\{ -\frac{(x-\mu_j)^2}{2\delta_j^2} \right\}.$$

According to (2.16), the parameters of the component $g(x, \alpha_j, \mu_j, \delta_j)$ can be estimated from the 1-D formulation of the component specific local log-likelihood, i.e.

$$\begin{aligned} \mathcal{L}_j(\theta) &= \sum_{i=1}^{N} K_h(x_i - s_j) \ \log\left[P(\mathcal{S}_j)\, \mathcal{N}(x; \mu_j, \delta_j)\right] \\ &= \sum_{i=1}^{N} K_h(x_i - s_j) \ \log\left[\frac{\alpha_j}{\sum_{q=1}^{M} \alpha_q}\, \mathcal{N}(x; \mu_j, \delta_j)\right]. \end{aligned} \tag{2.27}$$

It will lead to

$$\alpha_j = \frac{1}{N} \sum_{i=1}^{N} K_h(x_i - s_j) \tag{2.28}$$

$$\mu_j = \frac{\sum_{i=1}^{N} K_h(x_i - s_j) x_i}{\sum_{n=1}^{N} K_h(x_n - s_j)} \tag{2.29}$$

$$\delta_j^2 = \frac{\sum_{i=1}^{N} K_h(x_i - s_j)(x_i - \mu_j)^2}{\sum_{n=1}^{N} K_h(x_n - s_j)}. \tag{2.30}$$

In the soft version of our proposed FPW density estimator, we use the Gaussian kernel as the local smooth kernel, i.e.,

$$K_h(x_i - s_j) = \frac{1}{\sqrt{2\pi}h} \exp\left\{ -\frac{(x_i - s_j)^2}{2h^2} \right\}.$$

Then, the parameters estimated by Eqs. (2.28)-(2.30) can be rewritten in terms of the density of the classical Parzen Windows estimator $\tilde{f}(s_j)$ with the window width h as follows: (see Appendix .2 for derivation details):

$$\alpha_j = \tilde{f}(s_j), \tag{2.31}$$

$$\mu_j = \frac{h^2 \tilde{f}'(s_j)}{\tilde{f}(s_j)} + s_j, \tag{2.32}$$

$$\delta_j^2 = \frac{h^2 \tilde{f}^2(s_j) + h^4 \tilde{f}''(s_j)\tilde{f}(s_j) - h^4 \tilde{f}'^2(s_j)}{\tilde{f}^2(s_j)}. \tag{2.33}$$

Let $g_j(x)$ represent the component of FPW-S associated with the partition s_j: $g(x; \alpha_j, \mu_j, \delta_j)$,

the density estimated by our FPW-S on the partition center s_j is the sum of the contribution from component $g(x;\alpha_j,\mu_j,\delta_j)$ and other ones, i.e.

$$\hat{p}(s_j;h) = g_1(s_j) + g_2(s_j) + \ldots + g_j(s_j) + \ldots + g_M(s_j).$$

Instead of analyzing the overall accuracy, we investigate the contribution of component $g_j(s_j)$, which is believed providing the major contribution among these components. Bringing the parameters estimated by Eqs. (2.31)-(2.33) to the formulation of component $g_j(s_j)$ of the FPW-S density estimator:

$$g_j(s_j) = \frac{\alpha_j}{\sum_{q=1}^{M}\alpha_q}\frac{1}{\sqrt{2\pi}\delta_j}\exp\left\{-\frac{(s_j-\mu_j)^2}{2\delta_j^2}\right\}, \tag{2.34}$$

we have the estimated component $\hat{g}_j(s_j)$ in terms of the classical Parzen Windows estimator $\tilde{f}(x)$ as:

$$\begin{aligned}\hat{g}_j(s_j) =& \hat{g}(s_j;\alpha_j,\mu_j,\delta_j)\\ =& \frac{\tilde{f}(s_j)}{\sqrt{2\pi}\sum_{q=1}^{M}\tilde{f}(s_q)}\sqrt{\frac{\tilde{f}^2(s_j)}{h^2\tilde{f}^2(s_j)+h^4\tilde{f}''(s_j)\tilde{f}(s_j)-h^4\tilde{f}'^2(s_j)}}\\ & \exp\left\{-\frac{1}{2}\frac{h^2\tilde{f}'^2(s_j)}{\tilde{f}^2(s_j)+h^2\tilde{f}''(s_j)\tilde{f}(s_j)-h^2\tilde{f}'}\right\}.\end{aligned} \tag{2.35}$$

Let $A(x) = \tilde{f}^2(x) + h^2\tilde{f}''(x)\tilde{f}(x) - h^2\tilde{f}'^2(x)$ and $x = s_j$, we get the comparable results between the j-th component of our FPW-S and density estimator in [1] shown in Figure 2.2.

$$\begin{aligned}\hat{g}_j(s_j) &= \frac{1}{\sqrt{2\pi}\sum_{q=1}^{M}\tilde{f}(s_q)h}\tilde{f}(s_j)\sqrt{\frac{\tilde{f}^2(s_j)}{A(x)}}\exp\left\{-\frac{1}{2}\frac{h^2\tilde{f}'^2(s_j)}{A(x)}\right\}.\\ \hat{f}(s_j) &= \tilde{f}(s_j)\sqrt{\frac{\tilde{f}^2(s_j)}{A(x)}}\exp\left\{-\frac{1}{2}\frac{h^2\tilde{f}'^2(s_j)}{A(x)}\right\}.\end{aligned}$$

Figure 2.2: Theoratical analysis results of the component $\hat{g}_j(s_j)$ in FPW-S algorithm and the locally parametric density estimator $\hat{f}(s_j)$ [1].

Therefore, the relation between the component $\hat{g}_j(s_j)$ of the FPW-S and locally

parametric nonparametric model learnt from local log likelihood in [1] is

$$\hat{g}_j(s_j) = \frac{1}{\sqrt{2\pi}h\sum_{q=1}^{M}\tilde{f}(s_q)}\hat{f}(s_j). \tag{2.36}$$

If we use εr to represent (approximate) the distance between the components' centers, the relation between component $\hat{g}_j(s_j)$ of our FPW-S and locally parametric nonparametric model $\hat{f}(s_j)$ shown in (2.36) could be reformulated as:

$$\hat{g}_j(s_j) = \frac{\varepsilon r}{h\sqrt{2\pi}\sum_{q=1}^{M}\tilde{f}(s_q)\varepsilon r}\hat{f}(s_j) \tag{2.37}$$

Note that, the kernel width h is equal to the fixed radii r used at the partitioning step in our algorithm. If r is a very small number, the number of the partitions M will be large. This is guaranteed by the partition algorithm in Section 2.2.1. If r is approaching 0 (when the number of data points N goes to infinity), M will tend to infinity. In this case, the term $\sum_{q=1}^{M}\tilde{f}(s_q)\varepsilon r$ approximates $\int f(x)dx$, which equals to 1. Therefore,

$$\begin{aligned}\hat{g}_j(s_j) =& \frac{\varepsilon}{\sqrt{2\pi}\sum_{q=1}^{M}\tilde{f}(s_q)\varepsilon r}\hat{f}(s_j)\\ \approx& \frac{\varepsilon}{\sqrt{2\pi}}\hat{f}(s_j).\end{aligned} \tag{2.38}$$

From Eq. (2.38) and (2.21), we obtain the expectation and variance the estimated component $\hat{g}_j(s_j)$ on partition center s_j as follows:

$$\begin{aligned}\mathbf{E}\hat{g}_j(s_j) =& \frac{\varepsilon}{\sqrt{2\pi}}\mathbf{E}\hat{f}(s_j)\\ \mathbf{Var}\hat{g}_j(s_j) =& \frac{\varepsilon^2}{2\pi}\mathbf{Var}\hat{f}(s_j).\end{aligned} \tag{2.39}$$

Therefore, we could claim that the expectation and variance of the component $g_j(s_j)$ of our FPW-S at the partition center s_j have similar properties to that of the locally parametric density estimator $\hat{f}(s_j)$ (up to a scale which is independent of the parameter of density estimation r). The bias of the estimated component $\hat{g}_j(s_j)$ is also sensitive to the curvature of the true density and also related to the characteristics of the kernel form and the weight functions.

2.3.2 Numerical experiments

This subsection aims to verify the theoretical findings by some numerical simulations. The datasets we used are generated from 1-D density distribution

$$f(x) = \frac{6}{10}\mathcal{N}(-2.6, 0.36) + \frac{4}{10}\mathcal{N}(1.7, 0.36), \tag{2.40}$$

where $\mathcal{N}(\mu, \delta^2)$ denotes the normal distribution with mean μ and variance δ^2. In the experiments, two different ways of selecting the partition centers are employed. One is the partition algorithm proposed in Section 2.2.1. Another one is utilizing the equally spaced partition centers.

The first experiment is to confirm the relation between the number of the observations N and the algorithm parameter r (h). That is with more data samples (N), the r (or h) goes small and the number of the components M in FPW density estimator goes large. For the equally spaced partitioning scheme, we use r' to represent interval of the partition centers and M' to represent the number of the partition centers. The estimated optimal parameters (by 10-fold cross validation) with respect to different number of observations in the dataset are reported in Table 2.1. It shows that

1. Analogously to classical density estimators, for both partition schemes, the size of the partition is getting smaller as the number of the observation is getting larger. At the same time, the number of partition centers must increase to ensure the coverage of the data space.

2. Our partition algorithm requires smaller number of centers than the equally spaced partitioning scheme in the density estimation.

In the second experiment, we aim to illustrate relation between the main contributed component (component centered on partition center s_j) in FPW-S and locally parametric density, which is formulated in Eq. (refeq: grelatedf). To do this, we generate 800 data points from $f(x)$ and use 42 equally spaced centers as the partition centers. At each partition center, we estimate the densities by local parametric density model and the FPW-S component centered on it. The theoretical relation between them is formulated in Eq. (2.38). In Figure 2.3, we use red square markers to represent the density estimated at the fixed centers by local parametric density (denoted by LL in the figure). Accordingly, blue circles are plotted to indicate the the contribution

Table 2.1: The optimal parameters (set by 10-fold cross validation) of FPW-S for two different partition centers selecting schemes with respect to the increased number of the observations sampled from the distribution (2.40).

N	r′	M′	r	M
40	0.7173	12	0.8607	11
80	0.6244	18	0.8326	14
160	0.5436	23	0.6523	20
320	0.4417	31	0.5679	25
800	0.3677	42	0.4728	32
1600	0.3200	50	0.4116	40

from its main contributed component (component centered on the partition center) by our proposed FPW-S. We can see that the densities represented by blue markers is the densities represented by the red markers scaled down by a constant factor, which is consistent with our theoretical finding (2.38).

Finally, since our preliminary theoretical work does not reflect the accuracy of the overall density estimated by FPW-S, we would like to do some numerical comparisons on the expectation and bias of the density value between our FPW-S and the classical PW.

In Figure 2.4, we plot the expectation (subplot (a)) and bias (subplot (b)) of the density values estimated at the fixed partition centers of FPW-S by classical Parzen Windows and FPW-S, respectively. The kernel width h of the Parzen Windows is selected as $h = 0.0525$ (estimated by 10-fold cross validation). The space width of the FPW-S is $r = 0.3677$ (from table 2.1). The calculation of the estimated expectation and bias on the fixed partition centers is averaged on 1000 repetitions of the experiments.

Similarly, the expectation and bias of the density value estimated by our FPW-S with using the flexible centers ($r = 0.4728$) selected by partitioning algorithm in the 1000 repetitions is plotted in Figure 2.5. From Figures 2.4 and 2.5, we notice that, similar to the classical Parzen Windows, the bias of the FPW-S with both fixed partition centers and partition centers selected by the algorithm proposed in Section 2.2.1 is related to the curvature of the true density. However, since the FPW-S with partition centers selected by the proposed algorithm uses less components to represent the density compare to the FPW-S with fixed partition centers, it has larger bias.

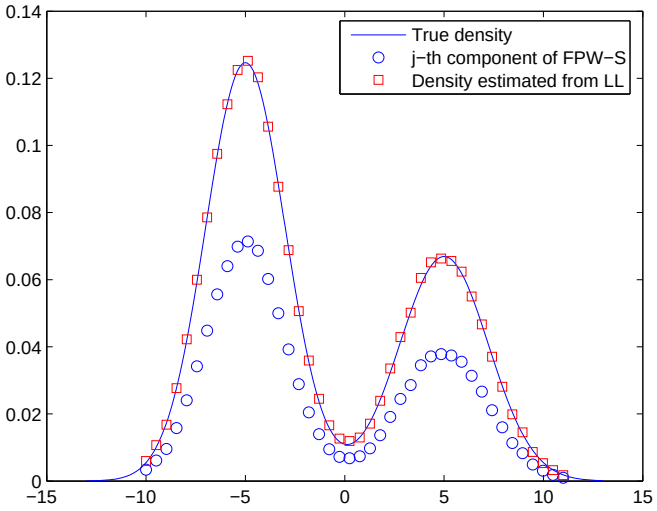
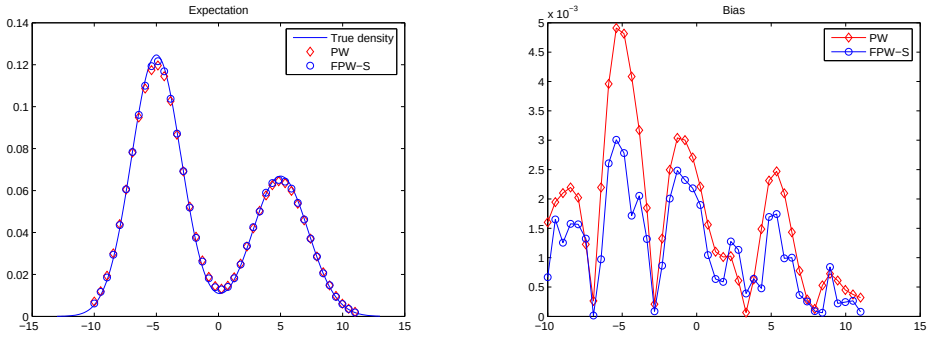


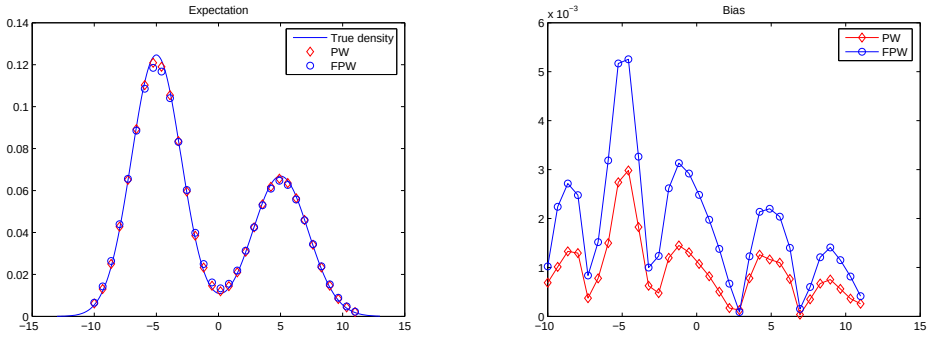
Figure 2.3: The underlying distribution (solid line) and density estimated by different density estimators. The squares markers represent the density of the partition centers s_j estimated by locally parametric density estimator: $\hat{f}(s_j)$ and the circle markers show the density obtained from the center's component: $\hat{g}_j(s_j)$ of 1-D FPW-S.



(a) Expectation of the 1-D FPW-S and the classical PW

(b) Bias of the 1-D FPW-S and the classical PW

Figure 2.4: Expectations and biases of the classical PW and the 1-D FPW-S with equally spaced partition centers.



(a) Expectation of the 1-D FPW-S and the classical PW

(b) Bias of the 1-D FPW-S and the classical PW

Figure 2.5: Expectations and biases of the classical PW and the 1-D FPW-S with partition centers selected by the proposed algorithm.

2.4 Experiments

In this section, we test our presented FPW method on two synthetic datasets and real large scale astronomical simulation datasets. The experiments demonstrate that, compared with other recently proposed methods for sparse kernel density estimation, our Fast Parzen Windows can lead to similar or better performance in terms of the accuracy and the sparseness of representation, but at much less computational cost. Furthermore, the advantages of our simple, yet effective method for density estimation are highlighted in the experiments on large datasets of galaxy disruption simulations.

2.4.1 One-Dimensional Example

Our first experiment is carried out on a 1D dataset used in [66]. 1800 points are drawn from the Gaussian mixture

$$f(x) = \frac{8}{18}\mathcal{N}(-2.6, 0.09) + \frac{6}{18}\mathcal{N}(-0.8, 0.36) + \frac{4}{18}\mathcal{N}(1.7, 0.64),$$

where $\mathcal{N}(\mu, \delta^2)$ denotes the normal distribution with mean μ and variance δ^2.

We compare the performance of Parzen Windows (PW), Simplified Mixture Model (SMM) [66], Reduced Set Density Estimator (RSDE) [65], and our Fast Parzen Windows (FPW) on estimation accuracy, representation sparseness and running time. Since the locally parametric density estimator [1] mentioned earlier is not able to estimate

the overall density distribution of the given dataset, we will not consider it in this section.

The underlying distribution and typical estimated distributions are illustrated in Figure 2.6. Parzen Windows employs Gaussian kernels with fixed bandwidth $h = 0.12$ (determined by 10-fold cross validation). For SMM, we use the same parameters as in [66], approximate the density by 5 kernels and repeatedly initialize the estimation by the K-means algorithm [80]. We also try the number of kernels 10, 50, 100, 250. The free parameter of RSDE (i.e. optimal kernel width) is set to 0.28 by 10-fold cross validation, and then the weighting coefficients are learnt in the optimization procedure. The proposed FPW-S uses the disk radius $r = 0.19$ (also set by 10-fold cross validation).

To have a quantitative evaluation, we randomly generate datasets from the Gaussian mixture 100 times, and compare these algorithms using the following error criteria with respect to the real distribution $f(\mathbf{x})$: 1) the L_1 error (2.41); 2) the L_2 error (2.42); and 3) an approximation of the standard KL divergence (2.43):

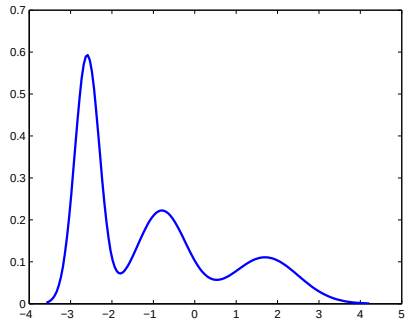
$$\epsilon_{L1} = \frac{1}{N}\sum_{i=1}^{N} |f(\mathbf{x}_i) - \hat{f}(\mathbf{x}_i)| \tag{2.41}$$

$$\epsilon_{L2} = \frac{1}{N}\sum_{i=1}^{N} |f(\mathbf{x}_i) - \hat{f}(\mathbf{x}_i)|^2 \tag{2.42}$$

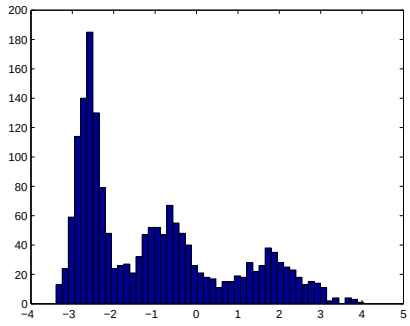
$$\epsilon_{KL} = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) \log \frac{f(\mathbf{x}_i)}{\hat{f}(\mathbf{x}_i)}, \tag{2.43}$$

where $\hat{f}(\mathbf{x}_i)$ is the estimated density. The results are reported in Table 2.2. We also investigate the sparseness of the methods by reporting the average number of components in the models. It determines the computational cost of estimating density for the testing data points. Parzen Windows leads to the largest model, since the number of components (kernels) is equal to the sample size N. All the experiments are performed on desk top computer having Pentium(R)4, 3.00GHz CPU and 512M RAM, and the algorithms are coded in Matlab 7.0. Table 2.3 shows the running time consumed on this 1800-sample dataset and the model complexity (i.e. the number of components, denoted by M). We report the time consumed in the form $t_1 + t_2$, where t_1 is the time used for building the model and t_2 is the time of deploying the model[1].

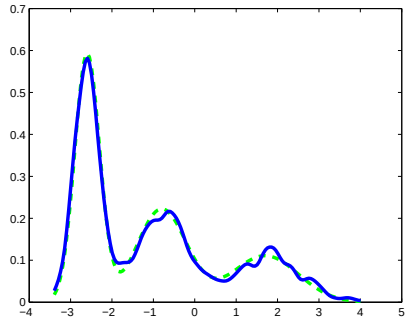
[1]Time for parameter estimation (e.g. 10-fold cross validation) is not included.



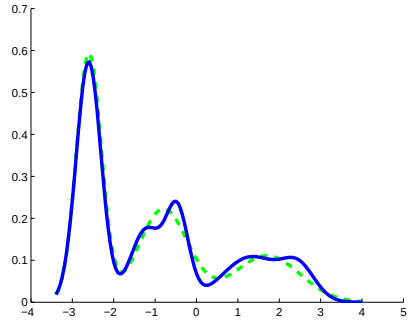
(a) True underlying distribution



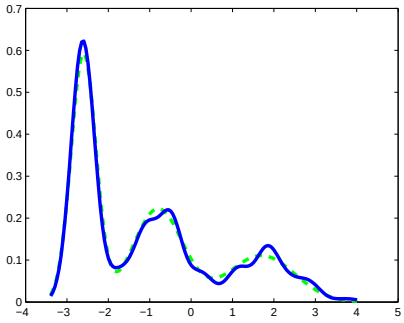
(b) Histogram



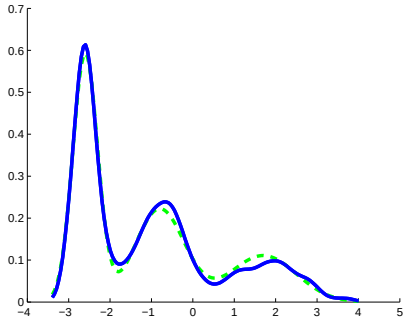
(c) Parzen Window



(d) SMM



(e) RSDE



(f) FPW-S

Figure 2.6: The underlying distribution and typical estimated distributions by different approaches. Green-dashed: real density distribution; Blue: estimated density.

Table 2.2: Error measures of density estimated by PW, SMM, RSDE and FPW-S on the 1-D dataset

Methods	L_1 error (mean±std) $\times 10^{-2}$	L_2 error (mean±std) $\times 10^{-4}$	KL distance (mean±std) $\times 10^{-2}$
PW	0.89 ±0.46	0.99 ±0.97	0.88 ±0.46
SMM (M=5)	1.87 ± 1.19	4.89 ± 7.70	1.85 ± 1.17
SMM (M=10)	1.7 ± 0.6	3.22 ± 2.25	1.69 ± 0.6
SMM (M=50)	0.98 ± 0.62	1.32 ± 1.48	0.97 ± 0.62
SMM (M=100)	0.94 ±0.63	1.22 ±1.36	0.95 ± 0.59
SMM (M=250)	0.91± 0.55	1.11 ± 1.28	0.90 ± 0.55
RSDE	**0.50±0.28**	**0.32±0.36**	**0.49±0.28**
FPW-S	1.74±0.72	3.54±2.80	1.73±0.71

Table 2.3: Time elapsed (in CPU seconds) on estimating and deploying the density distribution models by using PW, SMM, RSDE, FPW-S and the number of components in each density model.

Methods	CPU Time (sec)	M
PW	0+76.695	1800
SMM	360.006 + 0.012	**5**
RSDE	289.552+ 1.481	270
FPW-S	**0.440+0.021**	14

The RSDE algorithm achieves the lowest approximation error. The performance of FPW-S is comparable to that of SMM, however, at a much lower computational cost. The hard partitioning version FPW-H tends to be more sensitive to the initialization of data space coverage by disks.

2.4.2 Data aligned along a lower dimensional manifold

The data points in the dataset (300 training and 10,000 testing) used in this experiment are two dimensional and are sampled as follows:

$$\begin{aligned} x_1 &= 0.04t\sin(t) + \epsilon_{x_1}, \\ x_2 &= 0.04t\cos(t) + \epsilon_{x_2} \end{aligned}$$

where $t \sim \mathcal{U}(3, 15)$, $\epsilon_{x_1} \sim \mathcal{N}(0, 0.01)$, $\epsilon_{x_2} \sim \mathcal{N}(0, 0.01)$, $\mathcal{U}(a, b)$ is the uniform distribution over the interval (a, b) and $\mathcal{N}(0, \delta)$ is the zero-mean Gaussian distribution with standard deviation δ.

We compare the performance of Parzen Windows (PW), Simplified Mixture Model (SMM) [66], Reduced Set Density Estimator (RSDE) [65], Manifold Parzen Windows [3], Gaussian Mixture Model (GMM) (We use GMM^k to denote the model initialized with K-means as opposed to GMM initialized randomly from the data) and our approaches (FPW-H and FPW-S). We keep the same parameter setting for Manifold Parzen Windows as in [3]. Figure 2.7 shows the training set, as well as the densities estimated by these algorithms. Note the excessive 'bumpiness' and holes in Figures 2.7 (a)-(c) caused by the sparseness of the training set. As expected, the Manifold PW density in Figure 2.7 (d) is better aligned with the underlying manifold. Our approach

was not only able to learn the underlying manifold-aligned density well, but also represent it in a very sparse manner (see Figures 2.8 (a),(b) and Table 2.5). The sparseness of our presented approach will be showing significant advantages on both training and testing efficiencies when the dataset is very large.

Besides the measurements L_1, L_2 and KL distances used in Section 2.4.1, another popular used quantitative measurement for estimated density distribution is average log likelihood (ALL). By noting that the classical Kullback-Leibler distance can be written as follows:

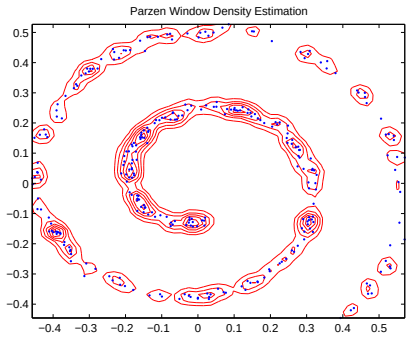
$$\begin{aligned}\int f(\mathbf{x}) \log \frac{f(\mathbf{x})}{\hat{f}(\mathbf{x})} d\mathbf{x} &= \int f(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} - \int f(\mathbf{x}) \log \hat{f}(\mathbf{x}) d\mathbf{x} \\ &= \int f(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} - \int \log \hat{f}(\mathbf{x}) dF_N(\mathbf{x}) \\ &= \int f(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x} - \frac{1}{N} \sum_{i=1}^{N} \log \hat{f}(\mathbf{x}_i),\end{aligned} \tag{2.44}$$

where $F_N(\mathbf{x})$ is the empirical distribution function. Since $\int f(\mathbf{x}) \log f(\mathbf{x}) d\mathbf{x}$ is a constant, the average log likelihood (ALL) is linear proportional to the Kullback-Leibler distance. The higher average log likelihood will lead to the smaller KL distance. Since in the real density estimation problems, the true probability density function is unknown, maximizing the average log likelihood (ALL) of the density modoel is equivalent to minimizing its KL distance to the true density. Therefore, we use the average (across 30 randomized initializations) log likelihood (ALL) on the test dataset as the quantitative comparisons of the models in the following experiments.

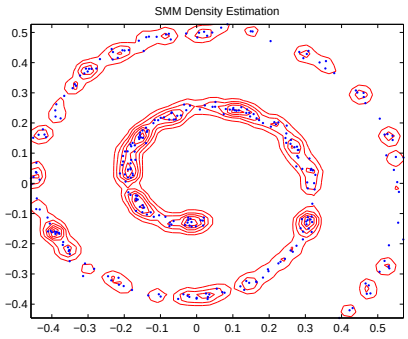
The performance measures along with the parameters used (estimated via 10-fold cross validation) are listed in Table 2.4. We use another column (i.e., *Signif*) to indicate if either FPW-H (h) or FPW-S (s) shows significantly better performance than the corresponding algorithm (as detected by t-test at significance level 0.01). The results reveal that our presented FPW performs much better than SMM and GMM generally in terms of the ALL measure.

Table 2.5 shows the running time consumed on this 300-sample dataset and the model complexity (by the number of the components, denoted by M). We report the time consumed in the form $t_1 + t_2$, where t_1 is the time used for building the model and t_2 is the time of deploying the model [1]. It is clear that the proposed work has

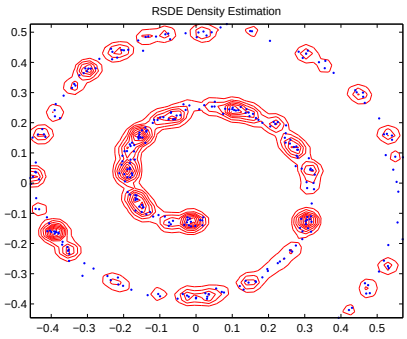
[1]Note that time for parameter estimation, like 10-fold cross validation used here, is not included.



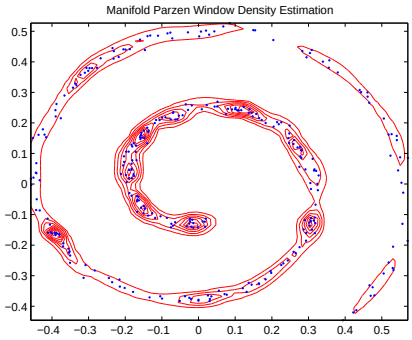
(a) Density estimated by PW



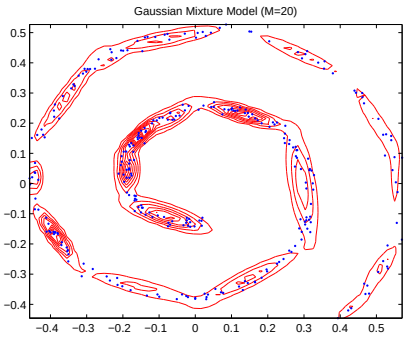
(b) Density estimated by SMM



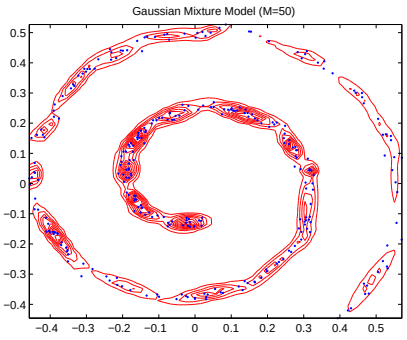
(c) Density estimated by RSDE



(d) Density estimated by Manifold PW

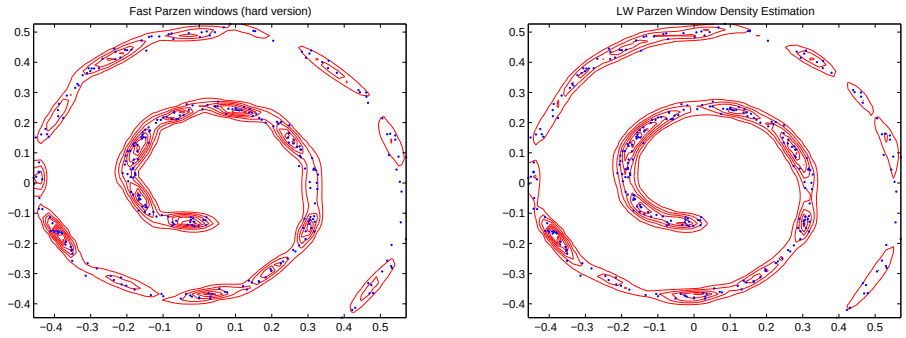


(e) Density estimated by GMM (M=20)



(f) Density estimated by GMM (M=50)

Figure 2.7: Estimated density distributions of the 2-D dataset aligned along a 1-D manifold by using different approaches: blue dots represent the data points, red contour shows the density estimated.



(a) Density estimated by FPW-H (b) Density estimated by FPW-S

Figure 2.8: Estimated density distributions of the 2-D dataset aligned along a 1-D manifold by using different approaches: blue dots represent the data points, red contour shows the density estimated (continue Figure 2.7).

Table 2.4: Performance measures and the parameters settings of PW, SMM, RSDE, GMM^k GMM, FPW-S and FPW-H on the 1-D manifold aligned 2-D artificial dataset.

Methods	Parameters	ALL (mean $\pm$ std)	Signif
PW	δ=0.0173	1.3417$\pm$0	–
SMM	M=20	0.8573$\pm$0.3356	h,s
SMM	M=50	1.2126$\pm$0.0693	h,s
SMM	M=100	1.3075$\pm$0.0187	h,s
RSED	δ=0.0190	1.0515$\pm$0	–
GMM^k	M=50	1.4575$\pm$0.0243	s
GMM	M=70	1.4609$\pm$ 0.0271	s
Man PW	d=1, k=11, δ=0.09	1.4660$\pm$ 0	–
FPW-H	r=0.141, γ=0.0001	**1.3737 $\pm$ 0.0246**	s
FPW-S	r=0.051, γ=0.0001	**1.5045 $\pm$ 0.0090**	

significant advantage on the computation time.

Table 2.5: Time elapsed (in CPU seconds) on estimating and deploying the density distribution models by using PW, SMM, RSDE, GMM^k, GMM, FPW-S, FPW-H and the number of components in each density model.

Methods	M	CPU Time (sec)
PW	300	0+1.229
SMM	20	16.751 + 0.117
SMM	50	20.319 + 0.215
SMM	100	33.319 + 0.686
RSDE	112	5.299 + 0.546
GMM^k	50	9.162+0.346
GMM	70	12.078+0.463
Manifold PW	300	0.155+1.312
FPW-H	21	**0.156+0.102**
FPW-S	57	**0.580+0.231**

2.4.3 Experiments on galaxy disruption simulations

In the following experiment, we investigate the performance of our proposed FPW density estimator on a series of large-scale simulation datasets. These datasets are generated from the astronomical model which simulates the disruption process of the satellite galaxy (M32) and large galaxy (M31)[81, 82]. To track the evolution process starting from a particular initial condition, the astronomers collect the simulation datasets at successive evolution stages.

The ultimate goal of the astronomal research is to identify the most plausible set of initial conditions leading to the distribution of stars currently observed by the astronomers. Of course, particle simulations cannot be compared with the real observations on a point by point bases, but the observed density of stars can be compared with that of the simulated particles. The first step is to build good density models on the simulated data. The densities will be then projected into the observation space (typically 2 spatial coordinates + the line of sight velocity) and the likelihood given the real observations will be calculated. The full work of identifying the most plausible

simulation datasets against the observation is presented in next chapter as an application of our Fast Parzen Window density estimator. Here we are only concerned with models built in the full 6-dimensional (simulation) space.

We have 22 galaxy simulation datasets indexed from 0 to 21. They describe one disruption process sequencely. The distributions of the data in different disruption stages are differnet. The number of particles representing the satellite galaxy being disrupted is typically large (in our case $\approx$ 30,000) and we found that using RSDE was computationally infeasible. Encouraged by the results in the previous experiments, we employ the PW, GMM and our methodology as the principal density estimators.

In the first set of experiments we used 10-fold cross validation in each simulation data set to measure the quality of the estimated density models. The average log-likelihoods (ALL) on each individual set estimated by PW, our approach (FPW) and SMM are plotted in Figure 2.9 (using hyper parameters estimated by 10-fold cross validation within the training folds). The X-axes indicates the simulation set index (numbered from 0 to 21), the Y-axes shows the ALLs estimated by the models. Our

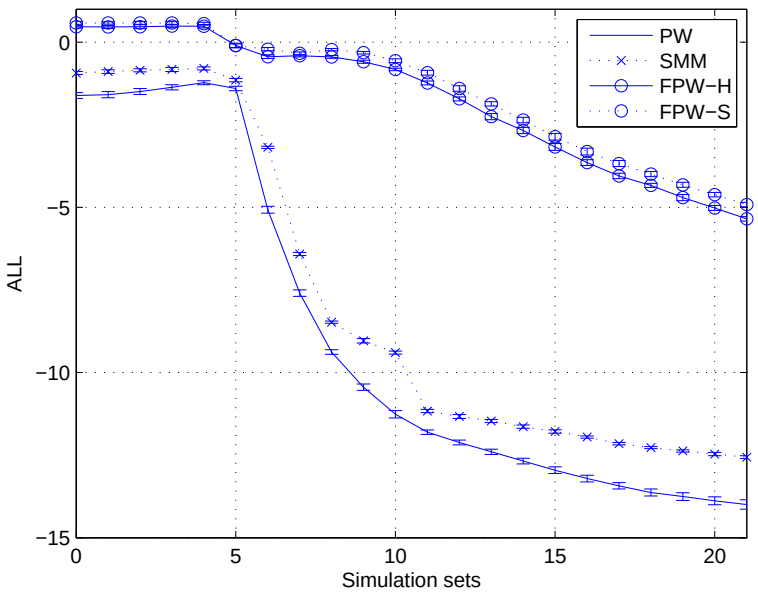


Figure 2.9: Average log-likelihood (ALL) of density models estimated by classical PW and other sparse density estimators on 22 galaxy simulation datasets.

FPW method shows a superior performance relative to both PW and SMM estimators

across all the simulation sets considered.

RSDE involves the calculations of $N \times N$ matrices. As mentioned earlier, running the algorithm on $N = 32,768$ points has been proved to be infeasible. To be able to investigate the performance of RSDE and compare it to our approach, for each of the 22 stages, we estimated the density from a downsampled simulation set (10% of the original set) and tested on the hold-out data (90% of the original set). Analogously, we demonstrate the performance of GMMs on the downsampled simulation sets due to the computational demand of E-M parameter fitting. The process was repeated 10 times. The results for RSDE, GMM and FPW are shown in Figure 2.10.

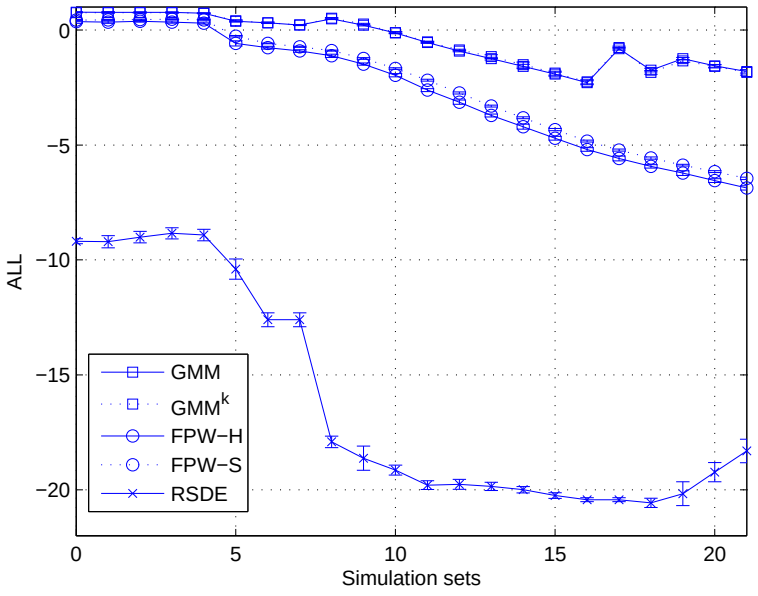


Figure 2.10: Average log-likelihood (ALL) of density models estimated by classical PW and other sparse density estimators on 22 downsampled galaxy simulation datasets.

In Figure 2.11, we report the time consumed on building the density model by using these approaches on the 22 stages of galaxy disruption. The X-axes indicates the simulation sets, and the Y-axis shows the time used (seconds). Note that RSDE, GMM and GMM^k are estimated on the downsampled simulation set (10% of the original set). A clear advantage of our approaches on running times are illustrated.

In the second set of experiments, we investigate how reliably can a stage in the galaxy disruption process be detected based on "observations" not used in the model

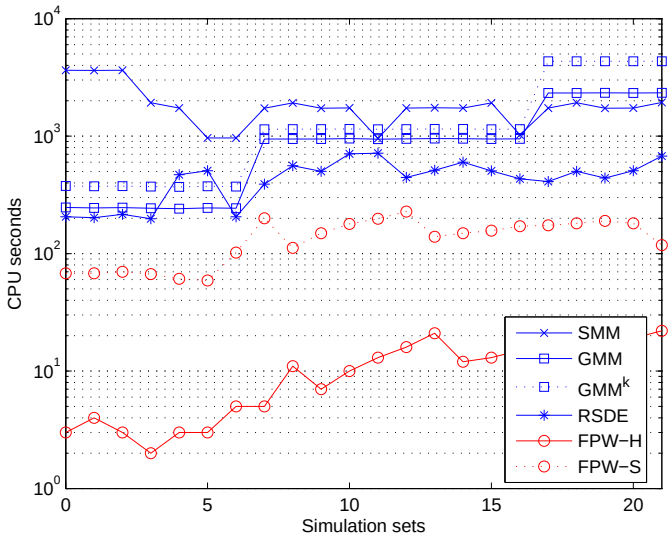


Figure 2.11: Time elapsed (in CPU seconds) on estimating the density distribution of the galaxy simulation sets by using our proposed FPW-H, FPW-S, SMM (estimated on full size simulation), GMM, GMM with K-means initialization and RSDE (estimated on downsampled simulation set).

building process. We run a rolling window of size 3 over the series of 22 simulation data sets. Thus we obtain 20 simulation set triplets (denoted by (f, g, h)), starting with simulation sets (f, g, h) at stages (0,1,2) and ending with the triplet (f, g, h) containing simulation set for stages (20,21,22). For each triplet of consecutive simulation sets (f, g, h), we estimate 3 models, one on 90% of data from f, one on 90% of data from g, and one on 90% of data from h. (We use 10% here for GMM and GMM^k) All three models are then tested on the 10% hold-out set from g. In this way we can determine how well the "true" source density g can be distinguished from the densities at the nearby stages f and h. This process is repeated 10 times. The densities corresponding to the nearby stages of galaxy disruption can be quite similar and obtaining an accurate density model is essential for further investigations by the astronomers. As an illustration, we present in Figure 2.12 two sets of 3-dimensional projections of the simulation data for the triplet $(f, g, h) = (20, 21, 22)$. The first projection is onto the spatial coordinates, the second is onto the leading 3 eigenvectors found by the Principal Component Analysis of the merged f, g and h sets.

The models constructed on the set g have the highest hold-out ALL in each of the

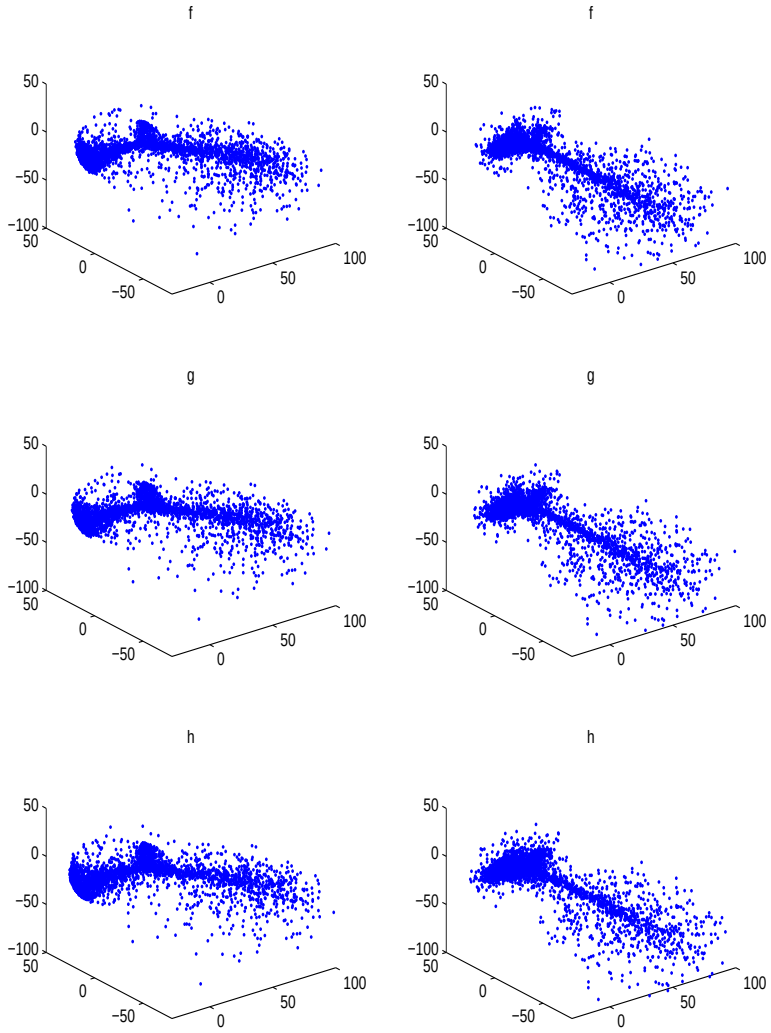


Figure 2.12: 3-D projections of the simulation data for the triplet $(f, g, h) = (20, 21, 22)$. The first projection (1st column) is onto the 3 spatial coordinates, the second (2nd column) is onto the leading 3 eigenvectors found by the Principal Component Analysis of the merged f, g and h sets.

20 triplets (f, g, h). To qualify the confidence margin with which the true source is detected, we calculate likelihood ratios (LR) :

$$\begin{aligned} LR_{fg} &= \frac{\mathcal{L}_f}{\mathcal{L}_g} \\ LR_{hg} &= \frac{\mathcal{L}_h}{\mathcal{L}_g} \end{aligned} \tag{2.45}$$

where $\mathcal{L}_f$, $\mathcal{L}_g$ and $\mathcal{L}_h$ denote the average likelihood on the testing data obtained by models built on the sets f, g and h, respectively. The smaller likelihood ratio value, the better performance the density estimators give. Figures **??** and 2.14 show the likelihood ratios LR_{fg} and LR_{gh}, for the PW and FPW methods. The likelihood ratios of the two FPW versions are almost the same and in both cases the variations in LR due to different experimental runs are negligible. The FPW methods outperform the classical PW estimation and show performance levels very similar to those of GMM, but with much less computational effort (as illustrated in Figure 2.11).

The number of components in FPW-H and FPW-S varied from 150 to 850 and 700 to 4000, respectively. Note that the number of components in PW estimates was $N \approx 30,000$.

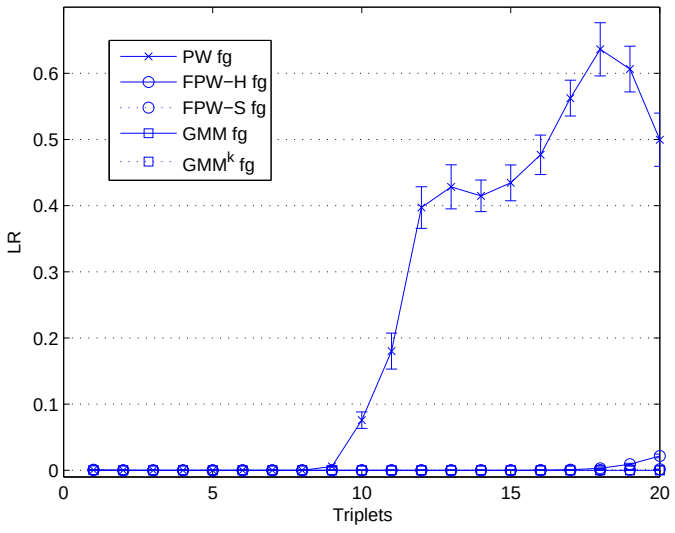


Figure 2.13: Likelihood Ratios (LR) of finding the correct source of the testing data between simulation sets f and g.

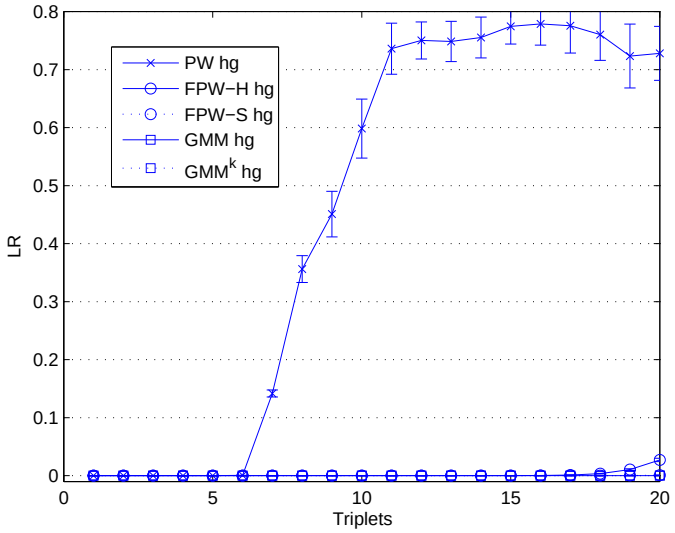


Figure 2.14: Likelihood Ratios (LR) of finding the correct source of the testing data between simulation sets g and h.

2.5 Summary

We presented an efficient and sparse probability density estimator manifold aligned dataset, termed Fast Parzen Windows (FPW). This algorithm partition the data space into a set of fixed radii hyper-discs first and then approximate the manifold aligned by a Gaussian with full covariance matrix in each hyper-disc. In this way, global manifold aligned density in the data space is approximated by a mixture of Gaussians. Since all densities are estimated locally and no global optimization is required in FPW, this density estimator could be viewed as a "sloppy Gaussian Mixture Model".

We presented two versions of FPW: FPW-H uses a hard-partitioning of the data samples, while FPW-S partitions the data samples in a soft manner. The performance of the FPW-S is analyzed theoretically and it shows that the performance of the FPW-S is dependent on the curvature of the true density. The former version is simpler and computationally more efficient than the latter one. Compared with the soft version, the hard version typically leads to only small performance degradation.

Compared with other sparse kernel based density estimation methods, FPW methods often showed comparable or superior performance at a much lower computational cost.

Chapter 3

Principled calibration of galaxy disruption simulations

In this Chapter, we demonstrate a real application of our efficient and sparse density estimator (Fast Parzen Windows) on the calibration of galaxy disruption simulations. First, we introduce the real astronomical problem of identifying the most plausible simulated disruption process against the real observation dataset. Then, we briefly review the coordinate systems adopted in simulation datasets and the observation datasets respectively, then clarify the relation between them. In Section 3, we describe our strategy of calibrating the galaxy disruption simulations against the observation. In Section 4, we verify our proposed calibration methodology on a series of pseudo-observation datasets generated from disruption simulations. The results show that our approach is capable of detecting the real source of the pseudo-observation datasets and it is reliable to the observation noise.

3.1 Astronomical problem statement

In the hierarchical growth of structure of the Universe, galaxies grow by merging. Most of the mergers in the lifetime of a giant galaxy involve smaller satellites, which get tidally disrupted and assimilated in the process. Deep observations of the nearby Universe have revealed that tidal streams, which constitute direct evidence of this process of assimilation, are ubiquitous [83]. Detailed studies of these tidal streams and debris can provide valuable insight into the detailed mechanism of galaxy formation and evolution.

Other than the satellite system of the Milky way, the most-studied system of streams involves the spiral galaxy M31 [84], where structures such as shells and streams of stars have been discovered in abundance in its vicinity. These exciting discoveries led the astronomers to investigate the possibility that such structures are in fact remnants of disrupted smaller satellite galaxies [81, 82]. A simulation model was designed to simulate the process of satellite galaxy disruption in the vicinity of a large galaxy [81]. It is a particle model, with particles representing stars. Each particle has six dimensions, three describing the spatial position, the other three describing the velocity along the spatial coordinates. The particle evolution is governed by the laws of physics. To track the evolution process starting from a particular initial condition, the astronomers collect the simulation datasets at successive evolution stages. Hence, the disruption process is captured in a series of simulation datasets.

In these simulation datasets, the observed low-dimensional structures evolve along with the satellite galaxy. With different initial conditions, the observed low-dimensional structures evolved in slightly different way. It makes that the low-dimensional structure in these simulation dataset looks very similar, but not identical. The ultimate goal of astronomers is to identify the most plausible set of initial conditions leading to the distribution of stars currently observed by the astronomers.

One possible way of learning the most plausible set of initial condition is to identify the simulated stars having the most similar distribution to that of the currently observed stars. It is impossible to compare the particles in simulation to the real observations on a point by point bases, but the observed density of stars can be compared with that of the simulated particles. In this chapter, we propose a strategy to measure the similarity between the simulated datasets and the observation dataset through its probability density distribution. First, we learn the probability density function (pdf) from the simulation datasets. Then, we adopt the pdf learnt in simulation space to the observation space. After that, we compute the (log-)likelihood of the observation dataset generated from the adopted pdf. Among a set of simulation datasets, the one having highest (log-) likelihood is believed as the one generating the observation dataset. Due to the huge amount of data in both simulation datasets and observation datasets, we employ Fast Parzen Window (FPW) algorithm proposed and verified in last chapter to learn the pdf efficiently. The proposed approach is verified by a series of simulation datasets and pseudo-observation datasets.

3.2 From astronomical simulation to observation

Due to the observation limitation, the observed galaxies are not in the same coordinate system as the simulated galaxies. In this section, we review their respective coordinate systems, and describe the relation between them. This relation is used to formulate the pdf for the observation datasets by projecting the pdf estimated from simulation dataset and generate the pseudo-observation datasets for the verification.

3.2.1 Simulation data

Given one particular initial conditional setting, the astronomical simulation model could generate a sequence of dataset $\{\mathcal{D}_i\}$ to describe the galaxy disruption process, where i is the index of disruption stages. Each simulation dataset $\mathcal{D}_i$ contains N particles describing the main galaxy (M31) and a satellite galaxy, written as $\mathcal{D}_i = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. The vector $\mathbf{x}_q = (x_1, x_2, x_3, v_1, v_2, v_3)^T$ and $\mathbf{x} = (x_1, x_2, x_3)^T$ represents the 3-D spatial position and $\mathbf{v} = (v_1, v_2, v_3)^T$ represents the corresponding velocity of the particle.

3.2.2 Observation data

We denote the observation dataset as $\mathcal{O}$, which contains N' observed stars $\mathcal{O}$=$\{\mathbf{y}_1, \ldots, \mathbf{y}_{N'}\}$, vector $\mathbf{y}_j$ represents the j-th observed information. In the real observation, the dimensions of the observed information vector $\mathbf{y}_j$ are different in different regions. We explain the observed $\mathbf{y}_j$ and its relation to the original (simulation) coordinates $\mathbf{x}_j$.

3.2.2.1 Observation space

To explain the composition of the observation dataset, we introduce the observation space first. Observation space is the space where the observed data (stars) stay. When we observe the galaxy from a fixed viewpoint outside the galaxy, the stars locating in the 3-D space are projected on the 2-D sky perpendicular to the line of sight.

We illustrate the observation space in Figure 3.1. Therefore, the 3-D spatial position is projected onto the 2-D observation plane and 3-D velocity is projected to the line-of-sight vector. Then, the full observation space of the galaxy is composed of 2-D position space and 1-D velocity information $\mathbf{y} = (y_1, y_2, u)$.

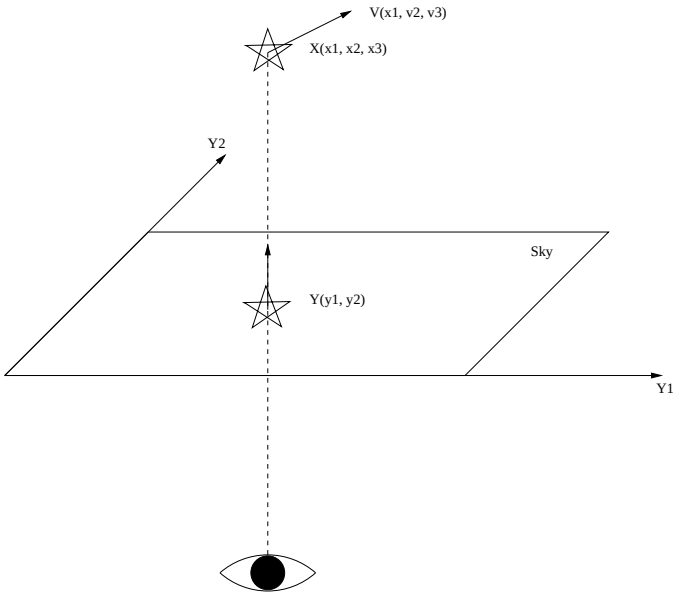


Figure 3.1: Observation space: the 2-D sky perpendicular to the line of sight is the observation plane. We use (y_1, y_2) to represent star's coordinates on the sky. The observable velocity is along the line-of-sight, which is the line connecting the observer and the star in the sky and is perpendicular to the sky.

The mapping from the full-phase space to the observation space could be represented by a linear projection matrix $\mathbf{M}_f$

$$\mathbf{M}_f = \begin{bmatrix} \mathbf{M}_s & 0 \\ 0 & \mathbf{M}_v \end{bmatrix} \tag{3.1}$$

where 3×2 matrix $\mathbf{M}_s$ represents the projection 3-D spatial space to the 2-D observed spatial space, and 3×1 matrix $\mathbf{M}_v$ transforms the 3-D velocity to the line-of-sight velocity. Since the line-of-sight velocity is perpendicular to the 2-D observation plane, we know that, $\mathbf{M}_v$=*cross*$(\mathbf{M}_s(:,1), \mathbf{M}_s(:,2))$, where *cross*$(,)$ represents the cross product and $\mathbf{M}_s(:,1)$, $\mathbf{M}_s(:,2)$ represent the first and second column of the matrix $\mathbf{M}_s$. $\mathbf{0}$ is zeros matrix. Then the observation $\mathbf{y}_j$ is

$$\begin{aligned} \mathbf{y}_j &= \mathbf{M}_f^T \mathbf{x}_j \\ &= \begin{pmatrix} \mathbf{M}_x^T \mathbf{x} \\ \mathbf{M}_v^T \mathbf{v} \end{pmatrix} \end{aligned} \tag{3.2}$$

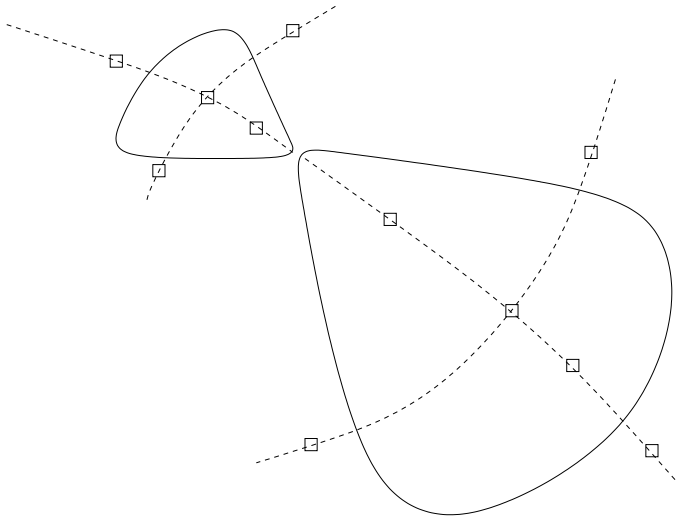


Figure 3.2: Observation fields: observation fields are chosen according to the spatial distribution of the galaxy. Closed contours represent the low dimensional structures observed and squares along with the dotted line are the chosen observational fields.

where $\mathbf{x}_j$ represent the simulation coordinate of the observation $\mathbf{y}_j$.

3.2.2.2 Observation fields

As we discussed above, in the observation space, we could obtain the 2-D position and line-of-sight velocity. The astronomers usually perform this observation within a few carefully selected observation fields. For the region outside the observation fields, the observation just have the 2-D position information. The principle of selecting the observation fields is illustrated in Figure 3.2. We use the closed contours to represent the embedded low-dimensional structures(stream and shells). The squares of the same size are the observation fields, which are chosen to go along and cross these low-dimensional structures. To do so, velocities of the stars in different area according to the low-dimensional structures are captured.

Therefore, the observation datasets $\mathcal{O}$ composes of two subsets $\mathcal{O}^3$ and $\mathcal{O}^2$, where $\mathcal{O}^3$ and $\mathcal{O}^2$ represent the observation taken in the observation fields and outside the observation respectively. The 3-D or 2-D observed $\mathbf{y}_j$ can be written as

$$\mathbf{y}_j = \begin{cases} \tilde{\mathbf{M}}_s^T \mathbf{x}_j, & \mathbf{y}_j \in \mathcal{O}^2 \\ \mathbf{M}_f^T \mathbf{x}_j, & \mathbf{y}_j \in \mathcal{O}^3 \end{cases} \tag{3.3}$$

where $\mathcal{O} = \mathcal{O}^3 \cup \mathcal{O}^2$ and $\tilde{\mathbf{M}}_s$ is the projection matrix from 6-D simulation space to the 2-D observation spatial space.

$$\tilde{\mathbf{M}}_s = \begin{bmatrix} \mathbf{M}_s \\ \mathbf{0} \end{bmatrix}, \tag{3.4}$$

3.3 Calibration process

In this section we introduce our strategy of identifying the most plausible simulation dataset given the observations. Among a set of L simulation datasets $\{\mathcal{D}^1, \mathcal{D}^2, \ldots, \mathcal{D}^L\}$, the procedure of identifying the most 'statistically similar' simulation set to the given observation dataset $\mathcal{O}$ includes the following three steps:

- **Step 1:** Use Fast Parzen Window (PFW) to estimate a probability density function (pdf) $\mathcal{M}_i$ on each simulation set $\mathcal{D}^i$, $i = 1, 2, ..., L$. Due to the scale of the simulation set, we use hard version of Fast Parzen Windows (FPW-H) in this chapter.

- **Step 2:** Adapt the pdfs $\mathcal{M}_i$ to observation (test) data:
  We project the $\mathcal{M}_i$'s onto the low-dimensional observation space (either 2-D or 3-D) to obtain the projected pdfs $\mathcal{M}_i^2$ and $\mathcal{M}_i^3$.

- **Step 3:** Compute the average (log-)likelihood (ALL) of the adapted densities on the observation dataset $\mathcal{O}$ and select the density that is maximally likely given $\mathcal{O}$. The simulation dataset that gave rise to the selected density is identified as the most 'statistically similar' to the observations $\mathcal{O}$.

Theoretically, we could select the most plausible simulation dataset by estimating density functions on the 2-D and 3-D observation spaces directly from the projected data at the price of increased computational cost and loss of flexibility of adopting a unified density model to possibly different projection spaces.

3.3.1 Learning the pdf

The simulation dataset generated from astronomical model includes the main galaxy (M31, 1 simulation dataset) and its satellite galaxy (L simulation datasets). We estimate

the pdfs of M31 and the satellite separately (for notational simplicity we omit the index i of the simulation dataset used to fit the satellite density):

$$\mathcal{M}_{31}(\mathbf{x}) = \sum_{j=1}^{M} P(\mathcal{S}_j)\mathcal{N}(\mathbf{x}; \mathbf{m}_j, C_j) \tag{3.5}$$

$$\mathcal{M}_{\text{sat}}(\mathbf{x}) = \sum_{j=1}^{M'} P(\mathcal{S}'_j)\mathcal{N}(\mathbf{x}; \mathbf{m}'_j, C'_j) \tag{3.6}$$

where M and M' denote the number of components in $\mathcal{M}_{31}$ and $\mathcal{M}_{\text{sat}}$ respectively; $\mathcal{S}_j$ and $\mathcal{S}'_j$ are the partition sets; $P(\mathcal{S}_j)$ and $P(\mathcal{S}'_j)$ denote the mixing weights of the Gaussian components with means $\mathbf{m}_j$ ($\mathbf{m}'_j$) and covariance matrices C_j (C'_j) .

Probabilistic model of the simulation data is a mixture

$$\mathcal{M}(\mathbf{x}) = P_{31}\mathcal{M}_{31}(\mathbf{x}) + P_{\text{sat}}\mathcal{M}_{\text{sat}}(\mathbf{x}) \tag{3.7}$$

where P_{31} and P_{sat} are the mixing co-efficients estimated from the mass ratio of M31 and its satellite. In our experiment, $P_{31} = 0.9688$ and $P_{sat} = 0.0312$.

3.3.2 Adapting pdfs to observations

Since the observation space is different from the simulation space, we need to transform the pdf $\mathcal{M}$ learnt in simulation space to the observation space.

Given a D-dimensional dataset $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, its (orthogonal) projection onto a d-dimensional linear subspace ($d < D$) via a projection matrix $\mathbf{M}$ is $\mathcal{D}' = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, $\mathbf{y}_i = \mathbf{M}^T\mathbf{x}_i$. The model $\mathcal{M}$ projected onto that subspace will be denoted by $\mathcal{M}^d$. We have

$$\mathcal{M}^d(\mathbf{y}) = P_{31}\mathcal{M}^d_{31}(\mathbf{y}) + P_{\text{sat}}\mathcal{M}^d_{\text{sat}}(\mathbf{y}) \tag{3.8}$$

where $\mathcal{M}^d_{31}$ and $\mathcal{M}^d_{\text{sat}}$ are the projected densities $\mathcal{M}_{31}$ and $\mathcal{M}_{\text{sat}}$, respectively. In observation space

$$\mathcal{M}^d_{31}(\mathbf{y}) = \sum_{j=1}^{M} P(\mathcal{S}_j)\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}_j, \boldsymbol{\Psi}_j) \tag{3.9}$$

$$\mathcal{M}^d_{\text{sat}}(\mathbf{y}) = \sum_{j=1}^{M'} P(\mathcal{S}'_j)\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}'_j, \boldsymbol{\Psi}'_j) \tag{3.10}$$

By employing the parameter estimation equations of FPW (Eqs. (2.9, 2.10)), we get the parameters of the projected densities as follows:

$$\begin{aligned}\boldsymbol{\mu}_j =& \frac{1}{|\mathcal{S}_j|}\sum_{i\in\mathcal{S}_j}\mathbf{y}_i = \frac{1}{|\mathcal{S}_j|}\sum_{i\in\mathcal{S}_j}\mathbf{M}^T\mathbf{x}_i \\ =& \mathbf{M}^T\frac{1}{|\mathcal{S}_j|}\sum_{i\in\mathcal{S}_j}\mathbf{x}_i = \mathbf{M}^T\mathbf{m}_j\end{aligned} \tag{3.11}$$

$$\begin{aligned}\boldsymbol{\Psi}_j =& \frac{1}{|\mathcal{S}_j|}\sum_{i\in\mathcal{S}_j}(\mathbf{y}_i-\boldsymbol{\mu}_j)(\mathbf{y}_i-\boldsymbol{\mu}_j)^T \\ =& \frac{1}{|\mathcal{S}_j|}\sum_{i\in\mathcal{S}_j}(\mathbf{M}^T\mathbf{x}_i-\mathbf{M}^T\mathbf{m}_j)(\mathbf{M}^T\mathbf{x}_i-\mathbf{M}^T\mathbf{m}_j)^T \\ =& \mathbf{M}^T\sum_{i\in\mathcal{S}_j}\frac{1}{|\mathcal{S}_j|}(\mathbf{x}_i-\mathbf{m}_j)(\mathbf{x}_i-\mathbf{m}_j)^T\mathbf{M} \\ =& \mathbf{M}^T C_j\,\mathbf{M}\end{aligned} \tag{3.12}$$

Analogously,

$$\begin{aligned}\boldsymbol{\mu}'_j =& \mathbf{M}^T\,\mathbf{m}'_j \\ \boldsymbol{\Psi}'_j =& \mathbf{M}^T C'_j\,\mathbf{M}.\end{aligned} \tag{3.13}$$

Projected models on the 3-D and 2-D observational spaces can be obtained by employing the projection matrices $\mathbf{M}_f$ (3.1) and $\tilde{\mathbf{M}}_s$ (3.4), respectively:

$$\mathcal{M}^2(\mathbf{y}) = P_{31}\mathcal{M}^2_{31}(\mathbf{y}) + P_{\text{sat}}\mathcal{M}^2_{\text{sat}}(\mathbf{y}), \tag{3.14}$$

$$\mathcal{M}^3(\mathbf{y}) = P_{31}\mathcal{M}^3_{31}(\mathbf{y}) + P_{\text{sat}}\mathcal{M}^3_{\text{sat}}(\mathbf{y}). \tag{3.15}$$

3.3.3 Computing the (log-)likelihood

For every simulation dataset $\mathcal{D}^i$, we construct two density functions $\mathcal{M}^{i,2}$, $\mathcal{M}^{i,3}$ as described above. The (log-)likelihood given the observation data is given by:

$$\mathcal{L}_i(\mathcal{O}) = \frac{1}{|\mathcal{O}|}\sum_{d=2}^{3}\sum_{\mathbf{y}\in\mathcal{O}^d}\log\mathcal{M}^{i,d}(\mathbf{y}). \tag{3.16}$$

where $|\mathcal{O}|$ is the size of $\mathcal{O}$.

For a given set of simulation datasets $\{\mathcal{D}^1, \mathcal{D}^2, \ldots, \mathcal{D}^L\}$, the index of the most likely

(under our model structure) disruption process is then

$$i = \arg \max_{i=1,2,\ldots,L} \mathcal{L}_i(\mathcal{O}). \tag{3.17}$$

3.4 Experiments and evaluation

In this section, we verify our proposed identification methodology on a set of simulation datasets and their related pseudo observation datasets.

3.4.1 Data description

The astronomical simulation datasets used to investigate the reliability of the proposed identification procedure were generated from one astronomical simulation model. In this model, the main galaxy (M31) is represented by 5,089,594 particles and another 163,840 particles are used to describe the satellite galaxy (M32).

Given 12 different initial conditions, the astronomical model create 12 different galaxies' disruption process. We denote them as $\mathcal{G}_i$, $(i = 1, \ldots, 12)$. In each process, 11 successive simulated particle datasets were recorded to describe the satellite galaxy's disruption process from stage 15 to stage 25. In these simulated particle datasets, the particles for the main galaxy (galaxy M31) are remaining the same during the evolution in the astronomical simulation model.

In the following experiments, we choose groups 1, 5 and 9 $\{\mathcal{G}_1, \mathcal{G}_5, \mathcal{G}_9\}$ among these 12 disruption groups to verify our proposed strategy. And we name these disruption process as F, G and H respectively.

3.4.2 Building model for simulation datasets

The first step of this experiment is to build density model $\mathcal{M}$ from the simulation datasets, which includes $\mathcal{M}_{31}$ and $\mathcal{M}_{\text{sat}}$. We use Fast Parzen Windows (FPW) to learn the models from 90% of the simulation data of galaxy M31 and satellite M32 seperately (training data). We denote the satelite density fitted at stage j of the disruption process g by $\mathcal{M}_{\text{sat}}^{g,j}$, $g \in \{F, G, H\}$, $j = 15, 16, \ldots, 25$. We then test performance of the overall density model

$$\mathcal{M}^{g,j}(\mathbf{x}) = P_{31}\mathcal{M}_{31}(\mathbf{x}) + P_{\text{sat}}\mathcal{M}_{\text{sat}}^{g,j}(\mathbf{x}) \tag{3.18}$$

on pseudo-observations generated from the remaining 10% of the disruption process data, $g \in \{F, G, H\}$, at the same simulation stage j, or at related stages $j-1$, j, $j+1$. The FPW parameters are set by 10-fold cross-validation on the training data.

3.4.3 Constructing the observation spaces

Spatial simulation of M31 operates within a co-ordinate system (X_G, Y_G, Z_G). The disk lies in the $(X_G,\ Y_G)$ space and Z_G points along the spin axis Denote the basis transformation matrix from this system onto our observation space (X_S, Y_S, Z_S) by $\mathbf{M}_s^r$. In the sky, east is to the left and west is to the right ad so in our coordinate system X_S points to the east (left), Y_S points to the north (up) and Z_S points away from us. The basis transformation involves two rotations. The first one is related to inclination that rotates the galaxy by angle $i = -77^o$ around the Y_S axis. Then we rotate the galaxies by angle $\phi = 37^o$ around Z_S axis. The rotations are represented by orthogonal matrices

$$\mathbf{M}_{inl} = \begin{pmatrix} \cos i & 0 & \sin i \\ 0 & 1 & 0 \\ -\sin i & 0 & \cos i \end{pmatrix} \tag{3.19}$$

and

$$\mathbf{M}_{pos} = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.20}$$

The product of these rotation matrices gives the transformation from simulation coordinate to the sky coordinate:

$$\mathbf{M}_{trans} = \begin{pmatrix} \cos\phi\cos i & \sin\phi & \cos\phi\sin i \\ -\sin\phi\cos i & \cos\phi & -\sin\phi\sin i \\ -\sin i & 0 & \cos i \end{pmatrix}. \tag{3.21}$$

Since on the sky the coordinate X_S points to the left, we have

$$\mathbf{M}_s^r = \begin{pmatrix} -\cos\phi\cos i & -\sin\phi & -\cos\phi\sin i \\ -\sin\phi\cos i & \cos\phi & -\sin\phi\sin i \end{pmatrix}^T. \tag{3.22}$$

Besides the realistic orientation of the observation space, we construct two other synthetic observation spaces differing in the extent to which the spatial structure in

the simulation space is preserved. To that end we determine 3 principle components $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$ of the galaxy's spatial distribution in the simulation space and construct the projection matrices as follows:

$$\begin{aligned} \mathbf{M}_s^1 &= (\mathbf{p}_1, \mathbf{p}_2), \quad \mathbf{M}_v^1 = (\mathbf{p}_3) \\ \mathbf{M}_s^2 &= (\mathbf{p}_2, \mathbf{p}_3), \quad \mathbf{M}_v^2 = (\mathbf{p}_1). \end{aligned} \tag{3.23}$$

In Figure 3.3, we show the 20-th stage of galaxies' disruption distribution on different projection spaces. Plots in the second column are the projection of disruption process F, the third column is the projection of disruption process G and the last column is the projection of disruption process H. The first column illustrates the psuedo-observation set (5-$\mathcal{PO}$ construction detailed later) and the whole spatial space is represented by 200×200 bins (the same setting is also used later for classical chi-square test), and the brightness of the bins indicates the logarithm of stars' amount in the bins.

3.4.4 Constructing pseudo-observation datasets

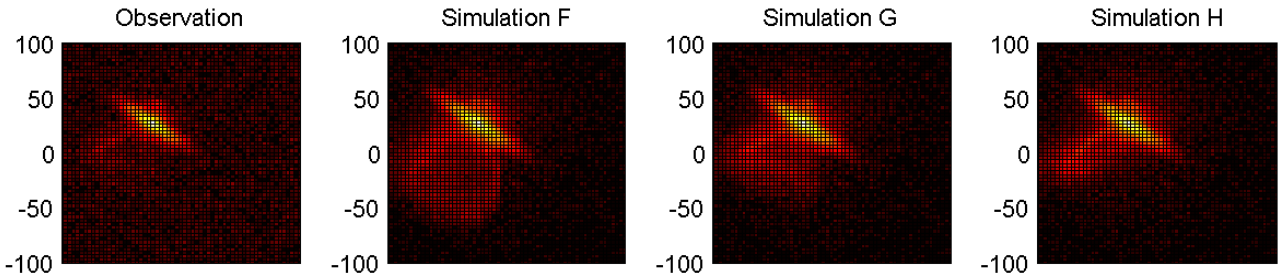
Given an observation space, we generate a series of pseudo-observations to test the reliability of our proposed methodology. The pseudo-observation datasets are constructed from the 10% hold-out simulation datasets at each stage of the disruption process G. For each projection space defined by the projection operator $\mathbf{M}_s$ ($\mathbf{M}_s \in \{\mathbf{M}_s^r, \mathbf{M}_s^1, \mathbf{M}_s^2\}$, we construct a series of pseudo-observation data sets with increasing levels complexity.

3.4.4.1 Pseudo-observation 1 (1-$\mathcal{PO}$)

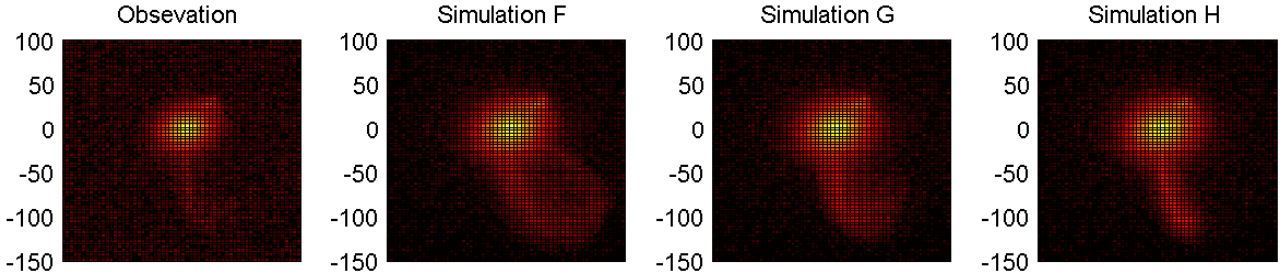
Pseudo-observation set 1 (1-$\mathcal{PO}$) only contains 2-D positional information in the observation space. 1-$\mathcal{PO} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$,

$$\mathbf{y}_i = \tilde{\mathbf{M}}_s^T \mathbf{x}_i; \tag{3.24}$$

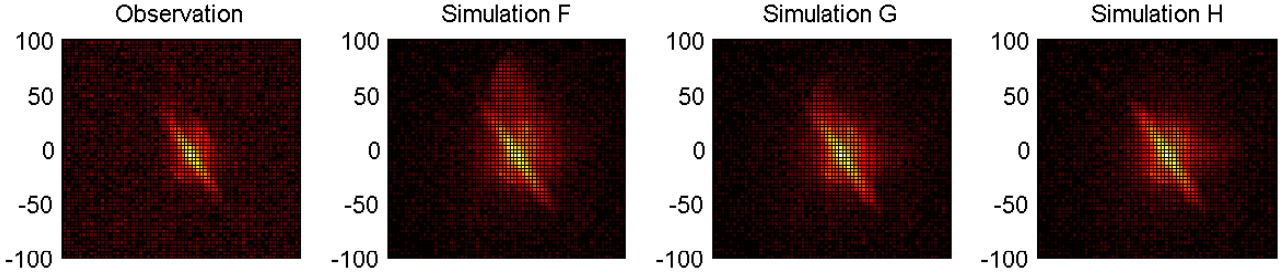
where $\tilde{\mathbf{M}}_s$ is defined in Eq. (3.4).



(a) Illustration of pseudo-observation (5-$\mathcal{PO}$) and simulation F, G and H on real observation space ($\mathbf{M}_s^r$)



(b) Illustration of pseudo-observation (5-$\mathcal{PO}$) and simulation F, G and H on real observation space ($\mathbf{M}_s^1$)



(c) Illustration of pseudo-observation (5-$\mathcal{PO}$) and simulation F, G and H on real observation space ($\mathbf{M}_s^2$)

Figure 3.3: Illustration of pseudo-observation (5-$\mathcal{PO}$, described later) and three simulation datasets (at disruption stage 20 of disruption process F, G and H) on the real observation space ($\mathbf{M}_s^r$), and two synthetic observation spaces ($\mathbf{M}_s^1$ and $\mathbf{M}_s^2$) respectively.

3.4.4.2 Pseudo-observation 2 (2-$\mathcal{PO}$)

Pseudo-observation set 2 ,2-$\mathcal{PO}$, takes into an account line-of sight velocities measured for stars within the observational fields. For stars outside the observation fields (2-$\mathcal{PO}^2$) we only have 2-D spatial information (as in 1-$\mathcal{PO}$), for those inside the observation fields (2-$\mathcal{PO}^3$) we have a 3-D spatial and velocity information.

$$\mathbf{y}_i = \begin{cases} \tilde{\mathbf{M}}_s^T \mathbf{x}_i, & \mathbf{y}_i \in \text{2-}\mathcal{PO}^2 \\ \mathbf{M}_f^T \mathbf{x}_i, & \mathbf{y}_i \in \text{2-}\mathcal{PO}^3 \end{cases} \tag{3.25}$$

We selected the observation fields manually based on the current observational practice. As an example, consider the stage 20 of the disruption process G. The chosen observation fields for every projection space are shown in Figure 3.4. Only the satellite galaxy is plotted, observation fields are the white squares on the 2-D observed data. Observation spaces 1, 2, 3 correspond to projection spaces represented by $\mathbf{M}_s^r$, $\mathbf{M}_s^1$ and $\mathbf{M}_s^2$, respectively.

3.4.4.3 Pseudo-observation 3 (3-$\mathcal{PO}$)

In the third Pseudo-observation set 3-$\mathcal{PO}$, in addition to observational fields we introduce a realistic additive observational noise in the line-of-sight velocity.

$$\mathbf{y}_i = \begin{cases} \tilde{\mathbf{M}}_s^T \mathbf{x}_i, & \mathbf{y}_i \in \text{3-}\mathcal{PO}^2 \\ \mathbf{M}_f^T \mathbf{x}_i + \epsilon_{\mathbf{y}_i}, & \mathbf{y}_i \in \text{3-}\mathcal{PO}^3 \end{cases} \tag{3.26}$$

where $\epsilon_{\mathbf{y}_i} \sim \mathcal{N}(\mathbf{0}, \psi)$ with the covariance

$$\psi = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{s}{\nu} \end{bmatrix}$$

and $s = 10, 20, 50$ km/s and $\nu = 656$ is the velocity scale in the simulation dataset.

3.4.4.4 Pseudo-observation 4 (4-$\mathcal{PO}$)

Pseudo-observation set 4 (4-$\mathcal{PO}$) (besides features captured by the previous sets) reflects the fact that compared with the original (simulated) distribution of M31, the observed density is actually slightly periodically varying with respect to the distance from the galaxy's center. We took account of this spatial feature as follows:

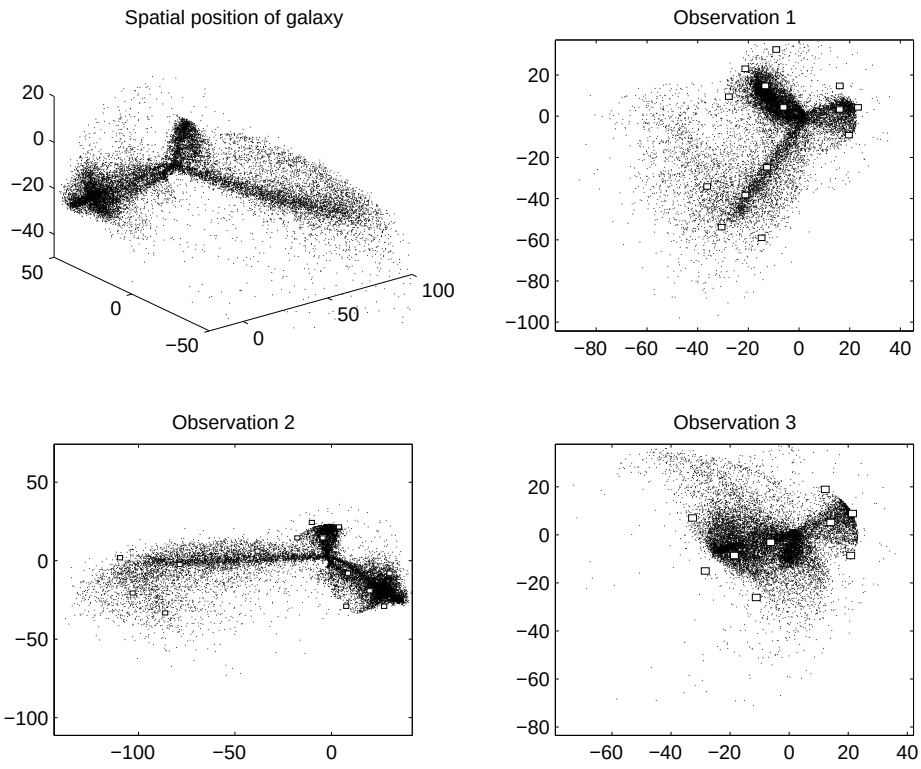


Figure 3.4: Observational fields chosen for the simulation at disruption stage 20 of disruption process G. The first plot shows the spatial distribution of the simulation data, the following 3 plots illustrate the chosen observation fields and the simulation data in coordinate system $x_1 - x_2$, $x_2 - x_3$ and $x_1 - x_3$ respectively.

- We estimated the density $\mathcal{F}(r)$ of M31 as a function of the distance from the galaxy center r. The density is shown in Figure 3.5 (a).
- We then use a cosine function shown in 3.5(b) to modulate the density $\mathcal{F}(r)$ with periodic features. Half period of the cosine function is equal to the width of the stream in its middle section (set to 5). Changes $\tau(r)$ to the original density read

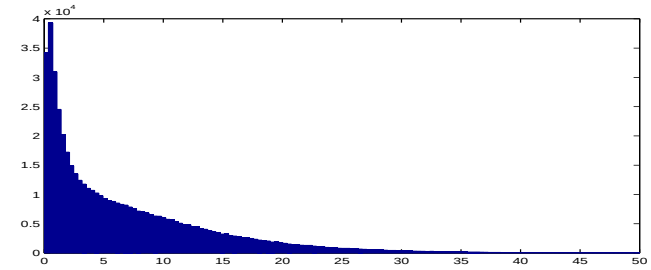
$$\tau(r) = \cos(r)\ \mathcal{F}(r)\ \delta, \tag{3.27}$$

  where $\delta = 3\%$ (see Figure 3.5 (c)).
- Finally, we adjust the distribution of M31 galaxy according to $\tau(r)$ and generate the observations by down proportionally sampling the original observations in regions with negative $\tau(r)$. and upsampling (according to local densities) in regions of positive $\tau(r)$.

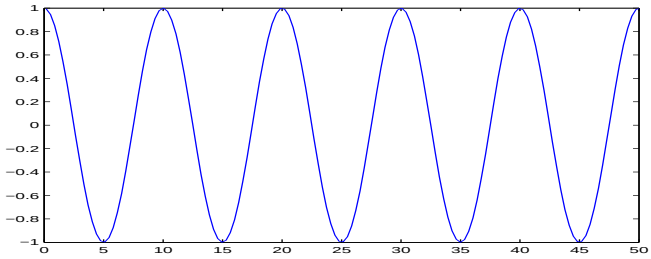
3.4.5 Experiments and Results

To test the reliability of our proposed methodology, we form 11 triplets (F_i, G_i, H_i) of simulation data sets corresponding to disruption simulation stages $i = \{15, \ldots, 25\}$ by rolling a 3×1 window over the three disruption processes F, G and H from stage 15 to 25. Each such triplet contains simulation datasets generated at same time stage but with different initial conditions.

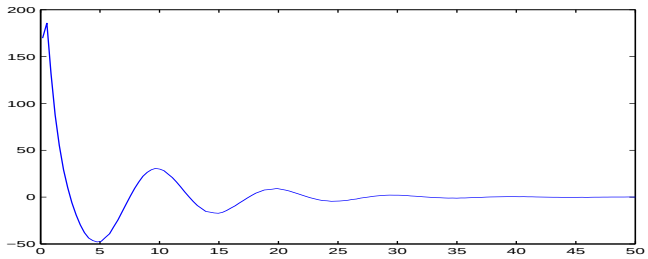
For each simulation stage i and projection operator $\mathbf{M}_s$ we test the capability of our method to detect the correct initial condition (in other words to detect the correct generative source of the pseudo-observations among the three simulation datasets in (F_i, G_i, H_i)). Using the 10% hold-out dataset from G_i, we construct 8 test sets of pseudo-observation data $\mathcal{PO} \in \{1\text{-}\mathcal{PO}, 2\text{-}\mathcal{PO}, 3\text{-}\mathcal{PO}, 4\text{-}\mathcal{PO}\}$ with parameters shown in Table 3.1. We repeated the experiments 10 times, each time with a different 10% hold-out set for pseudo-observation construction and the number of correct detections out of the 10 runs was recorded. Reassuringly, for all simulation stages $i = \{15, \ldots, 25\}$ and all projection operators $\mathbf{M}_s^r$, $\mathbf{M}_s^1$ and $\mathbf{M}_s^2$, the true source G was detected with 100% accuracy (correct detection in each of the 10 runs).



(a) Histogram



(b) cos function



(c) variable numbers

Figure 3.5: Simulated observational features of fields in the vicinity of M31.

Table 3.1: Parameter settings for the pseudo-observation datasets.

$\mathcal{PO}$	Parameters
1-$\mathcal{PO}$	$\mathbf{M}_s$
2-$\mathcal{PO}$	$\mathbf{M}_s$, $\mathbf{M}_f$
3-$\mathcal{PO}$	$\mathbf{M}_s$, $\mathbf{M}_f$, s=10km/s
3-$\mathcal{PO}$	$\mathbf{M}_s$, $\mathbf{M}_f$, s=20km/s
3-$\mathcal{PO}$	$\mathbf{M}_s$, $\mathbf{M}_f$, s=50km/s
4-$\mathcal{PO}$	$\mathbf{M}_s$, $\mathbf{M}_f$, s=10km/s, δ=3%
4-$\mathcal{PO}$	$\mathbf{M}_s$, $\mathbf{M}_f$, s=20km/s, δ=3%
4-$\mathcal{PO}$	$\mathbf{M}_s$, $\mathbf{M}_f$, s=50km/s, δ=3%

3.4.6 Background Screen

In the real observation, there are not only the galaxies but also the background and foreground noises coming from foreground Milky Way stars and unresolved background galaxies.Here we call it screen noise. In our last pseudo-observation datasets (5-$\mathcal{PO}$), we add the screen noise ($\mathcal{SN}$) to the observed galaxies. Therefore, we have

$$5\text{-}\mathcal{PO} = 4\text{-}\mathcal{PO} \cup \mathcal{SN}. \tag{3.28}$$

If we know the distribution of screen noise $\mathcal{SN}$ on 2-D and 3-D observation space (represented by $\mathcal{S}^2$ and $\mathcal{S}^3$ respectively), we could describe the distribution of 5-$\mathcal{PO}$ on 3-D and 2-D observation space by extending our galaxies model $\mathcal{M}^2$, $\mathcal{M}^3$ in 2-D and 3-D observation spaces to mixture model of galaxies and screen noise $\mathcal{M}_s^2$ and $\mathcal{M}_s^3$ respectively as follows:

$$\begin{aligned}
\mathcal{M}_s^2(\mathbf{y}) &= \alpha_m^2 \mathcal{M}^2(\mathbf{y}) + \alpha_s^2 \mathcal{S}^2(\mathbf{y}) \\
&= \alpha_m^2 P_{31} \mathcal{M}_{31}^2(\mathbf{y}) + \alpha_m^2 P_{\text{sat}} \mathcal{M}_{\text{sat}}^2(\mathbf{y}) + \alpha_s^2 \mathcal{S}^2(\mathbf{y}) \qquad (3.29) \\
\mathcal{M}_s^3(\mathbf{y}) &= \alpha_m^3 \mathcal{M}^3(\mathbf{y}) + \alpha_s^3 \mathcal{S}^3(\mathbf{y}) \\
&= \alpha_m^3 P_{31} \mathcal{M}_{31}^3(\mathbf{y}) + \alpha_m^3 P_{\text{sat}} \mathcal{M}_{\text{sat}}^2(\mathbf{y}) + \alpha_s^3 \mathcal{S}^3(\mathbf{y}) \qquad (3.30)
\end{aligned}$$

where α_m^2 and α_s^2 are the mixing coefficients of galaxies' model and the screen noise on 2-D observation space, α_m^3 and α_s^3 are the mixing coefficients of galaxies' model and the screen noise on 3-D observation space.

Similar to last subsection, for one triplet (F_i, G_i, H_i) and one given projection space (denoted by $\mathbf{M}_s$), the experiments of testing the capability of detecting the correct initial condition are carried out as follows:

- In order to learn the model triplet ($\{\mathcal{M}_s^2, \mathcal{M}_s^3\}_{Fi}$, $\{\mathcal{M}_s^2, \mathcal{M}_s^3\}_{Gi}$, $\{\mathcal{M}_s^2, \mathcal{M}_s^3\}_{Hi}$) on the given projection space, we make the following settings for Eq. (3.29) and Eq. (3.30): first, we set $\alpha_m^2 = 0.7$ and $\alpha_s^2 = 0.3$ as the proportion between screen and the galaxy M31 and M32 is 30:70 in 2-D observation space and the distribution of $\mathcal{S}^2$ to be uniform distribution. Then, because we donot consider the influence caused by the line-of-velocity of the screen noise at the current stage, we set $\alpha_m^3 = 1$ and $\alpha_s^3 = 0$ in this particular experiment.

- The testing data 5-$\mathcal{PO}$ is also constructed in the following way.

$$\begin{aligned} \text{5-}\mathcal{PO}^2 &= \text{4-}\mathcal{PO}^2 \cup \mathcal{SN}^2. \\ \text{5-}\mathcal{PO}^3 &= \text{4-}\mathcal{PO}^3 \end{aligned} \tag{3.31}$$

  where $|\mathcal{SN}^2|=\frac{3}{7}|\text{4-}\mathcal{PO}^2|$. And the data in $\mathcal{SN}^2$ are uniformly distributed on the observation space.

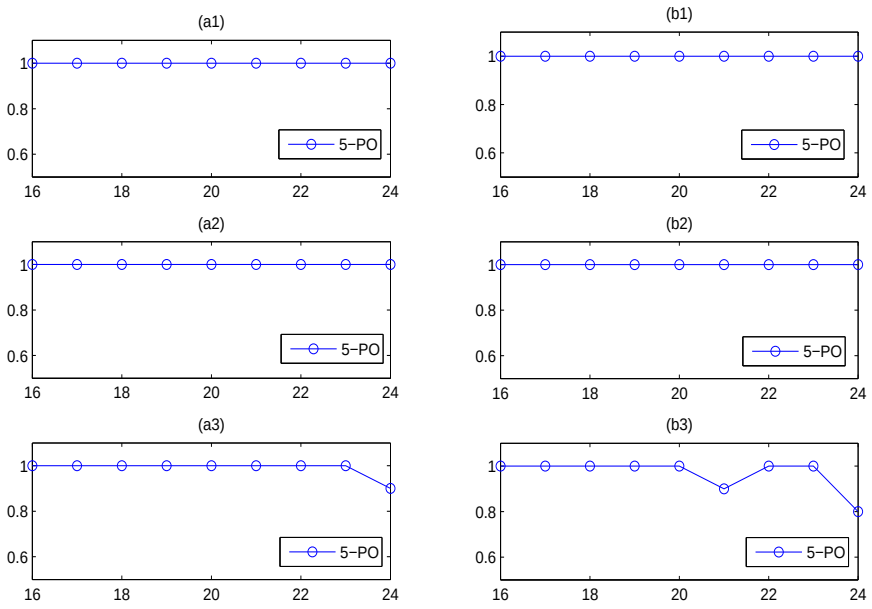
- The average log likelihood is computed as

$$\ln \mathcal{L}(\mathcal{O}; \{\mathcal{M}_s^2, \mathcal{M}_s^3\}_{gj}) = \frac{1}{|\mathcal{O}|}(\sum_{d=2}^{3} \sum_{\mathbf{y}_j \in \mathcal{O}^d} \log \mathcal{M}_s^d(\mathbf{y}_j) \tag{3.32}$$

  where g indicates the disruption process is the triplet and i is the index of the stage. So the ALL of the triplet is calculated as Eq. (3.33).

$$\begin{aligned} L_F &= \ln \mathcal{L}(\text{5-}\mathcal{PO}, \{\mathcal{M}_s^2, \mathcal{M}_s^3\}_{F_i}) \\ L_G &= \ln \mathcal{L}(\text{5-}\mathcal{PO}, \{\mathcal{M}_s^2, \mathcal{M}_s^3\}_{G_i}) \\ L_H &= \ln \mathcal{L}(\text{5-}\mathcal{PO}, \{\mathcal{M}_s^2, \mathcal{M}_s^3\}_{H_i}) \end{aligned} \tag{3.33}$$

Again, to show the correct rate, we plot the correct rate out of the 10 runs in Figure 3.6 (a). Subplots (a1), (a2) and (a3) show the correct rate on projection space $\mathbf{M}_s^r$,



(a) Triplets (b) 3×3 sets

Figure 3.6: Correct rates of the proposed principled calibration on detecting the source of the pseudo-observation sets 5-$\mathcal{PO}$ from triplets and 3×3 sets.

$\mathbf{M}_s^1$ and $\mathbf{M}_s^2$ respectively. x-axis labels the stages of the triplet from 16 to 24 and y-axis is the correct rate.

Finally, we extend our experiments from detecting the correct initial condition of the observation dataset to detecting both the initial condition and the stage of the observation datasets. Therefore, we extend the triple (F_i, G_i, H_i) into 3×3 set $\begin{pmatrix} F_{t-1} & F_t & F_{t+1} \\ G_{t-1} & G_t & G_{t+1} \\ H_{t-1} & H_t & H_{t+1} \end{pmatrix}$ where $t = \{16, \ldots, 24\}$.

Using the same pseudo-observation (5-$\mathcal{PO}$), for one 3×3 set and one given projection space, we could obtain a set of ALL learnt by the models of the simulation datasets in the 3×3 set. We represent them in a set as $\begin{pmatrix} L_{F_{t-1}} & L_{F_t} & L_{F_{t+1}} \\ L_{G_{t-1}} & L_{G_t} & L_{G_{t+1}} \\ L_{H_{t-1}} & L_{H_t} & L_{H_{t+1}} \end{pmatrix}$. To plot the correct rate, we also compute the likelihood ratio as in Eq. (3.34). Then we obtain Figure 3.6 (b). Subplots (b1), (b2) and (b3) show the correct rate on projection space

$\mathbf{M}_s^r$, $\mathbf{M}_s^1$ and $\mathbf{M}_s^2$ respectively.

$$LR_{ex} = \frac{\max \begin{pmatrix} \mathcal{L}_{F_{t-1}}, & \mathcal{L}_{F_t}, & \mathcal{L}_{F_{t+1}} \\ \mathcal{L}_{G_{t-1}}, & \emptyset, & \mathcal{L}_{G_{t+1}} \\ \mathcal{L}_{H_{t-1}}, & \mathcal{L}_{H_t}, & \mathcal{L}_{H_{t+1}} \end{pmatrix}}{\mathcal{L}_{G_t}} \tag{3.34}$$

3.4.7 Chi-square test for simulation datasets

In astronomical research, the classical chi-square test [85] is usually used for the calibration of the simulations. In this subsection, we briefly describe the procedure of this method and compare its performance with our strategy detailed earlier. To carry out the chi-square test, we first define an equally spaced 200×200 grid to segment the overall observation plane. The segmented planes of both real observation space ($\mathbf{M}_s^r$) and the synthetic observation spaces ($\mathbf{M}_s^1$, $\mathbf{M}_s^2$) are illustrated in the first column of Figure 3.3. The brightness of each cell in the planes represents the logarithm of the number of points in it. Using the same grid, we also plot the segmentation of simulation dataset from disruption process F, G and H on each observation spaces in the same figure.

For each triplet and 3×3 sets constructed earlier, we perform the chi-square test against the pseudo-observation 5-$\mathcal{PO}$. Since the pseudo observation (5-$\mathcal{PO}$) consists of 2-D position information and velocity information in the observation fields, we calculate the chi-square test by including two parts:

- chi-square test on position

  For the each simulation dataset in a triplet or a 3×3 set, we calculate the spatial test as follows:

  $$E_s^q = \sum_{i,j} \frac{(C_{i,j}^{\mathcal{O}} - C_{ij}^q)^2}{C_{i,j}^q} \tag{3.35}$$

  where C_{ij}^q is the counts of particle in cell (i, j) of simulation set indexed by q in the triplet or 3×3 set. $C_{i,j}^{\mathcal{O}}$ is the number of particles in cell (i, j) of observation dataset 5-$\mathcal{PO}$. Note that, if no particle observation in cell (i, j), $C_{i,j}^q = 0$ will lead to the problem of division by zero in Eq. (3.35). In order to avoid the zero-denominator, we add a small number 1 to each $C_{i,j}$. Therefore, the chi-square

test on position calculated by Eq. (3.35) is modified as follows:

$$\tilde{E}_s^q = \sum_{i,j} \frac{(C_{ij}^q - C_{i,j}^{\mathcal{O}})^2}{C_{i,j}^q + 1} \tag{3.36}$$

- chi-square test on velocity

  The velocity chi-square test between the observed velocity and simulated velocity in q-th set of the triplet/3×3 set is computed in the following way:

$$E_v^q = \sum_{i,j} \sum_{v \in R^{\mathcal{O}}} \frac{(v - \tilde{v}_q)^2}{\delta_{\tilde{v}_q}^2}, \tag{3.37}$$

  where v denotes the velocity observed in cell (i, j) of the observation 5-$\mathcal{PO}$ and $R^{\mathcal{O}}$ represents the collection of the velocities in this cell. Let $\mathbf{y}$ be the position corresponding to the velocity v, $\tilde{v}_q$ (with the associated variance $\delta_{\tilde{v}_q}^2$) is defined as the expected velocity at the position $\mathbf{y}$ in the q-th simulation set:

$$\tilde{v}_q = \frac{1}{\sum_{u \in T} K_h(\mathbf{y}_u - \mathbf{y})} \sum_{u \in T} K_h(\mathbf{y}_u - \mathbf{y}) v_u \tag{3.38}$$

$$\delta_{\tilde{v}_q}^2 = \frac{1}{\sum_{u \in T} K_h(\mathbf{y}_u - \mathbf{y})} \sum_{u \in T} K_h(\mathbf{y}_u - \mathbf{y})(v_u - \tilde{v}_q)^2 \tag{3.39}$$

  where T is the collection of the particle's neighbors at the position $\mathbf{y}$ in the q-th simulation set. To select its neighbors, we associated a weight kernel with the particle first. The width of the kernel h is $d/4$ (d is the length of the grid). Then the neighbors are the particles $\mathbf{y}_u$ that $K_h(\mathbf{y}_u - \mathbf{y}) > 0.01$.

The overall test value is the sum of the spatial test value and the velocity test value

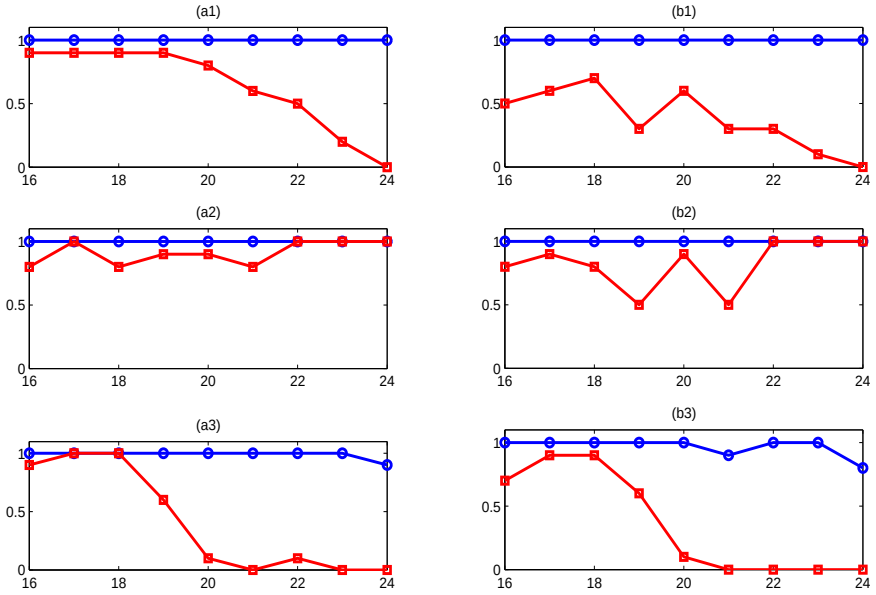
$$E^q = \tilde{E}_s^q + E_v^q.$$

Among a group of simulation dataset, the detected source of observation is given by:

$$q = \arg\min_q E^q. \tag{3.40}$$

In Figure 3.7, we illustrate the correct rates given by the chi-square test and our proposed principled calibration. Subplots (a1), (a2) and (a3) report correct rate of the 9 triplets on three different observation spaces respectively. Subplots (b1), (b2) and (b3)

report the correct rates of 3×3 sets on three different observation spaces respectively. It is clear that the performance of our proposed strategy always dominates over that of chi-square test.



(a) Triplets (b) 3×3 sets

Figure 3.7: Correct rates of the chi-square test and the proposed principled calibration on detecting the source of the pseudo-observation sets 5-$\mathcal{PO}$ from triplets and 3×3 sets. The blue circles are the correct rates of the principled calibration and the red squares are the correct rates of the chi-square test.

3.5 Summary

In the chapter, we are facing a real problem in astronomy field. To study the evolution of main galaxy M31's satellite galaxy M32, researchers build a particle model to simulate the evolution process. With different initial conditions, the model produces several different simulated processes. Among these similar simulations, it is difficult to identify the most plausible simulation against the real observation automatically.

Instead of comparing the simulation with the observation partical by partical, we originally proposed in this chapter that employing the probability density estimation

to describe the particles distribution. Due to the very huge amount of points in both the simulation datasets and the observation datasets, most probability density estimators are computationally too expensive to work, like Parzen windows, Gaussian Mixture model and so on.

We apply our effective and efficient density estimation Fast Parzen Window to the large-scale datasets. To investigate the performance our proposed likelihood based principled calibration method, we use a series of pseudo observation datasets, from very simple one to complex one approximating to the real observation. The experiment results demonstrate that the likelihood based principled calibration is much more reliable than the classical chi-square test based calibration.

Chapter 4

Manifold aligned density estimation through explicit manifold modeling

In the previous chapters, we proposed an efficient and sparse density estimator: Fast Parzen Windows. For each component in the estimator, it is designed to capture the local low dimensional structure. Accordingly, the overall density is estimated aligned to the underlying manifold. From this chapter, we try to understand the global low dimensional structures embedded in dataset, while estimating the density estimation. The research results in a multi-manifolds aligned density estimation with explicit multi-manifolds modeling. We first present a new manifold expanding algorithm, which is employed in the multi-manifold learning framework when initializing the multiple manifolds aligned density model .

In this chapter, we first briefly review one generative model based manifold learning algorithm: Generative Topographic Mapping (GTM) model, and demonstrate that the classical initialization of the GTM model may fail to capture a non-linear low dimensional structure. Then, Section 2 describes our proposed novel manifold expanding algorithm. The low dimensional structure discovered by the new approach is used to initialize the GTM model. In Section 3, we test our algorithm on several artificial datasets and real astronomical simulation datasets where interesting structures are embedded. Finally, Section 4 concludes this chapter.

4.1 Generative Topographic Mapping

Generative Topographic Mapping (GTM) [4] defines a non-linear, parametric mapping from a d-dimensional latent space ($\boldsymbol{\tau} \in \mathcal{R}^d$) to a D-dimensional data space ($\mathbf{x} \in \mathcal{R}^D$). Figure 4.1 illustrates an example of the mapping from 2-D latent space to the 3-D data space of GTM.

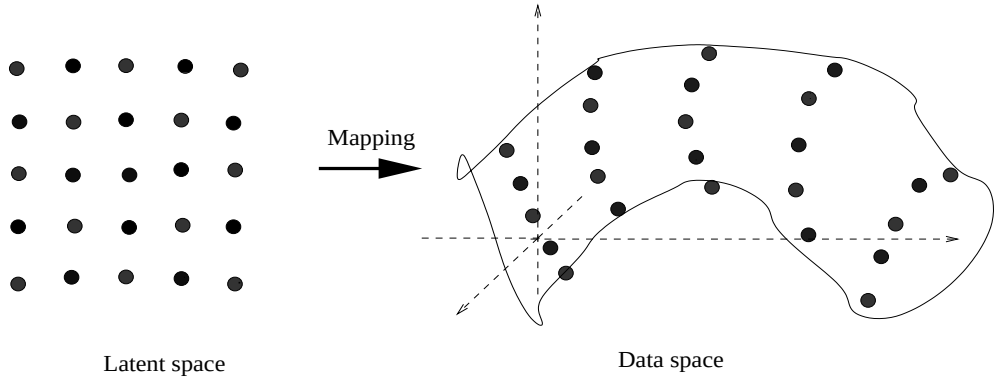


Figure 4.1: Illustration of mapping from the 2-D latent space to the 3-D data space of GTM.

4.1.1 Formulation of GTM

We use an equally spaced grid with M nodes ($\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_M$) to represent the latent space in GTM. Then, by associating an equally weighted delta function on each node of the regular grid, the probability distribution over the latent space $p(\boldsymbol{\tau})$ reads

$$p(\boldsymbol{\tau}) = \frac{1}{M} \sum_{m=1}^{M} \delta(\boldsymbol{\tau} - \boldsymbol{\tau}_m), \tag{4.1}$$

where $\boldsymbol{\tau}$ is 2-D latent variable and

$$\delta(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad \mathbf{x} = 0, \\ 0 & \text{if} \quad \mathbf{x} \neq 0. \end{cases} \tag{4.2}$$

To induce a corresponding probability distribution in the data space, we convolve it with an isotropic Gaussian noise distribution and obtain:

$$\begin{aligned} p(\mathbf{x}|\boldsymbol{\tau}, \mathbf{W}, \beta) =& \mathcal{N}(\mathbf{y}(\boldsymbol{\tau}, \mathbf{W}), \beta) \\ =& (\frac{\beta}{2\pi})^{-D/2} \exp\left\{ -\frac{\beta}{2} ||\mathbf{x} - y(\boldsymbol{\tau}, \mathbf{W})||^2 \right\}, \end{aligned} \tag{4.3}$$

where $\mathbf{x}$ is a point in the data space, β^{-1} denotes the noise variance and $\mathbf{y}(\boldsymbol{\tau}, \mathbf{W})$ represents the mapping of the latent grid $\boldsymbol{\tau}$ in the data space. Note that we have defined

$$\mathbf{y}(\boldsymbol{\tau}; \mathbf{W}) = \mathbf{W}\Phi(\boldsymbol{\tau}) \tag{4.4}$$

where $\Phi(\boldsymbol{\tau})$ is a column vector with K fixed basis functions $\phi_j(\boldsymbol{\tau})$ as follows:

$$\Phi(\boldsymbol{\tau}) = [\phi_1(\boldsymbol{\tau}), \phi_2(\boldsymbol{\tau}), \ldots, \phi_K(\boldsymbol{\tau})]^T, \tag{4.5}$$

and $\mathbf{W}$ is a $D \times K$ coefficient matrix.

By integrating out the latent variable, we get the probability distribution in the data space expressed as a function of the parameters β and $\mathbf{W}$,

$$\begin{aligned} p(\mathbf{x}|\mathbf{W}, \beta) =& \int p(\mathbf{x}|\boldsymbol{\tau}, \mathbf{W}, \beta) p(\boldsymbol{\tau}) d\boldsymbol{\tau} \\ =& \frac{1}{M} \sum_{m=1}^{M} p(\mathbf{x}|\boldsymbol{\tau}_m, \mathbf{W}, \beta). \end{aligned} \tag{4.6}$$

4.1.2 Optimization of GTM

Given a dataset of i.i.d points in the data space, we use E-M algorithm to maximize the likelihood of $p(\mathbf{x}|\mathbf{W}, \beta)$ and fit the parameters $\mathbf{W}$ and β to the dataset. The log likelihood function is given by

$$\mathcal{L}(\mathbf{W}, \beta) = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|\mathbf{W}, \beta). \tag{4.7}$$

In the E-step, we use old value of the parameters $\mathbf{W}$ and β ($\mathbf{W}_{old}$ and β_{old}) to calculate the responsibilities of each Gaussian component i for every data point $\mathbf{x}_n$ by using Bayes' theorem:

$$\begin{aligned} R_{in}(\mathbf{W}_{old}, \beta_{old}) =& p(\boldsymbol{\tau}_i|\mathbf{x}_n, \mathbf{W}_{old}, \beta_{old}) \\ =& \frac{p(\mathbf{x}_n|\boldsymbol{\tau}_i, \mathbf{W}_{old}, \beta_{old})}{\sum_{m=1}^{M} p(\mathbf{x}_n|\boldsymbol{\tau}_m, \mathbf{W}_{old}, \beta_{old})}. \end{aligned} \tag{4.8}$$

Therefore, the expectation of the complete-data log likelihood is written in the form:

$$< L_{comp}(\mathbf{W}, \beta) > = \sum_{n=1}^{N} \sum_{i=1}^{M} R_{in}(\mathbf{W}_{old}, \beta_{old}) \{\ln p(\mathbf{x}_n|\boldsymbol{\tau}.\mathbf{W}, \beta)\} \tag{4.9}$$

Maximizing (4.9) with respect to **W**, and using (4.3) and (4.5), we have

$$\sum_{n=1}^{N}\sum_{i=1}^{M} R_{in}(\mathbf{W}_{old}, \beta_{old})\{\mathbf{W}_{new}\Phi(\boldsymbol{\tau}_i) - \mathbf{x}_n\}\Phi^T(\boldsymbol{\tau_i}) = 0. \tag{4.10}$$

In matrix notation, it could be written as

$$\mathbf{\Phi}^T\mathbf{G}_{old}\mathbf{\Phi}\mathbf{W}_{new}^T = \mathbf{\Phi}^T\mathbf{R}_{old}\mathbf{T} \tag{4.11}$$

where:

- $\mathbf{\Phi}$ is a $M \times K$ matrix with elements $\phi_{ij} = \phi_j(\mathbf{x}_i)$
- $\mathbf{X}$ is a $N \times D$ matrix with elements x_{nk}
- $\mathbf{R}$ is a $M \times N$ matrix with element R_{in}
- $\mathbf{G}$ is a $M \times M$ diagonal matrix with elements $G_{ii} = \sum_{n=1}^{N} R_{in}(\mathbf{W}, \beta)$

Similarly, maximizing Eq. (4.9) with respect to β, we obtain the following re-estimation formula

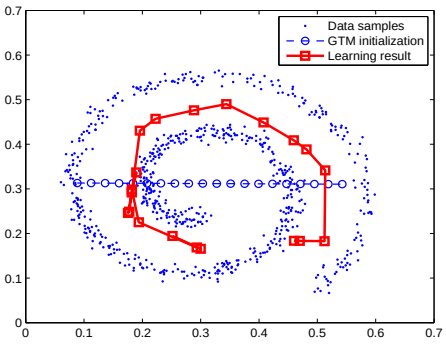
$$\frac{1}{\beta} = \frac{1}{ND}\sum_{n=1}^{N}\sum_{i=1}^{M} R_{in}(\mathbf{W}_{old}, \beta_{old})||\mathbf{W}_{new}\Phi(\boldsymbol{\tau}_i) - \mathbf{x}_n||^2. \tag{4.12}$$

4.1.3 Initialization of GTM

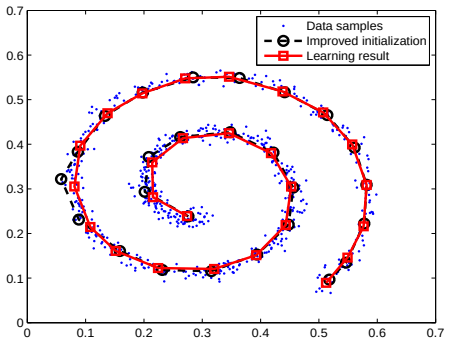
In principle, we determine the optimal value for parameters **W** and β for GTM by maximizing the log likelihood function through the E-M algorithm. These parameters are traditionally initialized such that the GTM model initially approximates principle component analysis (PCA)[4]. To do this, we first evaluate the data covariance matrix and obtain the first and second principle eigenvectors before determining **W** by minimizing the error function

$$E = \frac{1}{2}\sum_{i=1}^{M}||\mathbf{W}\Phi(\boldsymbol{\tau_i}) - \mathbf{U}\boldsymbol{\tau_i}||^2 \tag{4.13}$$

where the columns of **U** are given by the eigenvectors. The value of β^{-1} is initialized to be the larger one of either the $L+1$ eigenvalue from PCA (representing the variance of the data away from the PCA plane), or the square of half of the grid spacing of the



(a) Initialization and learning result of classical GTM



(b) GTM initialized with the new manifold learning approach

Figure 4.2: Learning results of GTM model with different initializations: plot (a) illustrates the classical initialization of GTM and the corresponding learning result. Plot (b) illustrates the initialization given by our proposed manifold learning algorithm and the learning result.

PCA-projected latent points in the data space. However, in the case that a strong non-linear manifold structure is embedded in the data, the optimization procedure with PCA linear global initialization cannot guarantee the global optimum.

Figure 4.2 demonstrates such an example. The data set we used is generated from the following distribution of two dimensional (x_1, x_2) points:

$$x_1 = 0.02t\sin(t) + 0.3 + \epsilon_{x_1}; \qquad x_2 = 0.02t\cos(t) + 0.3 + \epsilon_{x_2} \tag{4.14}$$

where $t \sim \mathcal{U}(3, 15)$, $\epsilon_{x_1} \sim \mathcal{N}(0, 0.01)$, $\epsilon_{x_2} \sim \mathcal{N}(0, 0.01)$, $\mathcal{U}(a, b)$ denotes the uniform distribution over the interval (a, b) and $\mathcal{N}(0, \delta)$ is the zero-mean Gaussian distribution with the standard deviation δ. In Figure 4.2, we illustrate the initialization and training results of classical GTM on the dataset described above. For the left plot, the initial Gaussian centers are obtained by mapping the pre-defined, one dimensional latent variables through global PCA. The learning result with such an initialization approximates the non-linear spiral data manifold poorly. In contrast to Figure 4.2 (a), the right plot shows an much improved result by the GTM initialized with our proposed method described below.

In the following, we describe the new manifold expanding approach demonstrated above. The latent space of the low dimensional structure is represented by a special

graph in low dimensional space, and an associated high dimensional graph is used to represent the embedded shape in data space. The proposed manifold expanding approach uses an iterative algorithm to learn the graphs in the latent space and the data space simultaneously. These two graphs representing one specific manifold can be easily used to initialize GTM. We verify the effectiveness of the proposed algorithm on the artificial datasets. Numerical experiments also demonstrate that our algorithm is capable of detecting a manifold in a noisy environment.

4.2 Manifold learning algorithm

This section introduces a novel method to discover the embedded manifold and capture its non-linear structure, so that we could provide a more reliable initialization to the GTM model. The learning procedure is proposed under the following assumptions: **1)** data points in the high dimensional space are lying along a low-dimensional manifold upon some noise and **2)** there is only one manifold (low-dimensional structure) which is smooth and connected.

To represent the latent space and its associated mapping in high-dimensional space in the learning procedure, we employ two special graphs, $\mathcal{G}$ (latent space graph) and $\mathcal{G}^m$ (data space graph) as shown in Figure 4.3. It illustrates the graphs ($\mathcal{G}$ and $\mathcal{G}^m$) of the 2-D structure embedded in 3-D data space.

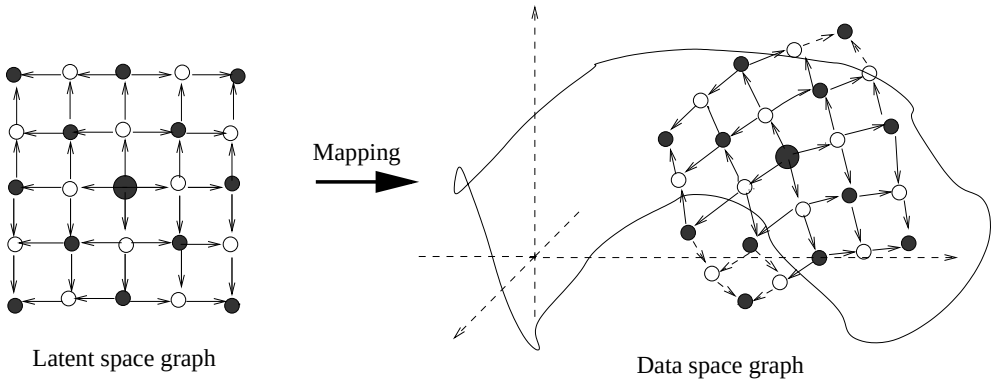


Figure 4.3: Illustration of mapping from the 2-D latent space to the 3-D data space of GTM. The 2-D latent space is represented by the latent space graph. The manifold embedded in 3-D space is represented by the data space graph.

Our algorithm starts by planting a "seed" of the latent and data space graphs ($\mathcal{G}$

and $\mathcal{G}^m$) on the latent space and the underlying low-dimensional structure embedded in the data space, respectively. These "seeds" are termed origins of the graphs for the rest of this thesis. Then, we iteratively discover the whole connected low dimensional structure in the data space as the "seed" gradually grows along the low dimensional structure. Taking this 2-D case as an example in Figure 4.3, we explain the manifold expanding process in the following subsections.

4.2.1 Notations in the algorithm

Prior to describing our manifold learning approach, we first define some notations. As shown in Figure 4.3, both the latent space graph and the data space graph are composed of vertices and directed edges connecting them. We use $\mathcal{Y}$ and $\mathcal{X}$ to denote the collections of the vertices in the latent space and data space graphs respectively.

Each vertex $\mathbf{Y}_i$ of the latent space graph ($\mathbf{Y}_i \in \mathcal{Y}$) contains the following attributes:

- i: index of the element in the collection.
- $\bar{\mathbf{y}}_i$: coordinates of the vertex in the graph.
- E_i^o: the directions of the outgoing edges. The outgoing edges of the vertex point to the vertices connected to it. In our implementation, we call the vertices connected by the outgoing edges "children".
- E_i^i: the directions of the incoming edges. The incoming edges of the vertex come from other vertices and point to the vertex itself. Likewise, we call the vertices connected by the incoming edges "parents".
- l: the length of the edge (since all edges in the graph have the same length, no index is set to l).
- Pa_i: the set of indices of its parents (since the values are the same for the latent and data space graphs, we use the same notation for both of them).

Each vertex $\mathbf{X}_i$ in collection $\mathcal{X}$ contains similar attributes: index i, coordinates $\bar{\mathbf{x}}_i$, directions of the outgoing edges $\mathcal{E}_i^o$, directions of the incoming edges $\mathcal{E}_i^i$, edge length L and the indices of its parents Pa_i.

The size of the outgoing edges of the node ($E^o/\mathcal{E}^o$) is the number of "children", and the size of the incoming edges indicates the number of "parents". For this particular

form of the graph used in our method, if the dimension of the graph is $d = 2$, the vertex of the graph has at most $2d = 4$ edges. Therefore, the sum of the number of "parents" and "children" is 4.

According to the number of vertices' parents, we partition all the vertices of latent space graph $\mathcal{Y}$ and obtain $\mathcal{Y} = \mathcal{Y}_0 \cup \mathcal{Y}_1 \cup \mathcal{Y}_2$, where $\mathcal{Y}_0$, $\mathcal{Y}_1$ and $\mathcal{Y}_2$ represent collections of vertices with $0-$, $1-$ and $2-$ parents[1].Similarly, we obtain the same partition of data space graph $\mathcal{X} = \mathcal{X}_0 \cup \mathcal{X}_1 \cup \mathcal{X}_2$, where $\mathcal{X}_0$, $\mathcal{X}_1$ and $\mathcal{X}_2$ represent collections of vertices with $0-$, $1-$ and $2-$ parents in the data space graph respectively. The origin of the latent space graph $\mathbf{Y}_0$ and the data space graph $\mathbf{X}_0$ are the unique, parentless vertices of the graphs in collections $\mathcal{Y}_0$ and $\mathcal{X}_0$ respectively. Section 4.2.2 explains how to initialize the origins. Other vertices are learnt from their parents. In Section 4.2.3, we describe one iteration for learning other vertices from their parents. The overall iterative algorithm to obtain all the vertices from the origins is explained in Section 4.2.4.

4.2.2 Initialization

The initialization of the latent and data space graphs is to set the attributes of $\mathbf{Y}_0$ and $\mathbf{X}_0$ respectively.

We set the attributes of $\mathbf{Y}_0$ (origin of the latent space graph) as follows:

1. The coordinate of the latent space graph origin is $\bar{\mathbf{y}}_0 = (0, 0)$;
2. The collection of the outgoing edges directions from the origin is E_0^o=$\{(1, 0), (-1, 0), (0, 1), (0, -1)\}$;

   The collection of the incoming edges directions to the origin is empty: E_0^i=$\emptyset$;
3. Edge length $l = 1$.
4. The set of parents' indexes of the origin is empty: $Pa_0 = \emptyset$.

The first step of initializing the origin of the data space graph ($\mathbf{X}_0$) is to set its coordinate. In this step, we need to ensure that the origin chosen is located on the underlying low dimensional structure, so that the graph learning procedure described below can continue when the dataset contains outliers (points which are not generated from

[1]The largest possible number of parents in 2-D latent space graph is 2.

the low dimensional structure). This can be achieved by checking the distribution of its neighbors. Therefore, one threshold parameter is required to check the density. After that, we determine the directions of its outgoing edges according to the local manifold patch. Different approaches can be used to learn the local manifold patch and some will be discussed in Section 4.2.5. Finally, we set the edge length L according to the local neighbors. Similar to the setting of the origin $\mathbf{Y}_0$ for the latent space graph, the collections of incoming edges and its parents' indices is empty. We summarize the initialization step as follows:

1. Randomly select one data point $\mathbf{x}$ from dataset $\mathcal{D}$.
2. If $\mathbf{x}$ is on the manifold (the local density around $\mathbf{x}$ is larger than a threhold parameter), then set it as $\mathbf{X}_0$'s coordinate: $\bar{\mathbf{x}}_0$=$\mathbf{x}$ ($\mathbf{x} \in \mathcal{D}$ and $\mathbf{x}$ is generated from the embedded manifold). Otherwise, go back step 1.
3. Find the neighborhood points of origin $\mathbf{X}_0$ from dataset $\mathcal{D}$, the set of the neighbours is denoted as $\mathcal{K}(\mathbf{X}_0, \mathcal{D})$.
   (a) The set of the outgoing edges' directions from the origin is $\mathcal{E}_0^o$=$\{\mathbf{u}_1, \ldots, \mathbf{u}_d, -\mathbf{u}_1, \ldots, -\mathbf{u}_d\}$,
   where $\{\mathbf{u}_1, \ldots, \mathbf{u}_d\}$ are the first d principal directions of $\mathcal{K}(\mathbf{X}_0, \mathcal{D})$.
   (b) The set of the incoming edges' directions to the origin is empty: $\mathcal{E}_0^i = \emptyset$.
   (c) Edge length L is set to be the average of the distance from its neighbors to $\mathbf{X}_0$.
   (d) The set of parents' indexes of the origin is empty: $Pa_0 = \emptyset$.

4.2.3 Vertex learning

Given the "seeds" of the graphs (the origin of the graphs), we obtain all the other vertices during the "growing" procedure. In the following, we explain how to obtain the vertices from their parents for both the latent and data space graphs separately.

Given a vertex in a latent space graph ($\mathbf{Y}_i$), we represent the collection of the coordinates of its children as $Ch(\mathbf{Y}_i)$. $Ch(\mathbf{Y}_i)$ is set deterministically by using the following

attributes of vertex $\mathbf{Y}_i$: coordinates ($\bar{\mathbf{y}}_i$), edge length (l) and the directions of the outgoing edges (E_i^o) as shown:

$$Ch(\mathbf{Y}_i) \quad = \quad \bar{\mathbf{y}}_i + lE_i^o \tag{4.15}$$

where the size of $Ch(\mathbf{Y}_i)$ is equal to the size of E_i^o. The coordinate $\bar{\mathbf{y}}_i$ is obtained from its parent(s) by using $Ch(\mathbf{Y}_{Pa_i})$, l is the same for all the edges and E_i^o is to be decided in the vertex learning.

To obtain the directions of the vertex's outgoing edges, we first indicate three local manifold patches of different vertex types in Figure 4.4. These are vertices with no parent, one parent and two parents. The edge directions for these three types are shown in Figure 4.5.

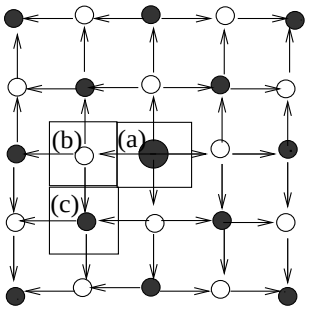
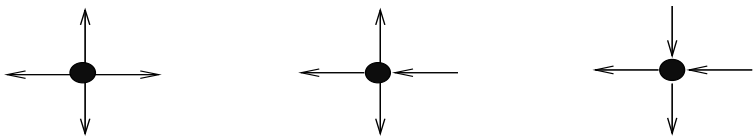


Figure 4.4: Local manifold patches associated with different types of vertices in the latent space graph.



(a) $\mathbf{Y}_i \in \mathcal{Y}_0$ (b) $\mathbf{Y}_i \in \mathcal{Y}_1$ (c) $\mathbf{Y}_i \in \mathcal{Y}_2$

Figure 4.5: Edges of a vertex having no parent, one parent and two parents in latent space graph.

Therefore, we determine its outgoing directions E_i^o of the given vertex $\mathbf{Y}_i$ according to its parents Pa_i and its incoming edges E_i^i as follows:

- A vertex having no parent ($\mathbf{Y}_i \in \mathcal{Y}_0$) is the origin of the latent space graph, the directions of its connected edges are set by the initialization.

- A vertex having only one parent ($\mathbf{Y}_i \in \mathcal{Y}_1$) inherits the outgoing directions from its parent. But if $\mathbf{Y}_i$ is connected to the origin, we need to remove the direction which is opposite to its incoming direction from its inheritance.
- The directions of the outgoing edges of vertex having two parents ($\mathbf{Y}_i \in \mathcal{Y}_2$) are the same as the directions of its incoming edges.

It can be written in the following way:

$$E_i^o = \begin{cases} E_0^o & \mathbf{Y}_i \in \mathcal{Y}_0 \\ E_{pa_i}^o \backslash -E_i^i & \mathbf{Y}_i \in \mathcal{Y}_1 \\ E_i^i & \mathbf{Y}_i \in \mathcal{Y}_2 \end{cases} \tag{4.16}$$

where $A \backslash B$ is the operation of removing items from A if they are also in B and $-B$ returns the opposite direction of B. The directions of the incoming edges of the new vertex and the indexes of its parents are collected in the learning process.

Likewise, we could write out the coordinates of the children vertices of a given vertex $\mathbf{X}_i$ ($Ch(\mathbf{X}_i)$) of the data space graph in the same format as in the latent space graph (Eq. (4.15)):

$$Ch(\mathbf{X}_i) \quad = \quad \bar{\mathbf{x}}_i + L\mathcal{E}_i^o, \tag{4.17}$$

where $\bar{\mathbf{x}}_i$ is the coordinates of vertex $\mathbf{X}_i$, L is the fixed edge length, and $\mathcal{E}_i^o$ is the collection of directions of the outgoing edges connected to vertex $\mathbf{X}_i$.

Similar to the latent space graph, we also have three different cases in the data space graph while estimating the directions of the outgoing edges. In Figure 4.6, we illustrate the manifold patches (a), (b) and (c) in the data space graph, which are associated with the manifold patches (a), (b), (c) in the latent space graph (shown in Figure 4.4).

According to our assumption **2)**, we know that the local linear manifold patches (a) and (b) (or (b) and (c)) in the data space are smoothly connected, but not on the same plane. Therefore, we need to project the outgoing edges direction onto the local manifold patch to ensure that the child vertex is learnt along the local manifold patch. Figure 4.7 illustrates the relation between the direction of one outgoing edge (e') and the direction of the incoming edge (e) from its parent vertex on the nearby local manifold patch. If we denote the projection matrix of the manifold patch around the new

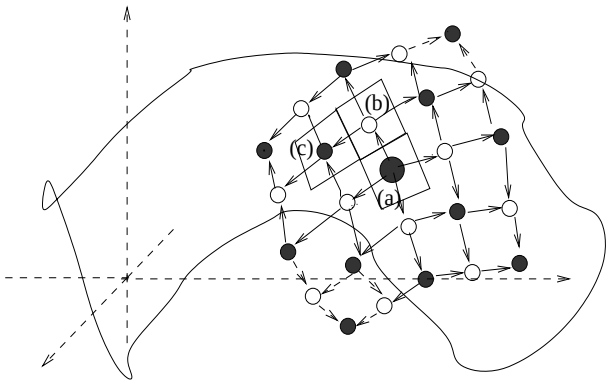


Figure 4.6: Local manifold patches associated with the vertices in the data space graph.

vertex $\mathbf{X}_i$ as $\mathbf{M}_{\mathbf{X}_i}$, we can calculate the direction e' along the manifold patch from the incoming direction e along the manifold patch nearby as follows:

$$e' = \mathbf{M}_{\mathbf{X}_i} e. \tag{4.18}$$

The projection matrix $\mathbf{M}_{\mathbf{X}_i}$ is constructed by the unit and orthogonal basis vectors $\{\mathbf{u}_1, \ldots, \mathbf{u}_d\}$ of the local manifold patch $\mathbf{X}_i$ in the following way:

$$\mathbf{M}_{\mathbf{X}_i} = [\mathbf{u}_1, \ldots, \mathbf{u}_d][\mathbf{u}_1, \ldots, \mathbf{u}_d]^T. \tag{4.19}$$

In our implementation, the $\{\mathbf{u}_1, \ldots, \mathbf{u}_d\}$ are obtained by decomposing the robust covariance matrix. The implementation details are discussed in Subsection 4.2.5.

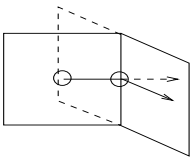


Figure 4.7: Relation of the edges connected to vertices on the neighboring local manifold patches in the data space.

Following the same rule as in the latent space graph and projecting the directions of the edges in the connected patches, we can obtain directions of the outgoing edges of vertex $\mathbf{X}_i$:

$$\mathcal{E}_i^o = \mathbf{M}_{\mathbf{X}_i} \mathcal{E}_i^*, \tag{4.20}$$

where

$$\mathcal{E}_i^* = \begin{cases} \mathcal{E}_0^o & \mathbf{X}_i \in \mathcal{X}_0 \\ \mathcal{E}_{pa_i}^o \setminus -\mathcal{E}_i^i & \mathbf{X}_i \in \mathcal{X}_1 \\ \mathcal{E}_i^i & \mathbf{X}_i \in \mathcal{X}_2. \end{cases} \tag{4.21}$$

Using the children vertices learning step described in this subsection, we then iteratively learn all the vertices of the latent and data space graphs to represent the manifold.

4.2.4 Manifold expanding process

Starting from the origins of both the latent and data space graphs, we use an iterative algorithm to obtain its children vertices and recover the whole manifold gradually.

To implement the expanding algorithm, we introduce another two sets $\tilde{\mathcal{Y}}$ and $\tilde{\mathcal{X}}$ to store the vertices to be processed for the latent space graph and the data space graph respectively. The steps are briefly stated as follows:

1. Set $\tilde{\mathcal{Y}} = \{\mathbf{Y}_0\}$, $\tilde{\mathcal{X}} = \{\mathbf{X}_0\}$ (collections of the vertices of the latent and data space graphs which will be processed).
2. Set $\mathcal{Y} = \emptyset$, $\mathcal{X} = \emptyset$ (collections of the vertices of the latent and data space graphs).
3. While $\tilde{\mathcal{Y}} \neq \emptyset$ ($\tilde{\mathcal{X}} \neq \emptyset$), repeat:
   - Select the first element of sets $\tilde{\mathcal{Y}}$ and $\tilde{\mathcal{X}}$: $\mathbf{Y}$ and $\mathbf{X}$ ($\mathbf{Y} \in \tilde{\mathcal{Y}}$ and $\mathbf{X} \in \tilde{\mathcal{X}}$).
   - Remove $\mathbf{Y}$ and $\mathbf{X}$ from sets $\tilde{\mathcal{Y}}$ and $\tilde{\mathcal{X}}$ respectively: $\tilde{\mathcal{Y}} \leftarrow \tilde{\mathcal{Y}} \setminus \{\mathbf{Y}\}$, $\tilde{\mathcal{X}} \leftarrow \tilde{\mathcal{X}} \setminus \{\mathbf{X}\}$, then add them to sets $\mathcal{Y}$ and $\mathcal{X}$ respectively: $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\mathbf{Y}\}$, $\mathcal{X} \leftarrow \mathcal{X} \cup \{\mathbf{X}\}$.
   - Learn the children vertices of $\mathbf{Y}$ and $\mathbf{X}$: $Ch(\mathbf{Y})$, $Ch(\mathbf{X})$.
   - Update the sets $\tilde{\mathcal{Y}}$ and $\tilde{\mathcal{X}}$ with $Ch(\mathbf{Y})$ and $Ch(\mathbf{X})$.

To update the $\tilde{\mathcal{Y}}$ with $Ch(\mathbf{Y})$ and $\tilde{\mathcal{X}}$ with $Ch(\mathbf{X})$, we need to consider the two following situations:

1. $Ch(\mathbf{X})$ is out of boundary.

   The newly learnt vertices from $Ch(\mathbf{X})$ may reach out of the boundary of the manifold. That means the vertex is not representing the manifold structure and should not be included in both $\tilde{\mathcal{X}}$ and $\mathcal{X}$. An idea to detect the vertex out of

the manifold bound is that, compared to the density of points in the manifold, the local density of points close to (or out of) the boundary decreases rapidly. The implementation of detecting the dropped local density may vary when using different ways to determine the local neighbors, the threshold parameter for detecting the density drop is closely related to the way of determine the local neighbors.

2. $Ch(\mathbf{Y})$ is in $\tilde{\mathcal{Y}}$ already.

   If $Ch(\mathbf{Y})$ is in the set $\tilde{\mathcal{Y}}$ already, it means that the vertex learnt from $Ch(\mathbf{Y})$ has been obtained from another parent before. Obviously the corresponding vertex $Ch(\mathbf{X})$ has been added to $\tilde{\mathcal{X}}$ as well. Therefore, we shall not add $Ch(\mathbf{Y})$ and $Ch(\mathbf{X})$ to the set $\tilde{\mathcal{Y}}$ and $\tilde{\mathcal{X}}$ respectively as a new vertex, but replace the existing vertex in $\tilde{\mathcal{X}}$ by the mean of the duplicated representations. Otherwise, we add new vertices $Ch(\mathbf{X})$ and $Ch(\mathbf{Y})$ to sets $\tilde{\mathcal{X}}$ and $\tilde{\mathcal{Y}}$ as appropriate.

To make the novel manifold expanding process more intuitive, we use a simple example to demonstrate the growing process of the latent and data space graphs ($\mathcal{G}$ and $\mathcal{G}^m$). The 3-D dataset we employed stays on a 2-D manifold and we sampled 2000 data points from the manifold in the following way:

$$x_1 = t + \epsilon_{x_1}; \qquad x_2 = t + \epsilon_{x_2}; \qquad x_3 = x_1(-x_1^2 + x_2^2) \tag{4.22}$$

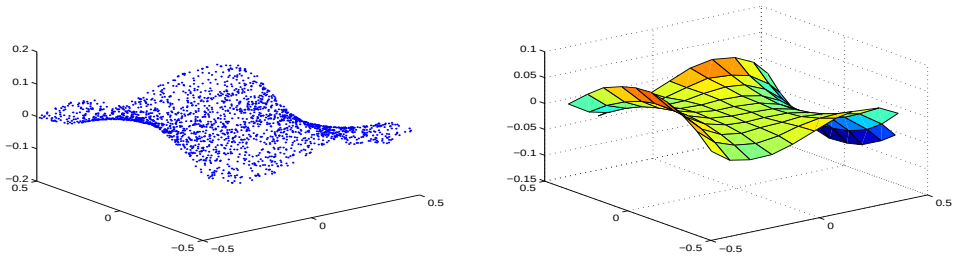
where $t \sim \mathcal{U}(-0.5, 0.5)$ generating a uniform variable between -0.5 and 0.5, and the noise $\epsilon_{x_1} \sim \mathcal{N}(0, 0.01)$ and $\epsilon_{x_2} \sim \mathcal{N}(0, 0.01)$.

First, we plot the data points generated from the distribution above in Figure 4.8 (a). The estimated low dimensional structure (constructed from connecting the nodes in the data space graph) is shown in Figure 4.8 (b).

Next, we use a series of subplots in Figure 4.9 to demonstrate the growing of the latent space and data space graphs in the learning process. The left column of Figure 4.9 show the plots of the vertices in the latent space graph, the right column shows the associated data space graphs and the data points.

4.2.5 Some details on learning local manifold patch

In our proposed manifold expanding algorithm, it is essential to estimate the local manifold patch accurately. Since we assume that the non-linear manifold is locally



(a) Data sampled from a 2-D underlying manifold

(b) Manifold learnt from our proposed manifold learning approach

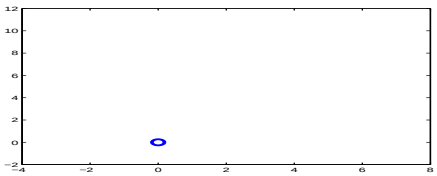
Figure 4.8: Plot (a) illustrates a 3-D dataset staying on a 2-D manifold. Plot (b) shows the manifold learnt by our proposed manifold expanding algorithm.

linear, the local manifold patch could be estimated by Principal Component Analysis (PCA). Finding the data points on the local manifold patch is equivalent to determining the neighbors of the given data point $\mathbf{x}$. The popular methods of determining the neighbors of $\mathbf{x}$ are the K nearest neighbor (KNN) algorithm and the r-radius neighbor algorithm [86]. In the KNN algorithm, the neighbors are set to be the K nearest data points. Although it is easy to implement, the KNN algorithm is sensitive to outliers. Compared to the KNN algorithm, the ϵ-radius neighbor method is more realistic to the true case, but it is very difficult to decide the essential parameter radius size ϵ. In PCA, the d dimensional ($d < D$) linear manifold is represented by d eigenvectors ($\{\mathbf{u}_1, \ldots, \mathbf{u}_d\}$) with the first d largest eigenvalues of the covariance matrix $S_{\mathbf{x}}$. The covariance matrix $S_{\mathbf{x}}$ is estimated as follows:

$$S_{\mathbf{x}} = \frac{\sum_{i=1}^{k} (\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T}{k},$$

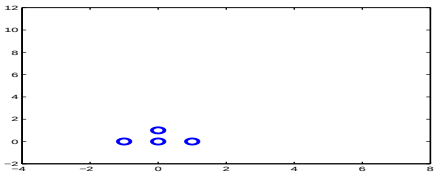
where $\mathbf{x}_i, i \in \{1, \ldots, k\}$ are k neighbors of the given data $\mathbf{x}$. In KNN, k is set as a parameter. In ϵ-radius neighbor algorithm, it is the number of data points whose distance to $\mathbf{x}$ is less than r.

Among many robust approaches for PCA [40, 87, 88, 89, 90, 91], a common and easy way to make the estimation robust to outliers thereby increasing the accuracy of the local manifold estimation is to replace the standard estimation of the covariance matrix $S_{\mathbf{x}}$ by a robust estimator of the covariance matrix $S^*_{\mathbf{x}}$. In our implementation,



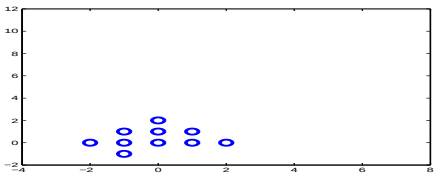
(a) Growing latent space graph 1

(b) Growing data space graph 1

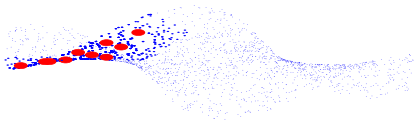


(c) Growing latent space graph 2

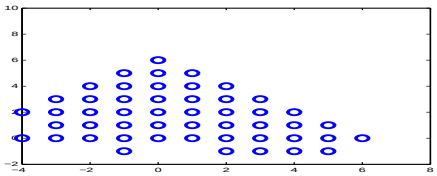
(d) Growing data space graph 2



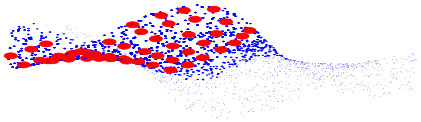
(e) Growing latent space graph 3



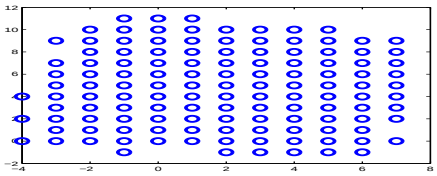
(f) Growing data space graph 3



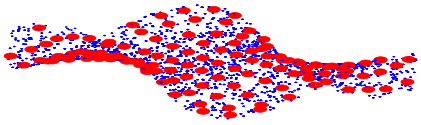
(g) Growing latent space graph 4



(h) Growing data space graph 4



(i) Growing latent space graph 5



(j) Growing data space graph 5

Figure 4.9: Illustration of the growing of the latent/data space graphs defined in our proposed manifold expanding process for learning the manifold shown in Figure 4.8.

the robust covariance matrix is calculated in the following way:

$$S^*_{\mathbf{x}} = \frac{\sum_{i=1}^{k'} K_r(\mathbf{x}_i;\mathbf{x})(\mathbf{x}_i-\mu_{\mathbf{x}})(\mathbf{x}_i-\mu_{\mathbf{x}})^T}{\sum_{i=1}^{k'} K_r(\mathbf{x}_i;\mathbf{x})}, \tag{4.23}$$

where $K_r(\mathbf{x}_i;\mathbf{x})$ is a weighting kernel (we use spherical Gaussian kernel), the kernel width r is one parameter in our manifold learning algorithm, k' is the number of the data points whose weights given by the weighting kernel are greater than 0.01 and

$$\mu_{\mathbf{x}} = \frac{\sum_{i=1}^{k'} K_r(\mathbf{x}_i;\mathbf{x})\mathbf{x}}{\sum_{i=1}^{k'} K_r(\mathbf{x}_i;\mathbf{x})}.$$

By considering that the data points close to the given data point $\mathbf{x}$ are more important when estimating the local manifold patch than the data points further away, calculating the eigenvalues and eigenvectors of this robust covariance matrix gives us eigenvalues that are robust to outliers.

4.3 Experiments

In this section, we use some synthetic datasets to verify our proposed manifold expanding algorithm. By initializing the GTM model from the learnt latent and data space graphs, we demonstrate the advantage of our manifold learning algorithm on learning non-linear manifolds. Then, we add noise to the manifold and test our proposed algorithm again. Finally, we use the proposed algorithm to detect the low dimensional structure in the astronomical datasets.

4.3.1 Initialization of GTM

Three non-linear artificial datasets are generated to verify our proposed work.

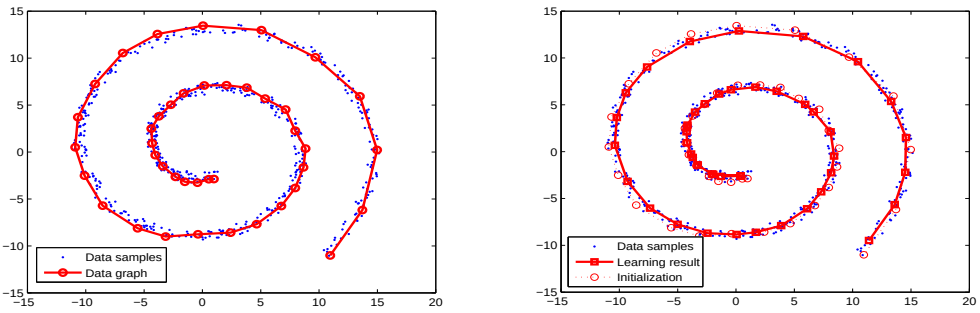
4.3.1.1 Dataset1: 2-D data along 1-D structure

The first dataset is similar to the dataset shown in Figure 4.2. The 2-D data points are lying along 1-D manifold and they are generated as follows:

$$x_1 = t\sin(t) + \epsilon_{x_1}; \qquad x_2 = t\cos(t) + \epsilon_{x_2}; \tag{4.24}$$

where $t \sim \mathcal{U}(3, 15)$, $\epsilon_{x_1} \sim \mathcal{N}(0, 0.01)$, $\epsilon_{x_2} \sim \mathcal{N}(0, 0.01)$, $\mathcal{U}(a, b)$ is the uniform distribution over the interval (a, b) and $\mathcal{N}(0, \delta)$ is the zero-mean Gaussian distribution with standard deviation δ.

In Figure 4.10 (a), we show the data space graph (with data points) learnt by using our proposed algorithm, which is then applied to initialize the GTM model. Figure 4.10 (b) illustrates the final manifold refined by GTM.



(a) Initialized GTM centers (b) Learnt GTM centers

Figure 4.10: 1-D manifold learnt (plot (b)) by the GTM model from 2-D dataset and its initialization (plot (a)) given by the proposed manifold learning algorithm.

4.3.1.2 Dataset2: 3-D data along 1-D structure

We extend the 2-D dataset into 3-D space and obtain the new dataset as follows:

$$x_1 = t\sin(t) + \epsilon_{x_1}; \qquad x_2 = t\cos(t) + \epsilon_{x_2}; \qquad x_3 = t + \epsilon_{x_3} \tag{4.25}$$

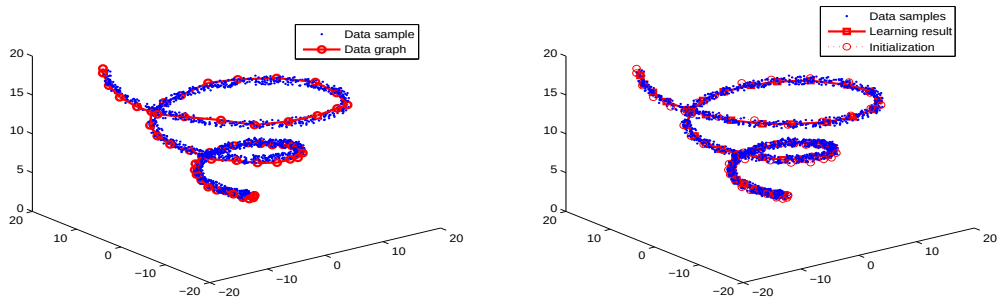
where $t \sim \mathcal{U}(3, 15)$, $\epsilon_{x_1} \sim \mathcal{N}(0, 0.01)$, $\epsilon_{x_2} \sim \mathcal{N}(0, 0.01)$ and $\epsilon_{x_2} \sim \mathcal{N}(0, 0.01)$.

Figure 4.11 (a) shows the data space graph (with data points) learnt by using our proposed model. Figure 4.11 (b) illustrates the manifold learnt by GTM model with the initialization of plot 4.11 (a).

4.3.1.3 Dataset3: 3-D data along 2-D structure

The third set is 3-D data lying along a 2-D structure, which are generated as follows:

$$x_1 = t\sin(t) + \epsilon_{x_1}; \qquad x_2 = \tilde{t} + \epsilon_{x_2}; \qquad x_3 = t\cos(t) + \epsilon_{x_3}; \tag{4.26}$$

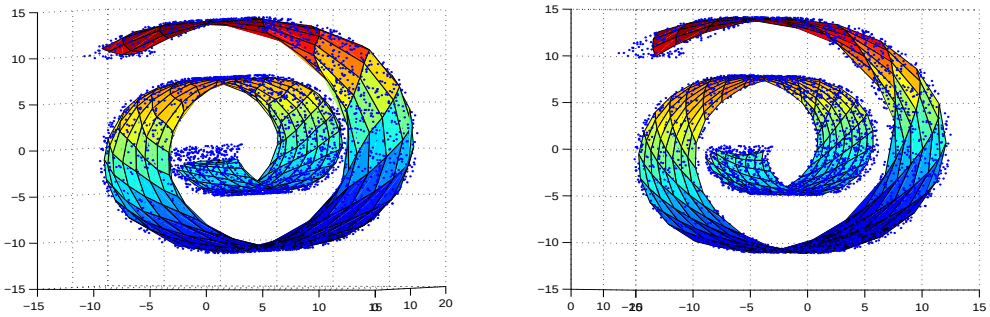


(a) Initialized GTM centers (b) Learnt GTM centers

Figure 4.11: 1-D manifold learnt (plot (b)) by the GTM model from the 3-D dataset and its initialization (plot (a)) given by the proposed manifold learning algorithm.

where $t \sim \mathcal{U}(3, 15)$, $\tilde{t} \sim \mathcal{U}(-5, 5)$, $\epsilon_{x_1} \sim \mathcal{N}(0, 0.01)$, $\epsilon_{x_2} \sim \mathcal{N}(0, 0.01)$ and $\epsilon_{x_3} \sim \mathcal{N}(0, 0.01)$.

In Figure 4.12 (a), we show the data space graph (with data points) learnt by using our proposed model, which is later used to initialize the GTM model. Figure 4.12 (b) illustrates the manifold learnt from the GTM model.



(a) Initialized GTM centers (b) Learnt GTM centers

Figure 4.12: 2-D manifold learnt (plot (b)) by the GTM model from the 3-D dataset and its initialization (plot (a)) given by the proposed manifold learning algorithm.

From these experimental results, it is clear that our proposed manifold learning algorithm can correctly detect and learn the non-linear low dimensional structures embedded in each dataset tested. Initializing the GTM model with the manifolds learnt by our algorithm, we obtain the model accurately representing the embedded low

dimensional structures.

4.3.2 Detection of the manifold in noisy environment

In this subsection, we investigate the performance of our proposed method on contaminated data. The dataset we used consists of 600 2-D data points generated from a 1-D manifold (the same one used in experiment 4.3.1) and outliers generated from a uniform distribution. The number of the outliers is represented by a certain percentage of the manifold datasets.

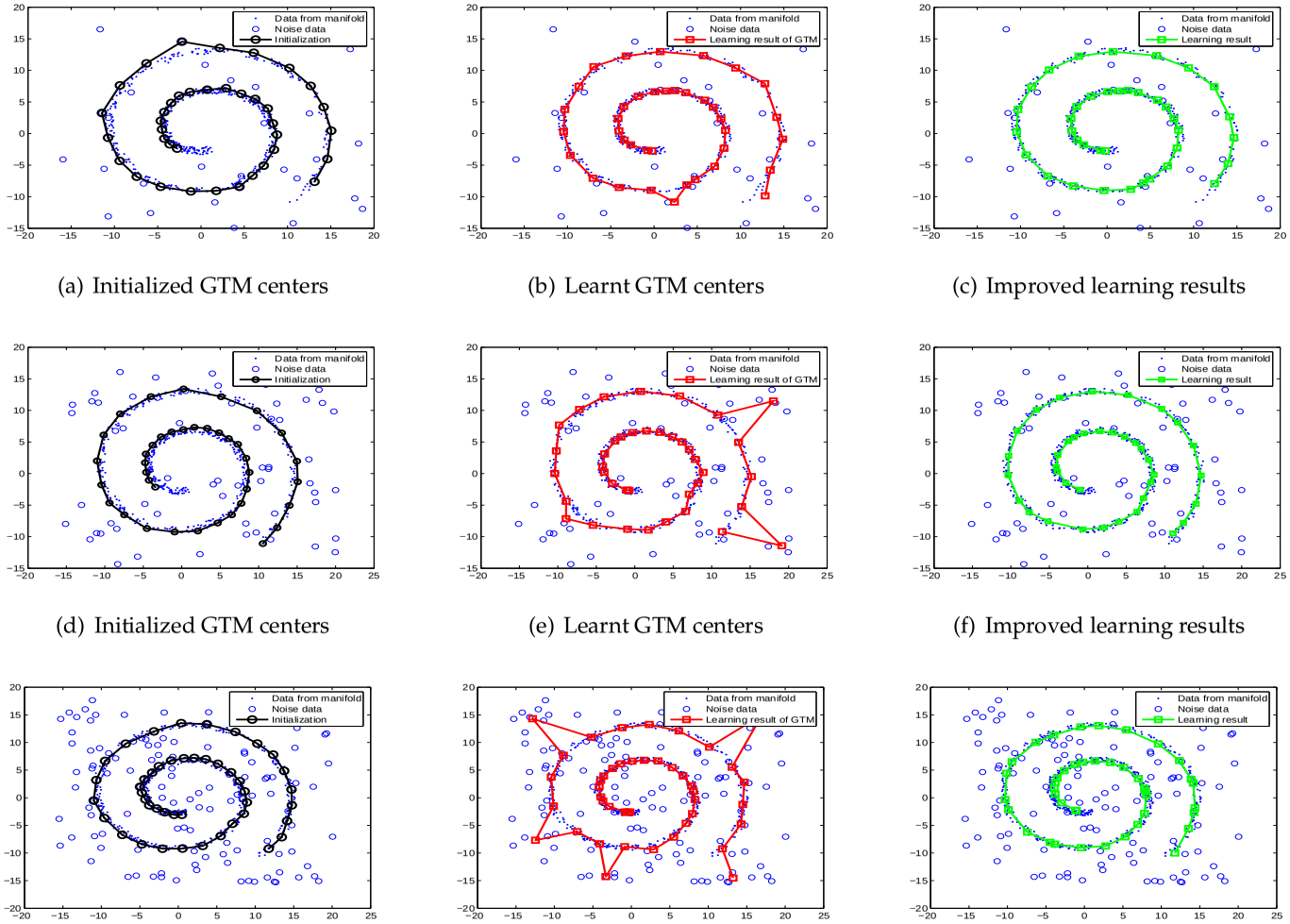
Figure 4.13 (a) illustrates the contaminated datasets with 5% of the outliers and the manifold learnt (data space graph with connected nodes) from our manifold learning algorithm. Then, by initializing the GTM model with the learnt graphs, we illustrate the refined manifold in Figure 4.13 (b).

Figures 4.13 (d) (e) and Figures 4.13 (g) (h) show the initialization and the learning results of GTM on the dataset with 10% and 20% outliers respectively. As shown in the figures, with a carefully chosen neighborhood size, our manifold learning algorithm is capable of recovering the non-linear low dimensional structure from the noisy environment. While, for the dataset containing more noise, we need to use a smaller neighborhood size to reduce the effect of the outliers and learn the local manifold accurately. Analysis of the parameter selection with respect to the amount of the outliers and the tolerance of our algorithm to outliers are left as future work. However, even with reasonable initializations of the GTM model, the noisy data points in the dataset still result in an incorrect manifold learning result.

Different works have been proposed to improve the robustness of the generative model towards outliers [92, 93]. In this experiment, we use another model representing the distribution of the noises. The results are learnt from a model that mixes the GTM and the noise model. The mixture model used here is actually a special case of the hierarchical model introduced in the next chapter. Subplots (c), (f) and (i) of Figure 4.13 illustrate the improved learning results. We can see that the embedded manifolds are correctly represented from the three contaminated datasets.

4.3.3 Real astronomical datasets

In this subsection, we further test our proposed manifold learning approach on the astronomical simulation data. The datasets are generated from a realistic astronomical



(a) Initialized GTM centers (b) Learnt GTM centers (c) Improved learning results

(d) Initialized GTM centers (e) Learnt GTM centers (f) Improved learning results

(g) Initialized GTM centers (h) Learnt GTM centers (i) Improved learning results

Figure 4.13: GTM initialization, learning result, and improved result on contaminated datasets with a different number of noise data points. (5%, 10% and 20% of each line)

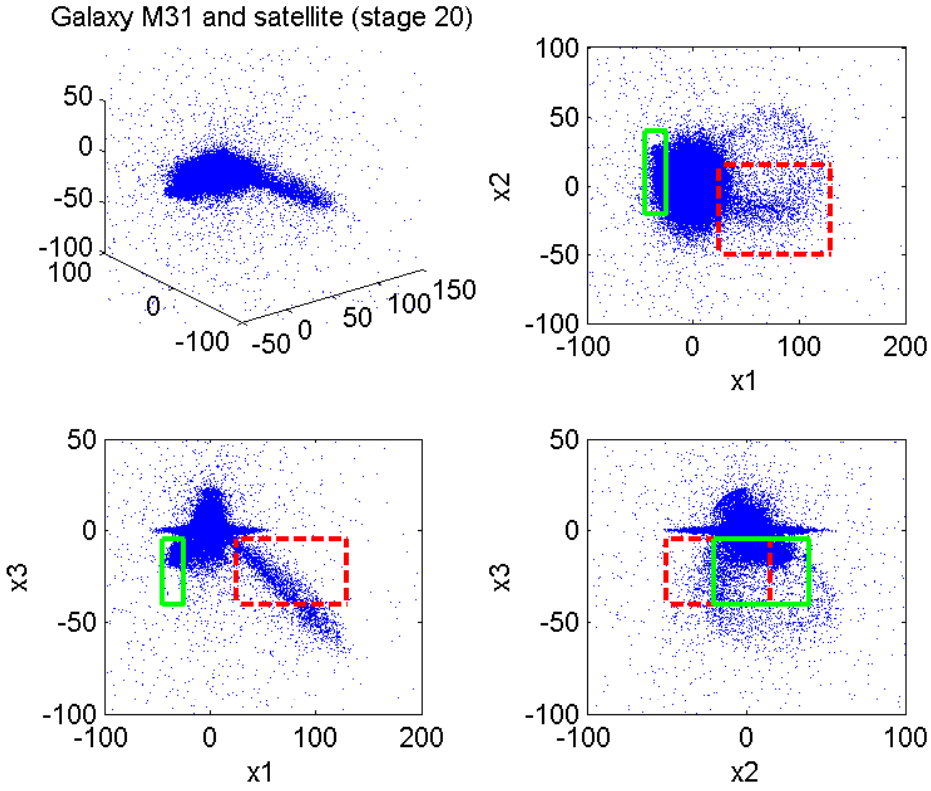


Figure 4.14: Simulation dataset at disruption stage 20: The upleft plot shows the spatial distribution. In the other 3 subplots, we view the simulation dataset in coordinate systems $x_1 - x_2$, $x_1 - x_3$ and $x_2 - x_3$ respectively and use dotted rectangles to mark the regions of "stream" data points and solid rectangles to mark the regions of "shell" data points.

particle model, which is used to simulate the disruption process of galaxy M31 and M32 (see Chapter 2).

First, we illustrate the spatial distribution of the simulated galaxies at disruption stage 20 in Figure 4.14 (the upper left plot). In the following three subplots, we illustrate the simulated galaxies on the coordinate systems $x_1 - x_2$, $x_1 - x_3$ and $x_2 - x_3$ respectively. In the vicinity of the main galaxy M31, we use a dotted rectangle to mark the region containing "stream" data points, and a solid rectangle to mark the region containing "shell" data points (where interesting low dimensional structures are observed by astronomers).

In the following, we use the proposed manifold learning algorithm to learn 2-D manifold from the "stream" data points at different disruption stages. Figure 4.15 (a) illustrates the "stream" data points of disruption stage 15. The manifold surface learnt by the presented approach is viewed from three different angles and shown as three subplots ((b), (c) and (d)) in the second line of Figure 4.15. Note that, during the galaxy disruption process, data points forming the "stream" gradually lose their energy and it makes the tail part of the stream sparse. Our manifold expanding algorithm terminates when the density of the "stream" becomes low and therefore misrepresents the structure at the tail of "stream". We initialize the GTM with the 2-D manifold learnt and impose the prior knowledge about the distribution of "stream" data points to the GTM model learning process. The three different views of the refined manifold illustrated in subplots (e), (f) and (g) respectively (the third line) of Figure 4.15 show that the improved manifold recovers the sparse area of the "stream" data points.

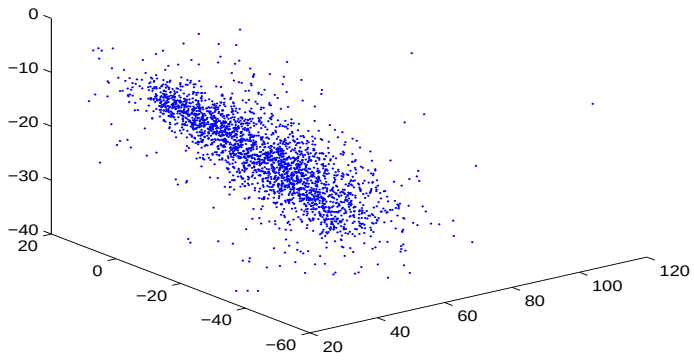
For the "stream" data points of simulations at the disruption 20 and 25, we show the learning results in Figure 4.16 and Figure 4.17.

We also apply the proposed manifold learning approach to explore "shell" data points of the simulation dataset generated at different disruption stages. In Figure 4.18, we illustrate the "shell" data points of the disruption stage 15 in subplot (a) and the 2-D structure of different views in subplots (b), (c) and (d). Moreover, the results on the stages 20 and 25 are reported in Figure 4.19 and 4.20 respectively. To refine the manifold learnt from the "shell" data points, we need to know the prior knowledge about the distribution of the "shell". Without this kind of information, we could not improve the manifold learnt from "shell" data points by GTM model.

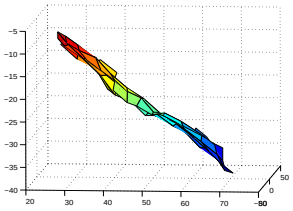
4.4 Summary

By noting that the GTM model with classical initialization may fail to capture the non-linear structure of the manifold, we proposed one novel manifold expanding approach to discover the underlying non-linear manifold. The presented algorithm starts with one point located in the manifold (called the "seed" in this chapter) and gradually recovers the connected manifold embedded by an iterative procedure.

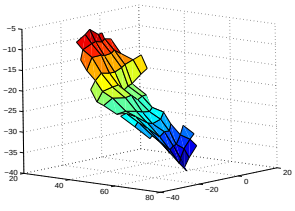
Our manifold learning algorithm is verified on a set of synthetic datasets generated from non-linear manifolds upon some noise. Initializing the GTM with the man-



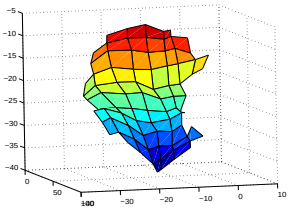
(a) "stream" data points at disruption stage 15 (R=3.5)



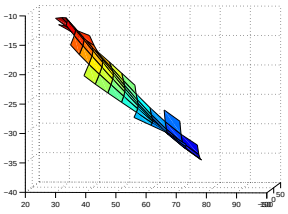
(b) Initialization (view 1)



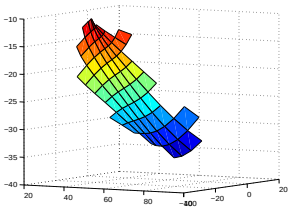
(c) Initialization (view 2)



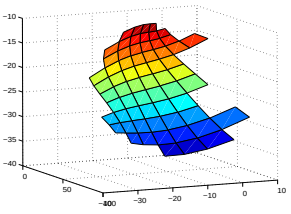
(d) Initialization (view 3)



(e) Learning result (view 1)

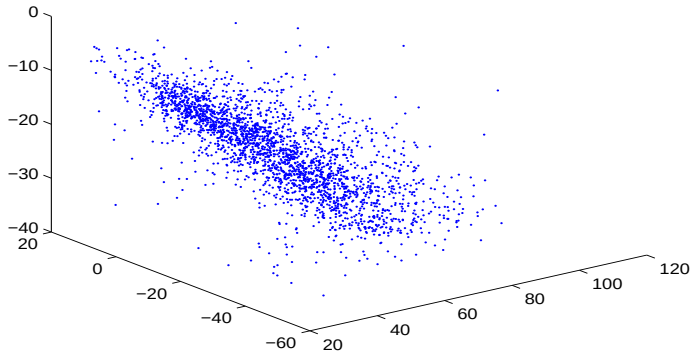


(f) Learning result (view 2)

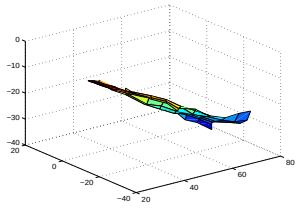


(g) Learning result (view 3)

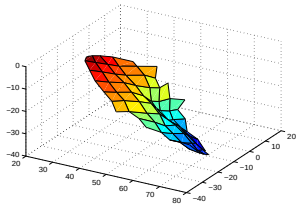
Figure 4.15: Manifold learnt from "stream" data points of simulations at disruption stage 15. Subplot (a) shows the original data points. The manifold surface learnt by our proposed algorithm is viewed from three different angles and plotted in three subplots of the second line. Subplots in the third line show the views of the improved manifold surface by GTM model.



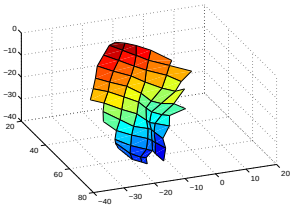
(a) "stream" data points at disruption stage 20 (R=4)



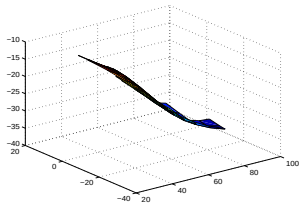
(b) Initialization (view 1)



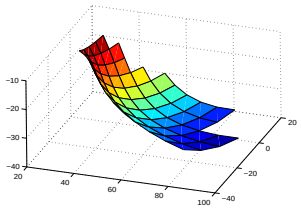
(c) Initialization (view 2)



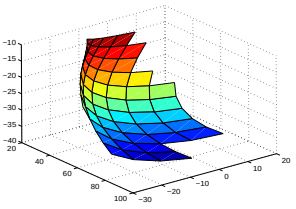
(d) Initialization (view 3)



(e) Learning result (view 1)

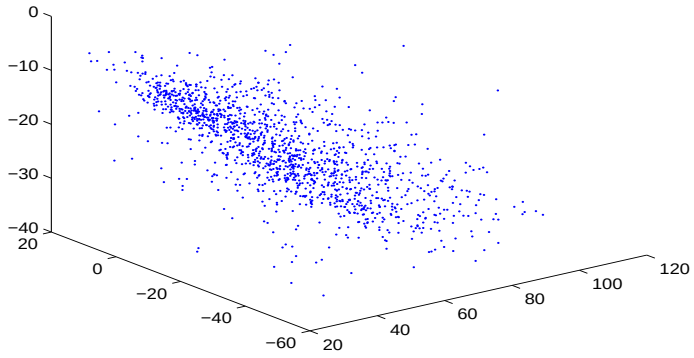


(f) Learning result (view 2)

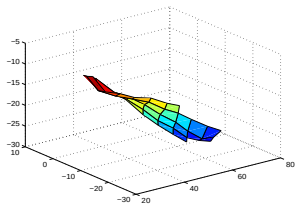


(g) Learning result (view 3)

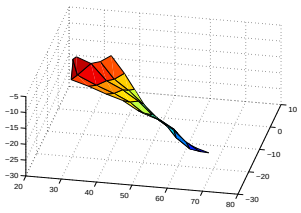
Figure 4.16: Manifold learnt from "stream" data points of simulations at disruption stage 20. Subplot (a) shows the original data points. The manifold surface learnt by our proposed algorithm is viewed from three different angles and plotted in three subplots of the second line. Subplots in the third line show the views of the improved manifold surface by GTM model.



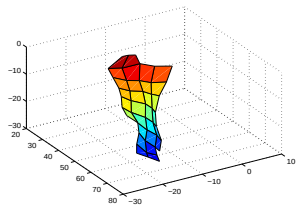
(a) "stream" data points at disruption stage 25 (R=4)



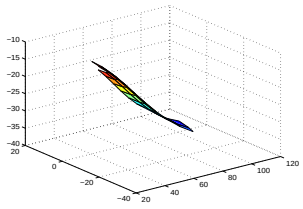
(b) Initialization (view 1)



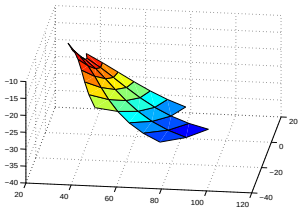
(c) Initialization (view 2)



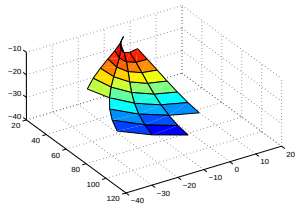
(d) Initialization (view 3)



(e) Learning result (view 1)

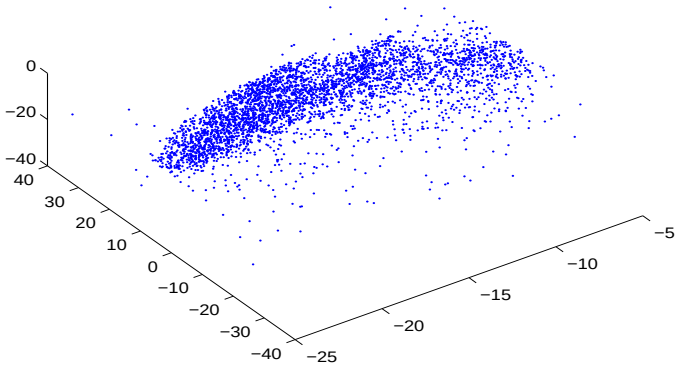


(f) Learning result (view 2)

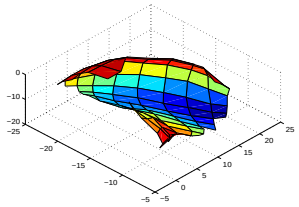


(g) Learning result (view 3)

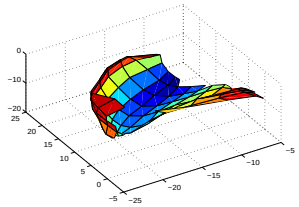
Figure 4.17: Manifold learnt from "stream" data points of simulations at disruption stage 25. Subplot (a) shows the original data points. The manifold surface learnt by our proposed algorithm is viewed from three different angles and plotted in three subplots of the second line. Subplots in the third line show the views of the improved manifold surface by GTM model.



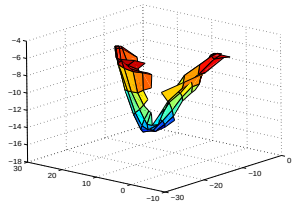
(a) "shell" data points at disruption stage 15 (R=1.5)



(b) Initialization (view 1)

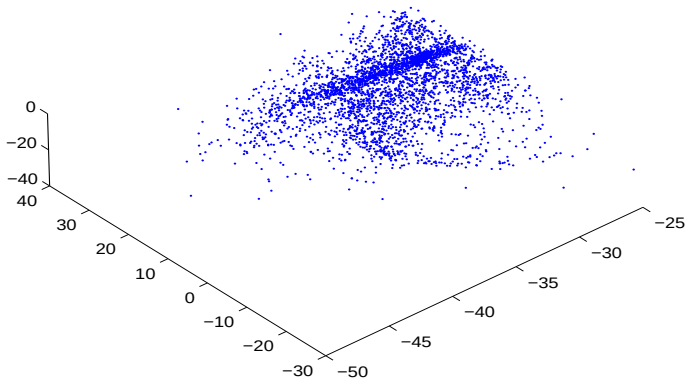


(c) Initialization (view 2)

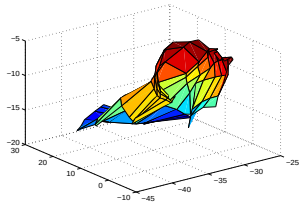


(d) Initialization (view 3)

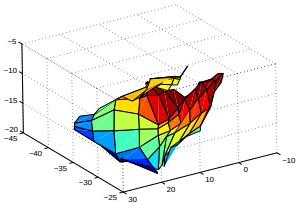
Figure 4.18: Manifold learnt from "shell" data points of simulations at disruption stage 15. Subplot (a) shows the original data points. The second line show the manifold surface learnt by our proposed algorithm from different views.



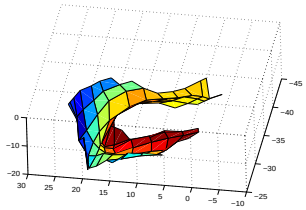
(a) "shell" data points at disruption stage 20 (R=1.8)



(b) Initialization (view 1)

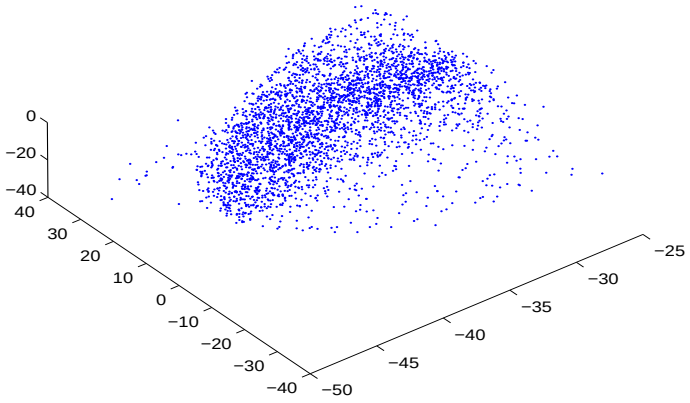


(c) Initialization (view 2)

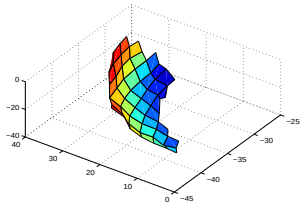


(d) Initialization (view 3)

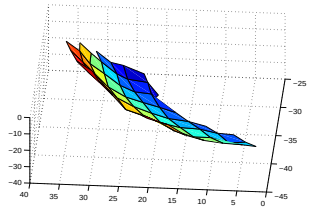
Figure 4.19: Manifold learnt from "shell" data points of simulations at disruption stage 20. Subplot (a) shows the original data points. The second line show the manifold surface learnt by our proposed algorithm from different views.



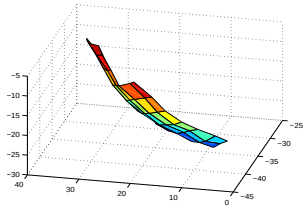
(a) "shell" data points at disruption stage 25 (R=2)



(b) Initialization (view 1)



(c) Initialization (view 2)



(d) Initialization (view 3)

Figure 4.20: Manifold learnt from "shell" data points of simulations at disruption stage 20. Subplot (a) shows the original data points. The second line show the manifold surface learnt by our proposed algorithm from different views.

ifold learnt by the presented approach, the learning results correctly represent the non-linear structure embedded. We also demonstrate that our manifold learning algorithm is capable of learning the manifold in a noisy environment. Using the real galaxy simulation dataset, we could successfully detect the 2-D manifold found in the vicinity of the main galaxy (M31) in the simulated galaxy disruption process.

The work in this chapter lays a foundation for describing a multi-manifold learning framework in the next chapter.

Chapter 5

Multiple manifolds aligned density estimation

In the last chapter, we investigated the estimation of the probability density on data points along a single low dimensional structure. For most manifold aligned estimators and manifold learning algorithms in the literature, the data in high dimensional spaces are predominantly assumed to be aligning (up to some noise) along a single underlying manifold. However, in reality for some complex datasets, it may be generated from more than one low dimensional structure. Figure 5.1 illustrates an example where the 3-D data are generated from both 1-D and 2-D manifolds. In the previous chapter, we use one generative model to describe the distribution of the data with only one structure. To represent the mix of the datasets aligned along different manifolds, an intuitive way is to use a mixture of generative models, where each generative model describes each manifold.

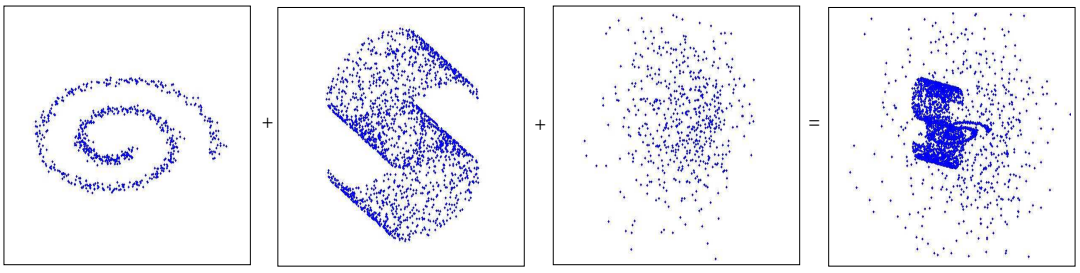


Figure 5.1: Multiple manifolds example: 700 3-D points aligned along a 1-D manifold, 2000 3-D points lying on a 2-D manifold and 600 3-D points generated form a mixture of 3 Gaussians.

In this chapter, we review some existing hierarchical density models designed to reveal the interesting local structures. Then we propose a mixture model to describe the complex dataset such as the example shown in Figure 5.1 and a framework to initialize the model to capture the data properly in Section 2. Section 3 demonstrates experimental results on artificial data and a dataset produced by realistic galaxy collision models. Finally, Section 4 concludes this chapter and discusses the directions of our future work.

5.1 Related works

To reveal the interesting local structures in these complex data sets, [94] proposed a hierarchical visualization algorithm based on a hierarchical mixture of latent variable models such as Probabilistic Principal Component Analysis (PPCA) or Factor Analysis (FA). The algorithm is designed to construct a hierarchical tree top down and provides successively refined models of the dataset. The complete data set is visualized at the top-level projection, perhaps revealing the presence of clusters. Driven interactively by users, at the lower-level hierarchy, projections display internal structure within individual clusters such as the presence of subclusters, which may not be apparent in higher-level projections. Tino and Nabney [95] extended this visualization algorithm by replacing the latent variable model with non-linear visualization block GTM (Generative Topographic Mapping) so that the non-linear projection manifolds could be visualized. Structures of the hierarchy in these visualization systems are also built interactively. [96] proposed a non-interactive hierarchy construction. A general framework for a principled hierarchical visualization of multivariate data was given in [97].

Similar to these hierarchical visualization models, we use one submodel to represent one particular interesting structure embedded in the dataset. The mixture of these submodels can be represented in a hierarchical manner. We construct the mixture model by first partition the entire dataset into several subsets of different intrinsic dimensionalities and then discover the low dimensional structures in each subset. Intrinsic dimension [98] of a dataset refers to the dimension of subspace where the data points lies. It is the dimension of low dimensional representation of the manifold

embedded in the dataset. In most of the manifold learning methods, the intrinsic dimension of the manifold d is either treated as prior knowledge given by the user or as a parameter to be estimated. [99] divides intrinsic dimension estimators into two families: global approaches and local methods. Global approaches [100, 101] unfold the whole dataset in the D-dimensional space and estimate the intrinsic dimension. However, for local methods such as Fukunaga-Olsen's algorithm [102], nearest neighbor algorithm [103, 104, 105] and methods based on topological representing networks [106, 107], the intrinsic dimensions are estimated using the information contained in sample neighborhoods.

As in the case of manifold learning algorithms, most of these intrinsic dimensionality estimators assume that all the data points are aligned along a single 'manifold'. A point level dimensionality estimator proposed in [108] is appealing due to the ability to deal with manifolds of different dimensionalities. The authors first represent each data point by a second order, symmetric, non-negative definite tensor, whose eigenvalues and eigenvectors fully describe the local dimensionality and orientation at each point. Then a voting procedure accumulates votes from its neighbors and provides an estimate of the local dimensionality.

5.2 Density model along multiple manifolds

In this section, we extend the probability density estimation of data points along a single manifold to the case of containing data points along multiple manifolds. As shown in Figure 5.1, data points are aligned along different manifolds of different dimensionalities.

The mixture model we proposed can be represented by a hierarchical tree structure with three levels. The leaves (third level) of the tree are the submodels of the mixture density model and they are generative models associated with the noisy manifolds of different intrinsic dimensionalities. The second-level of the hierarchy tree represents the mixture of the generative models clustered according to the intrinsic dimension. The root (top-level) of the hierarchical tree is the mixture of the mixture models for the overall dataset.

5.2.1 Model formulation

Given a complex dataset $\mathcal{D}=\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, we assume that it is the mixture of the subsets generated from more than one low-dimensional structure of different intrinsic dimensionalities. If we knew from which low dimensional structure, the data points were generated, we could cluster the points according to their sources. Accordingly, one subset or subcluster will be collecting the data points generated from one source.

In our model, for the data generated from the m-th manifold of intrinsic dimension d, we use probabilistic model $p(\mathbf{x}|d, m)$ formulated as a GTM model to describe its distribution along it:

$$\begin{aligned} p(\mathbf{x}|m,d) =& \frac{1}{Z^{m,d}} \sum_{z=1}^{Z^{m,d}} p(\mathbf{x}|\boldsymbol{\tau}_z, \mathbf{W}^{m,d}, \beta^{m,d}) \\ =& \frac{1}{Z^{m,d}} \sum_{z=1}^{Z^{m,d}} (\frac{\beta^{m,d}}{2\pi})^{-D/2} \exp\left\{-\frac{\beta^{m,d}}{2}||\mathbf{x}-\mathbf{y}(\boldsymbol{\tau}_z, \mathbf{W}^{m,d})||^2\right\} \end{aligned} \tag{5.1}$$

where $Z^{m,d}$ represents the number of the latent space centers in GTM model $p(\mathbf{x}|m,d)$, $\boldsymbol{\tau}_z$ represents the z-th latent space center, $\mathbf{W}^{d,m}$ and $\beta^{d,m}$ are the GTM parameters, and $\mathbf{y}(\boldsymbol{\tau}_z, \mathbf{W}^{m,d})$ is formulated as in Eq. (4.4).

If the data points have intrinsic dimension $d = D$, it is assumed that they are not generated from the low dimensional manifolds. We denote the mixture model of data points having intrinsic dimension d as $\mathcal{T}_d$ ($d \in \{1, \ldots, D\}$). The Gaussian mixture model $p(\mathbf{x}|\mathcal{T}_D)$ is used to model the data points having intrinsic dimension D:

$$\begin{aligned} p(\mathbf{x}|\mathcal{T}_D) &= \sum_{m=1}^{M_D} p(m)\mathcal{N}(\mathbf{x}|m) \\ &= \sum_{m=1}^{M_D} p(m) \frac{1}{\sqrt{2\pi||\boldsymbol{\Sigma}_m||}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_m)^T\boldsymbol{\Sigma}_m^{-1}(\mathbf{x}-\boldsymbol{\mu}_m)\right\} \end{aligned} \tag{5.2}$$

where M_D is the number of Gaussian components, $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m$ are the mean and covariance of the m-th component, and $p(m)$ is the mixing coefficient with $\sum_{m=1}^{M_D} p(m) = 1$. Submodels $p(\mathbf{x}|m,d)$ and $p(\mathbf{x}|\mathcal{T}_D)$ are the leaves of the hierarchical tree.

Now we could obtain the mixture model $\mathcal{T}_d$ for the data points generated from the manifolds of intrinsic dimension d as

$$p(\mathbf{x}|\mathcal{T}_d) = \sum_{m=1}^{M_d} \pi_{m|d}\, p(\mathbf{x}|m,d) \tag{5.3}$$

where M_d is the number of manifolds of the dimension d and $\pi_{m|d}$ ($\sum_{m=1}^{M_d} \pi_{m|d} = 1$) is the mixing coefficient.

The model $\mathcal{T}$ for the entire dataset $\mathcal{D}$ is formulated as mixture of the mixture models $p(\mathbf{x}|\mathcal{T}_d)$ ($d \in \{1, \ldots, D\}$):

$$p(\mathbf{x}|\mathcal{T}) = \sum_{d=1}^{D} \pi_d \, p(\mathbf{x}|\mathcal{T}_d) \tag{5.4}$$

where d is the dimension of the manifold, π_d ($\sum_{d=1}^{D} \pi_d = 1$) is the mixing coefficients of the probability density model for subsets lying on manifolds of dimension d, $\mathcal{T}_d$ ($1 \leq d < D$) is the mixture model for the manifolds with the dimensionality d.

The structure of the model $\mathcal{T}$ described above can be illustrated as a three-level tree in Figure 5.2. Each leaf in the third level corresponds to one manifold, indexed by m and d. For the manifolds having the same dimensionality, we cluster them together as one node in the second level. The top level of the tree represents the overall model on the whole dataset.

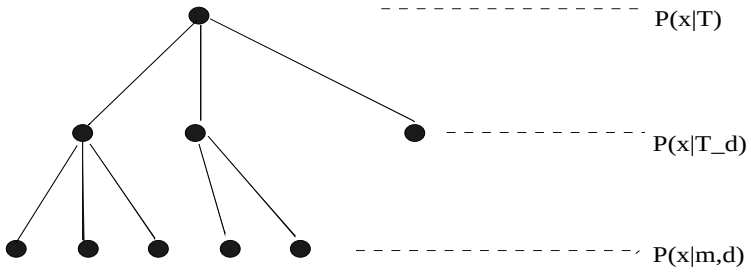


Figure 5.2: Tree representation of our proposed mixture model: The top level represents the overall model of the whole dataset, the nodes in the second level denote the submodels of the manifolds having the same intrinsic dimensionality, each leaf in the third level corresponds to one manifold indexed by m and d.

5.2.2 Parameters optimization

We learn the parameters of model (5.4) by maximizing the log likelihood function. It includes the mixing coefficients at every level and the parameters in each submodel.

Given the training data $\mathcal{D}$=$\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, the likelihood function of the proposed

model is

$$\mathcal{L} = \sum_{n=1}^{N} \ln p(\mathbf{x}_n|\mathcal{T}), \tag{5.5}$$

where $p(\mathbf{x}_n|\mathcal{T})$ is formulated as in Eq. (5.4).

In this model, we have three types of hidden variables:

- Latent binary assignment variables $\upsilon_{n,d}$ represents that the data $\mathbf{x}_n$ is generated from a manifold of intrinsic dimension d. If data $\mathbf{x}_n$ is indeed generated from one manifold of intrinsic dimension d, then $\upsilon_{n,d} = 1$. Otherwise, $\upsilon_{n,d} = 0$.

- When $1 \leq d < D$, we use latent binary assignment $\upsilon_{n,m|d}$ to represent the situation that data $\mathbf{x}_n$ is generated from the m-th manifold of dimension d.

  When $d = D$, variables $\upsilon_{n,m|D}$ is used to represent the case that data $\mathbf{x}_n$ is generated from m-th component of the Gaussian mixture model.

- For each submodel $p(\mathbf{x}_n|d,m)$, we still need to decide which latent space center $\boldsymbol{\tau}_z^{d,m}$, $z = 1, 2, \ldots, Z^{d,m}$ (where $Z^{d,m}$ represents the number of latent space centers) in the latent variable model $p(\mathbf{x}|d,m)$ corresponds to the Gaussian generating $\mathbf{x}_n$. We represent this by indicator variables $\upsilon_{n,z}^{d,m}$.

If we knew the values of the assignment variables $\upsilon_{n,d}$, $\upsilon_{n,m|d}$ and $\upsilon_{n,z}^{d,m}$ explained above, the log likelihood function (5.5) could be written as the complete data log likelihood function:

$$\begin{aligned}\mathcal{L}_c = &\sum_{n=1}^{N}\sum_{d=1}^{D-1}\upsilon_{n,d}\sum_{m=1}^{M_d}\upsilon_{n,m|d}\sum_{z=1}^{Z^{d,m}}\upsilon_{n,z}^{d,m}\ln\left\{\pi_d\pi_{m|d}p(\mathbf{x}_n|d,m)\right\}+\\ &\sum_{n=1}^{N}\upsilon_{n,D}\sum_{m'=1}^{M_D}\upsilon_{n,m'|D}\ln\left\{p(m')\mathcal{N}(\mathbf{x}_n|m')\right\}\end{aligned} \tag{5.6}$$

Taking the expectation (with respect to the posterior, given the data) over all types of hidden variables, we arrive at the expected complete-data likelihood

$$\begin{aligned}<\mathcal{L}_c> = &\sum_{n=1}^{N}\sum_{d=1}^{D-1}R_{d|n}\sum_{m=1}^{M_d}R_{m|d,n}\sum_{z=1}^{Z^{d,m}}R_{z|m,d,n}\ln\{\pi_d\pi_{m|d}p(\mathbf{x}_n,|d,m)\}+\\ &\sum_{n=1}^{N}R_{D|n}\sum_{m'=1}^{M_D}R_{m'|D,n}\ln\left\{p(m')\mathcal{N}(\mathbf{x}_n|m')\right\},\end{aligned} \tag{5.7}$$

where $R_{d|n}$, $R_{m|d,n}$ and $R_{z|m,d,n}$ are the posteriors of latent variables $\upsilon_{n,d}$,$\upsilon_{n,m|d}$ and $\upsilon_{n,z}^{d,m}$ respectively.

The first term in Eq. (5.7) could be written as the sum of the following three terms:

$$\sum_{n=1}^{N}\sum_{d=1}^{D-1} R_{d|n} \ln \pi_d, \tag{5.8}$$

$$\sum_{n=1}^{N}\sum_{d=1}^{D-1} R_{d|n} \sum_{m=1}^{M_d} R_{m|d,n} \ln \pi_{m|d} \tag{5.9}$$

and

$$\sum_{n=1}^{N}\sum_{d=1}^{D-1} R_{d|n} \sum_{m=1}^{M_d} R_{m|d,n} \ln p(\mathbf{x}_n|d,m). \tag{5.10}$$

Similarly, we rewrite the second term in Eq. (5.7) as the sum of the following three terms:

$$\sum_{n=1}^{N} R_{D|n} \ln \pi_d, \tag{5.11}$$

$$\sum_{n=1}^{N} R_{D|n} \sum_{m'=1}^{M_D} R_{m'|D,n} \ln p(m') \tag{5.12}$$

and

$$\sum_{n=1}^{N} R_{D|n} \sum_{m'=1}^{M_D} R_{m'|D,n} \ln \mathcal{N}(\mathbf{x}_n|d,m'). \tag{5.13}$$

Summing up the terms in Eq. (5.8) and Eq. (5.11), we obtain a new term:

$$\sum_{n=1}^{N}\sum_{d=1}^{D} R_{d|n} \ln \pi_d \tag{5.14}$$

Assuming the responsibilities $R_{d|n}$, $R_{m|d,n}$, $R_{z|m,d,n}$, $R_{D|n}$ and $R_{m'|D,n}$ are fixed, the M-step of the EM algorithm involves

- maximizing (5.14) w.r.t. the mixture coefficients 'π_d',
- maximizing (5.9) w.r.t. the mixture coefficients '$\pi_{m|d}$' of the manifolds,
- maximizing (5.12) w.r.t. the mixture coefficients '$p(m')$' of the components of the Gaussian mixture model,

- maximizing (5.10) w.r.t. the parameters $\mathbf{W}^{m,d}$ and $\beta^{m,d}$ of GTM model $p(\mathbf{x}_n|d,m)$,
- and maximizing (5.13) w.r.t. the parameters μ and Σ of Gaussian mixture model $p(\mathbf{x}_n|\mathcal{T}_D)$.

The maximization of Eq. (5.14) with respect to π_d needs to take account of the constraint

$$\sum_{d=1}^{D} \pi_d = 1. \tag{5.15}$$

This can be achieved by introducing a Lagrange multiplier λ and maximizing

$$\sum_{n=1}^{N}\sum_{d=1}^{D} R_{d|n} \ln \pi_d + \lambda(\sum_{d=1}^{D} \pi_d - 1). \tag{5.16}$$

This gives

$$\hat{\pi}_d = \frac{1}{N}\sum_{n=1}^{N} R_{d|n} \tag{5.17}$$

Similarly, the maximization of Eq. (5.9) with respect to $\pi_{m|d}$ needs to take account of the constraint

$$\sum_{m=1}^{M_d} \pi_{m|d} = 1. \tag{5.18}$$

Again, we introduce a Lagrange multiplier λ, then maximizing

$$\sum_{n=1}^{N}\sum_{d=1}^{D-1} R_{d|n} \sum_{m=1}^{M_d} R_{m|d,n} \ln \pi_{m|d} + \lambda(\sum_{m=1}^{M_d} \pi_{m|d} - 1) \tag{5.19}$$

yields:

$$\hat{\pi}_{m|d} = \frac{\sum_{n=1}^{N} R_{d|n} R_{m|d,n}}{\sum_{n=1}^{N} R_{d|n}}. \tag{5.20}$$

Also, the maximization of Eq. (5.12) with respect to $p(m')$ needs to consider the constraint

$$\sum_{m'=1}^{M_D} p(m') = 1. \tag{5.21}$$

Introducing a Lagrange multiplier λ, we maximize

$$\sum_{n=1}^{N} R_{D|n} \sum_{m'=1}^{M_D} R_{m'|D,n} \ln p(m') + \lambda(\sum_{m'=1}^{M_D} p(m') - 1) \tag{5.22}$$

and obtain

$$\hat{p}(m') = \frac{\sum_{n=1}^{N} R_{D|n} R_{m'|D,n}}{\sum_{n=1}^{N} \sum_{m'=1}^{M_D} R_{D|n} R_{m'|D,n}}. \tag{5.23}$$

Therefore, the mixing coefficients at each level of the proposed model are updated by Eq. (5.17), (5.20) and (5.23).

To update the parameters of the GTM models, we maximize Eq. (5.10) with respect to $\mathbf{W}^{m,d}$ and $\beta^{m,d}$. Using (4.4), (4.5) and (5.1), we have

$$\sum_{n=1}^{N} R_{n|d} \sum_{m=1}^{M_d} R_{m|d,n} \sum_{z=1}^{Z^{d,m}} R_{z|d,mn} (\mathbf{W}^{d,m} \Phi(\boldsymbol{\tau}_z^{d,m}) - \mathbf{x}_n) \Phi^T(\boldsymbol{\tau}_z^{d,m}) = 0. \tag{5.24}$$

The responsibilities $R_{d|n}$, $R_{m|d,n}$ and $R_{z|n,d,m}$ are calculated with the current (old) weights and inverse variance parameters of the probabilistic models $p(\mathbf{x}|d, m)$.

Written in matrix notation, we need to solve

$$(\mathbf{\Phi}^{d,m})^T \mathbf{B}^{d,m} (\mathbf{\Phi}^{d,m})^T (\mathbf{W}^{d,m})^T = (\mathbf{\Phi}^{d,m})^T \mathbf{R}^{d,m} \mathbf{T} \tag{5.25}$$

for $\mathbf{W}^{d,m}$.

The above system of linear equations involves the following matrices:

- $\mathbf{\Phi}^{d,m}$ is a $Z^{d,m} \times K^{d,m}$ matrix with elements $(\mathbf{\Phi}^{d,m})_{zj} = \phi_j(\boldsymbol{\tau}_z^{d,m})$.
- $\mathbf{T}$ is a $N \times D$ matrix storing the data points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ as rows.
- $\mathbf{R}^{d,m}$ is a $Z^{d,m} \times N$ matrix containing, for each latent space center $\mathbf{y}_z^{d,m}$ and each data point $\mathbf{x}_n$, scaled responsibilities $(\mathbf{R}^{d,m})_{zn} = R_{d|n} R_{m|n,d} R_{z|m,d,n}$
- $\mathbf{B}^{d,m}$ is a $Z^{d,m} \times Z^{d,m}$ diagonal matrix with diagonal elements corresponding to responsibilities of the latent space centers for the whole data sample $\mathcal{D}$, where $(\mathbf{B})_{zz} = \sum_{n=1}^{N} R_{d|n} R_{m|d,n} R_{z|m,d,n}$.

The GTM mapping can be regularized by adding a regularization term $\alpha^{d,m}$ [4] to the objective function. Inclusion of the regularizer modifies Eq. (5.25) to

$$\left[(\mathbf{\Phi}^{d,m})^T \mathbf{B}^{d,m} (\mathbf{\Phi}^{d,m})^T + \frac{\alpha^{d,m}}{\beta^{d,m}} \mathbf{I} \right] (\mathbf{W}^{d,m})^T = (\mathbf{\Phi}^{d,m})^T \mathbf{R}^{d,m} \mathbf{T}, \tag{5.26}$$

where $\mathbf{I}$ is the $Z^{d,m} \times Z^{d,m}$ identity matrix.

Finally, the re-estimation formulation for $\beta^{d,m}$ is

$$\frac{1}{\beta^{d,m}} = \frac{\sum_{n=1}^{N} R_{d|n} R_{m|d,n} \sum_{z=1}^{Z^{d,m}} R_{z|n,d,m} ||\mathbf{W}^{d,m} \phi(\mathbf{y}_z^{d,m}) - \mathbf{x}_n||^2}{D \sum_{n=1}^{N} R_{d|n} R_{m|d,n}}, \tag{5.27}$$

where $\mathbf{W}^{d,m}$ is the "new" weight matrix computed by solving (5.25) in the last step.

To update the parameters of the Gaussian mixture model $p(\mathbf{x}|\mathcal{T}_D)$, we maximize Eq. (5.13) with respect to $\boldsymbol{\mu}_{m'}$ and $\boldsymbol{\Sigma}_{m'}$ and obtain

$$\boldsymbol{\mu}_{m'} = \frac{\sum_{n=1}^{N} R_{D|n} R_{m'|D,n} \mathbf{x}_n}{\sum_{n=1}^{N} R_{D|n}} \tag{5.28}$$

$$\boldsymbol{\Sigma}_{m'} = \frac{\sum_{n=1}^{N} R_{D|n} R_{m'|D,n} (\mathbf{x}_n - \boldsymbol{\mu}_{m'})(\mathbf{x}_n - \boldsymbol{\mu}_{m'})^T}{\sum_{n=1}^{N} R_{D|n}}. \tag{5.29}$$

In the E-step of the EM algorithm, we estimate the latent space responsibilities $R_{z|n,d,m}$ in GTM models for the data generated from manifolds and component responsibilities $R_{m'|D}$ in the Gaussian mixture model:

$$R_{z|d,m,n} = \frac{p(\mathbf{x}_n|\boldsymbol{\tau}_z, d, m)}{\sum_{z'=1}^{Z^{d,m}} p(\mathbf{x}_n|\boldsymbol{\tau}_{z'}, d, m)} \tag{5.30}$$

$$R_{m|D,n} = \frac{p(m)\mathcal{N}(\mathbf{x}_n|m, D)}{\sum_{m'=1}^{M_D} p(m')\mathcal{N}(\mathbf{x}_n|m', D)} \tag{5.31}$$

Furthermore, the model responsibilities for manifolds of intrinsic dimension $R_{m|n,d}$ $(1 \leq d < D)$ and the intrinsic dimension responsibilities $R_{d|n} (1 \leq d \leq D)$ are specified as follows:

$$R_{m|d,n} = \frac{\pi_{m|d} p(\mathbf{x}_n|d, m)}{\sum_{m'=1}^{M_d} \pi_{m'|d} p(\mathbf{x}_n|d, m')} \tag{5.32}$$

$$R_{d|n} = \frac{\pi_d \sum_{m=1}^{M_d} \pi_{m|d} p(\mathbf{x}|d, m)}{\sum_{d'=1}^{D} \pi_{d'} \sum_{m'=1}^{M_{d'}} \pi_{m'|d'} p(\mathbf{x}_n|d', m')} \tag{5.33}$$

5.2.3 Multiple Manifolds Learning Framework

Similar to other E-M algorithm based parameter learning processes, our optimization also requires a proper initialization to guarantee a good learning result. In order to initialize the parameters of the GTM models $p(\mathbf{x}|d, m)$ according to the data points potentially generated from the manifold, we propose the following multiple mani-

folds learning framework to discover and learn the underlying manifolds of different dimensionalities. It includes **1)** estimating the intrinsic dimensions of the dataset $\mathcal{D}$ at point level and constructing the subsets $\mathcal{D}^d$ of intrinsic dimension d by clustering the data points according to the intrinsic dimensions d of the local manifold patches around them; **2)** discovering the d dimensional surfaces from the subset $\mathcal{D}^d$ by using the multiple manifolds learning algorithm (described at Section 5.2.3.2), learning and representing them by the graphs defined in the last chapter.

After that, we initialize the GTM models based on the learnt latent and data space graphs pairs. For the Gaussian mixture model $p(\mathbf{x}|\mathcal{T}_D)$, the corresponding parameters including Gaussian centers and covariances are initialized for example by a simple K-means algorithm [80]. The mixing coefficients could be estimated according to the proportion of each subset.

5.2.3.1 Step 1: Intrinsic Dimension Estimation

We estimate the intrinsic dimension of the given dataset $\mathcal{D}$ at point level. It is equivalent to estimating the dimensionality of the local manifold patch at each data point.

With the assumption that the manifold is locally linear, we represent the local patch by the covariance matrix of the points on it. In our implementation, the local covariance matrix is computed as in Eq. (4.23) in Chapter 4. Decomposing the (local) covariance matrix as $\sum_{i=1}^{D} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$, we obtain the orthogonal components as eigenvectors $\mathbf{u}_i$ and their corresponding eigenvalues λ_i (we rescale them, so that $\sum_{i=1}^{D} \lambda_i = 1$). The covariance matrix, in the 3-D data example, is written as

$$\begin{aligned}\sum_{i=1}^{3} \lambda_i \mathbf{u}_i \mathbf{u}_i^T =& \lambda_1 \mathbf{u}_1 \mathbf{u}_1^T - \lambda_2 \mathbf{u}_1 \mathbf{u}_1^T + \lambda_2 \mathbf{u}_1 \mathbf{u}_1^T + \lambda_2 \mathbf{u}_2 \mathbf{u}_2^T - \lambda_3 \mathbf{u}_1 \mathbf{u}_1^T \\ & - \lambda_3 \mathbf{u}_2 \mathbf{u}_2^T + \lambda_3 \mathbf{u}_1 \mathbf{u}_1^T + \lambda_3 \mathbf{u}_2 \mathbf{u}_2^T + \lambda_3 \mathbf{u}_3 \mathbf{u}_3^T \\ =& (\lambda_1 - \lambda_2) \mathbf{u}_1 \mathbf{u}_1^T + (\lambda_2 - \lambda_3)(\mathbf{u}_1 \mathbf{u}_1^T + \mathbf{u}_2 \mathbf{u}_2^T) \\ & + \lambda_3 (\mathbf{u}_1 \mathbf{u}_1^T + \mathbf{u}_2 \mathbf{u}_2^T + \mathbf{u}_3 \mathbf{u}_3^T).\end{aligned} \tag{5.34}$$

Note that $\mathbf{u}_1 \mathbf{u}_1^T$, $1/2(\mathbf{u}_1 \mathbf{u}_1^T + \mathbf{u}_2 \mathbf{u}_2^T)$ and $1/3(\mathbf{u}_1 \mathbf{u}_1^T + \mathbf{u}_2 \mathbf{u}_2^T + \mathbf{u}_3 \mathbf{u}_3^T)$ are the covariance matrices of the structures having intrinsic dimension $d = 1$, $d = 2$ and $d = 3$ respectively. The geometry of the 1-D, 2-D and 3-D structures and their eigenvectors and eigenvalues are illustrated in Table 5.1.

Table 5.1: Geometric features of manifolds in 3-D space.

Input	Eigenvectors	Eigenvalues	Saliency
		$\lambda_1 = 1$ $\lambda_2 = 0$ $\lambda_3 = 0$	$S_1 = \lambda_1 - \lambda_2 = 1$ $S_2 = 2(\lambda_2 - \lambda_3) = 0$ $S_3 = 3\lambda_3 = 0$
		$\lambda_1 = 0.5$ $\lambda_2 = 0.5$ $\lambda_3 = 0$	$S_1 = \lambda_1 - \lambda_2 = 0$ $S_2 = 2(\lambda_2 - \lambda_3) = 1$ $S_3 = 3\lambda_3 = 0$
		$\lambda_1 = 0.33$ $\lambda_2 = 0.33$ $\lambda_3 = 0.33$	$S_1 = \lambda_1 - \lambda_2 = 0$ $S_2 = 2(\lambda_2 - \lambda_3) = 0$ $S_3 = 3\lambda_3 = 1$

Using the saliences of these structures computed as $S_1 = \lambda_1 - \lambda_2$, $S_2 = 2(\lambda_2 - \lambda_3)$ and $S_3 = 3\lambda_3$ (Note that $S_1 + S_2 + S_3 = 1$), Eq. (5.34) can be rewritten as follows:

$$\sum_{i=1}^{3} \lambda_i \mathbf{u}_i \mathbf{u}_i^T = S_1 \mathbf{u}_1 \mathbf{u}_1^T + \frac{1}{2} S_2 (\mathbf{u}_1 \mathbf{u}_1^T + \mathbf{u}_2 \mathbf{u}_2^T) + \frac{1}{3} S_3 (\mathbf{u}_1 \mathbf{u}_1^T + \mathbf{u}_2 \mathbf{u}_2^T + \mathbf{u}_3 \mathbf{u}_3^T).$$

Then, the intrinsic dimension of the point is chosen as

$$d = \arg\max_i S_i.$$

The intrinsic dimension estimator we described is a variation of the approach in [108]. Compared to the estimator proposed in [108], our method is more intuitive and simpler. This is because we estimate the intrinsic dimensionality on the tangent space instead of the normal space and do not have the voting process.

Filtering the dataset according to the result of the presented intrinsic dimension estimator, the whole set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ is replaced by several subsets $\mathcal{D}^1 \cup \ldots \mathcal{D}^d \ldots \cup \mathcal{D}^D$ with d indicating the intrinsic dimension of the subset. Figure 5.3 illustrates the subsets obtained from the multiple manifolds dataset in Figure 5.1.

5.2.3.2 Step 2: Multiple Manifolds Learning Algorithm

After partitioning the entire dataset into several subsets, each subset $\mathcal{D}^d$ ($d \in \{1, \ldots D-1\}$) contains the data points generated from manifolds of dimension d. Here, we use

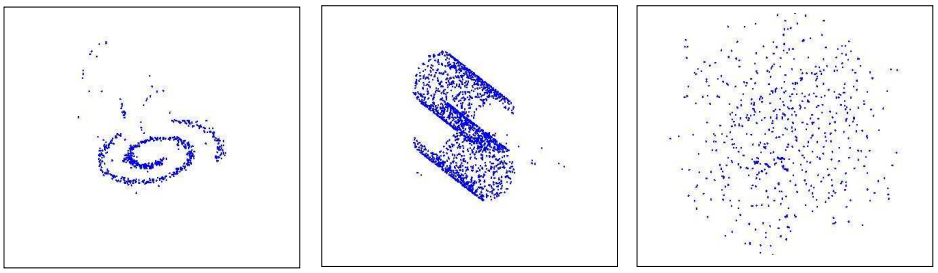


Figure 5.3: Three intrinsic-dimension-filtered subsets of data from Figure 5.1. (The value of the parameter used here is 0.0008, see Section 5.3.1.)

an iterative algorithm to discover and learn all the manifolds embedded in each subset $\mathcal{D}^d$ ($d \in \{1, \ldots D-1\}$).

In the iterative algorithm, we use one latent and data space graphs pair to represent one manifold. The latent/data space graphs are grown from their "seeds" (i.e. origins) along the local manifold patches, which is the same as the manifold learning algorithm described in the last chapter. Points in the data space which are not connected in the same manifold will be left unvisited. With other "seeds" for new latent and data space graphs, we explore the unvisited set $\tilde{\mathcal{D}}^d$ (entire filtered set $\tilde{\mathcal{D}}^d$ at the beginning) and obtain a new pair of latent/data space graphs for the other manifolds until all the underlying manifolds are detected.

When planting the "seeds" of the graphs, we need to ensure that the "seeds" are planted on the manifold and not among the outliers. If $\mathbf{x}_i$ is on the manifold, then most of its neighbors in the whole dataset (we use $\mathcal{K}(\mathbf{x}_i, \mathcal{D})$ to represent its K nearest neighbors in dataset $\mathcal{D}$) should appear in the unvisited filtered dataset $\tilde{\mathcal{D}}^d$. Therefore, we use the number of overlapped data points in both $\mathcal{K}(\mathbf{x}_i, \mathcal{D})$ and $\mathcal{K}(\mathbf{x}_i, \tilde{\mathcal{D}}^d)$ to determine whether $\mathbf{x}_i$ is on the manifold or not.

With the learnt latent/data space graphs, we next initialize the GTM model. Let $\mathcal{X}$ be the vertices in the data space graph $\mathcal{G}^m$, we use $\mathcal{K}(\mathcal{X}, \tilde{\mathcal{D}}^d)$ to denote the K nearest neighbors of the vertices in data space graph $\mathcal{X}$ in dataset $\tilde{\mathcal{D}}^d$ and $\tilde{\mathcal{D}}^d \backslash \mathcal{K}(\mathcal{X}, \tilde{\mathcal{D}}^d)$ to represent the rest of data points in the set $\tilde{\mathcal{D}}^d$. The implementation of the iterative algorithm is summarized as follows:

1. Initialize the unvisited data $\tilde{\mathcal{D}}^d$ as the filtered dataset $\mathcal{D}^d$: $\tilde{\mathcal{D}}^d = \mathcal{D}^d (d \in \{1, \ldots D-1\})$;

2. Set the number of manifolds in filtered dataset $\mathcal{D}^d$: $m = 0$;

3. While the unvisited dataset $\tilde{\mathcal{D}}^d$ has enough data points: $\tilde{\mathcal{D}}^d > K$ (K is the number of nearest neighbors)

   (a) Learn latent space graph $\mathcal{G}$ and data space graph $\mathcal{G}^m$ from $\tilde{\mathcal{D}}^d$; and set $m = m + 1$.

   (b) Initialize the parameters of the GTM model ($\mathbf{W}^{d,m}$, $\beta^{d,m}$ in $p(\mathbf{x}|d, m)$) from the graphs $\mathcal{G}$ and $\mathcal{G}^m$.

   (c) Remove the visited points (neighbors of the data space graph in the dataset) from the unvisited dataset $\tilde{\mathcal{D}}^d$: $\tilde{\mathcal{D}}^d = \tilde{\mathcal{D}}^d \backslash \mathcal{K}(\mathcal{X}, \tilde{\mathcal{D}}^d)$.

By this iterative algorithm, we can initialize all the submodels (GTM models) aligned to the non-linear manifolds obtained.

5.3 Experiments

First, we use some synthetic datasets (mix of data points generated from different low dimensional structures) to evaluate the performance of the proposed intrinsic dimension estimations. Then, taking two examples of synthetic 3-D datasets, we iteratively learn the embedded manifolds on the filtered subsets, construct the mixture model for these synthetic datasets and illustrate the training results. After that, we describe one application on astronomical simulation datasets to demonstrate that our proposed method is capable of detecting and learning the underlying low dimensional structures.

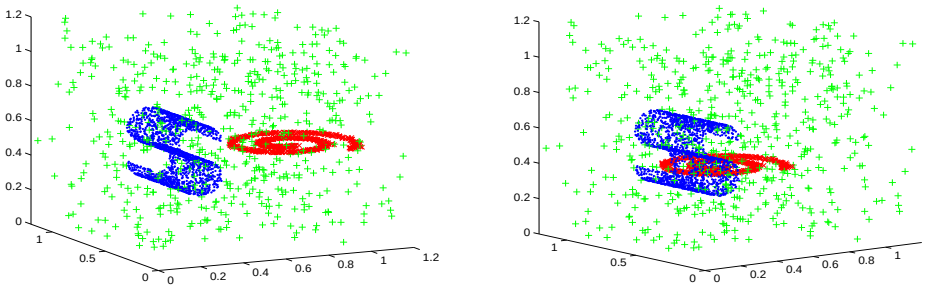
5.3.1 Intrinsic dimension estimation

In this part, we investigate the performance of our proposed intrinsic dimension estimator on two 3-D synthetic datasets, two 4-D synthetic datasets and two 10-D synthetic datasets respectively.

- 3-D datasets:

  The low dimensional structures used to form the 3-D synthetic datasets are one 1-D structure: a spiral and one 2-D structure: an S-curve (as shown in Figure 5.1).

This 3-D dataset consists of 700 3-D data points generated from the 1-D spiral structure, 2000 3-D data points generated from 2-D S-curve structure and a cloud consisting of 600 3-D noise data points. We illustrate the first 3-D synthetic dataset in Figure 5.4 (a). A more complicated situation is illustrated in Figure 5.4 (b), where the 1-D and 2-D structures are intersected together in some area.



(a) Synthetic dataset 1 (b) Synthetic dataset 2

Figure 5.4: Two 3-D synthetic datasets: (a) the 1-D manifold (spiral) and the 2-D manifold (S-curve) embedded are well separated. (b) the 1-D manifold (spiral) and 2-D manifold (S-curve) embedded are intersected.

- 4-D datasets:

  Then, we construct two 4-D synthetic datasets by mixing the 4-D data points generated from the same 1-D and 2-D manifolds used in the 3-D datasets, one 3-D manifold (cube) and 2000 noisy data uniformly distributed in the 4-D space and compute the correct intrinsic dimension estimation rate.

- 10-D datasets:

  Finally, we construct two 10-D synthetic datasets by mixing the 10-D data points generated from the same 1-D, 2-D, 3-D manifolds and 10000 noisy data points.

We use the proposed intrinsic dimension estimator to calculate the intrinsic dimension of datasets at the point level, filter the data points into subsets of varying intrinsic dimension and the correct rate of the filtering as follows:

$$R = \frac{n}{N}, \tag{5.35}$$

where N represents the size of the dataset and n is the number of the data points which are filtered correctly. By employing different parameter setting (the kernel size r in Eq. (4.23)), we illustrate the correct estimation rates in Figure 5.5, where R represents the correct rate on the dataset where the 1-D and 2-D manifolds are separated and R' represents the correct rate on the dataset where the 1-D and 2-D manifolds are intersected.

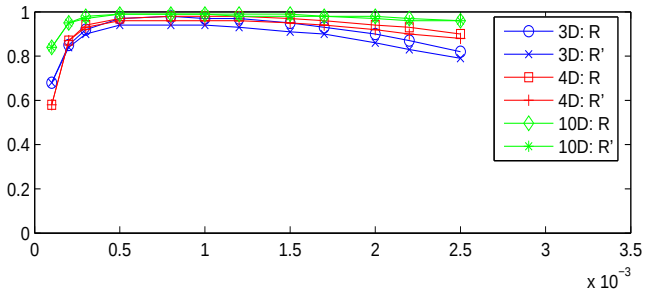


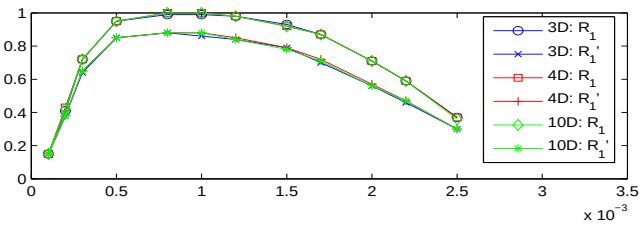
Figure 5.5: The correct rates of filtering data points in the 3-D, 4-D and 10-D synthetic datasets into the subsets of different intrinsic dimensionalities. The x-axis represents the width of the weighting kernel used in the intrinsic dimension estimation and the y-axis is the correct rate.

We also compute the correct rates of filtering the data points generated from the d dimensional structure (represented by R_d) for these synthetic datasets. The correct rate is computed as follows:

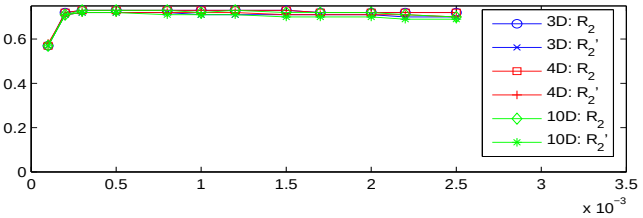
$$R_d = \frac{n_d}{N_d}, \tag{5.36}$$

where N_d represents the number of data points generated from d dimensional structure, n_d denotes the number of the data points be correctly identified as data points generated from d dimensional structure. When $d = D$, R_d is the correct rate of filtering the noise. R'_d is used to distinguish the correct rate of filtering the dataset where the manifolds are intersected.

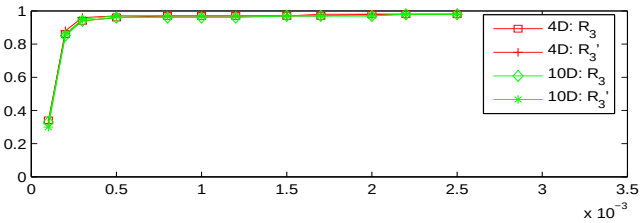
The correct rates of identifying the data points generated from the 1-D manifold, 2-D manifold, 3-D manifold (in 4-D and 10-D datasets) and noise data by using different parameter settings (the kernel size r) are shown in Figure 5.6 (a), (b), (c) and (d) respectively. From the results, it can be seen that the proposed intrinsic dimension estimator is robust to the setting of the kernel size r.



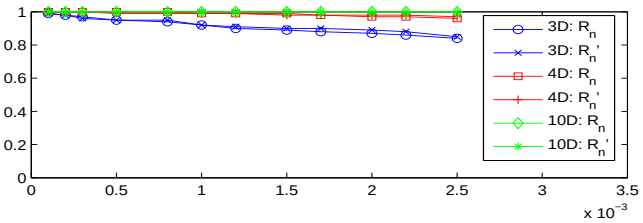
(a) Correct rates of filtering the data points generated from 1-D manifold



(b) Correct rates of filtering the data points generated from 2-D manifold



(c) Correct rates of filtering the data points generated from 3-D manifold



(d) Correct rates of filtering the noise data points

Figure 5.6: The correct rates of identifying the data points generated from the 1-D manifold (a), 2-D manifold (b), 3-D manifold (c) (in 4-D and 10-D datasets) and noise data (d) by using different parameter settings (the kernel size r)

5.3.2 Multiple manifolds with Varying Dimensions

With the two 3-D synthetic datasets shown in Figure 5.4, we illustrate the process of constructing the density model in this subsection.

First, we use the optimal parameter $r = 0.0008$ (the correct rate of the intrinsic dimension estimator is shown in the previous subsection) to estimate the intrinsic dimensionality of the data points shown in Figure 5.4 (a). Clustering the whole dataset (shown as the first level of the hierarchy in Figure 5.7) according to the estimated intrinsic dimension, we obtain the filtered subsets of the data points and put them at the second level of the hierarchical structure. From each subset, we use the proposed iterative algorithm to detect and learn the embedded manifolds. The manifolds obtained are shown in the lower level of hierarchy. These identified manifolds are used to initialize the GTM submodels of the mixture model, alongside the EM algorithm to fit the full mixture model to the dataset. We show the result at the bottom of the Figure 5.7.

Figure 5.8 illustrates a similar hierarchy learnt from the 3-D dataset shown in Figure 5.4 (b). The parameter selected for the intrinsic dimension estimator is also $r = 0.0008$. Because the manifolds are intersected, there is a gap splitting the single 1-D manifold into two parts as shown in the first plot of the second hierarchy. The points in the gap were taken to the set of intrinsically 2-D points. Given a strong prior knowledge concerning the connectiveness of the data manifolds, we could deal with such situations, however in the absence of such information we would prefer to have several isolated components dictated by the data. Therefore, we learn two 1-D manifolds from the 1-D filtered dataset. We present the learnt 1-D and 2-D manifolds on the third level in Figure 5.8 and show the learning result from the EM optimization process on the fourth level in the same figure.

As shown in these results, our proposed mixture model correctly represents the manifolds embedded in the datasets.

5.3.3 Identifying Streams and Shells in Disrupted Galaxies

Using one realistic setting of the particle model [81, 82] for satellite galaxy disruption by M31, we obtained several stages of the disrupted satellite, modeled by approximately 30,000 particles (stars). In each stage we applied the multiple manifolds

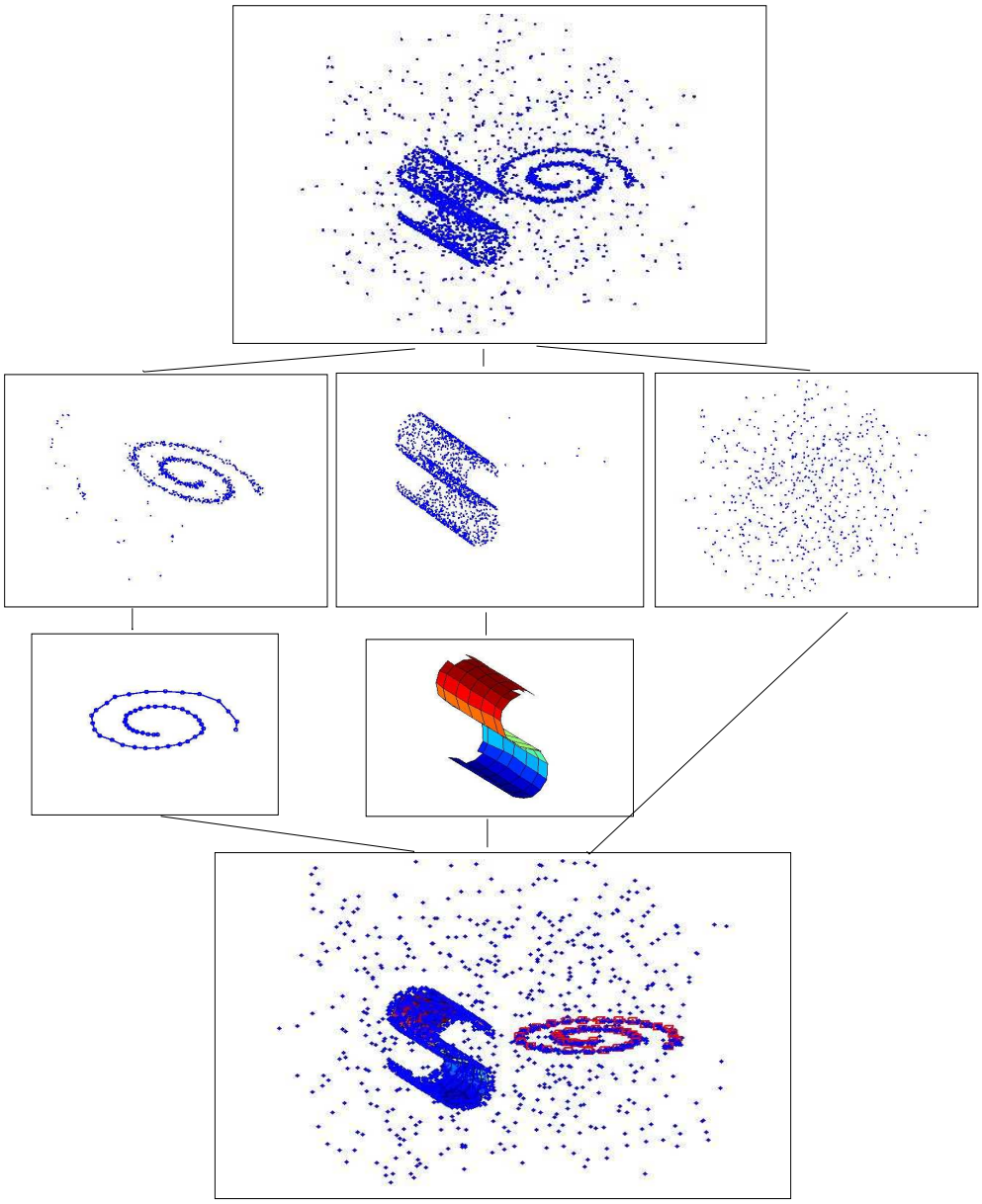


Figure 5.7: Multiple manifolds aligned density model built on the 3-D dataset shown in Figure 5.4 (a).

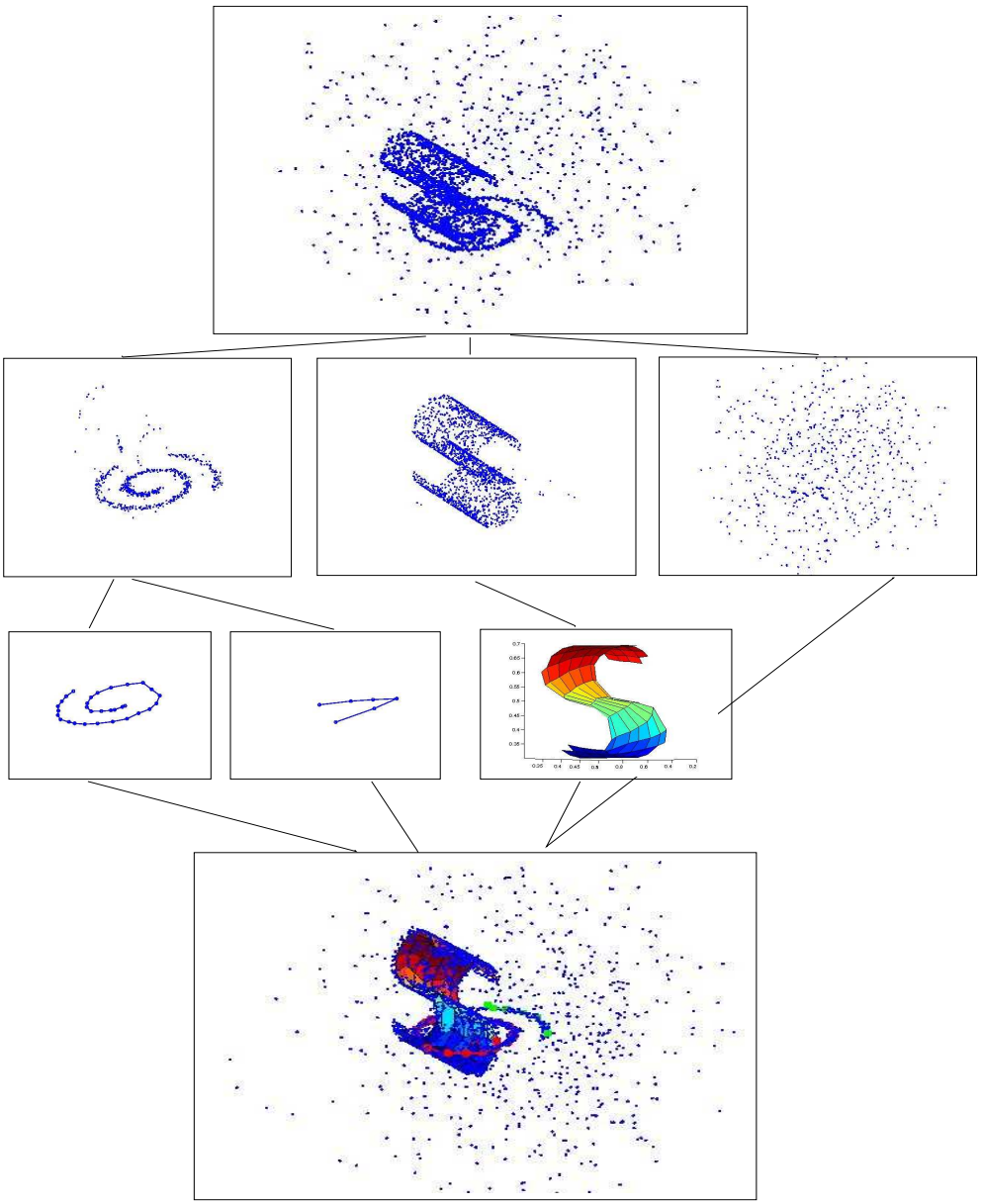
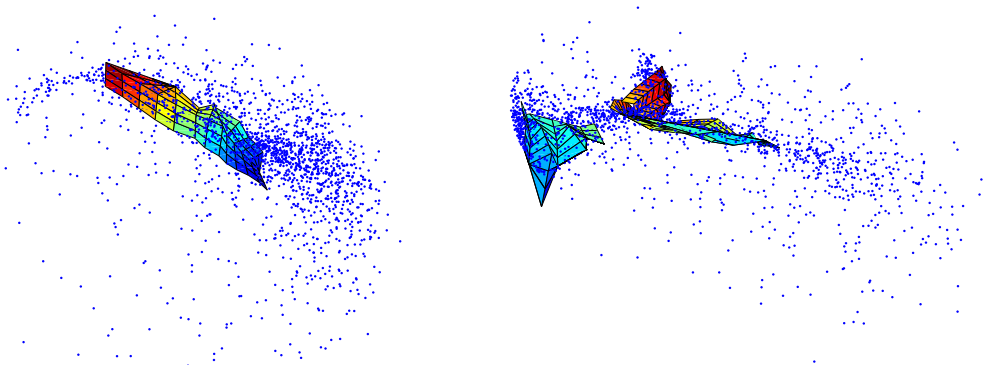


Figure 5.8: Multiple manifolds aligned density model built on the 3-D dataset shown in Figure 5.4 (b).

learning. In Figure 5.9 we show (along with the stars (particles)) the detected two-dimensional manifold structures ("skeletons" of the mixture components) in an early and a later stage of the disruption process. 1 and 3-dimensional structures are not shown. In the early stage a single stream was automatically detected, whereas in the later stage a stream and two shells were correctly identified. Two-dimensional structures are of most importance in such investigations, but our system can be used for investigation of structures across a variety of dimensions. It can also be used to build a hierarchical mixture model for the full system (large galaxy + a satellite) for the purposes of principled comparison of the simulated system with the real observations (in the projected plane). This is a matter for our future work.



(a) Disrupted satellite galaxy at early stage (b) Disrupted satellite galaxy at later stage

Figure 5.9: Identified 2-D manifolds in a disrupted satellite galaxy at an early (a) and later (b) stages of disruption by M31.

5.4 Summary

We represent the multiple manifolds aligned density distribution by a mixture model. To overcome the local minima problem of the EM algorithm based optimization, we proposed a novel hierarchical mixture model based framework to learn multiple manifolds of possibly different intrinsic dimensionalities. We first filter the data points with respect to the intrinsic dimensionality of the manifold patches they lie on. Then our new multiple manifolds learning algorithm is applied to each such filtered dataset of dimensionality d to detect d-dimensional manifolds along which the data are aligned. This is later used to initialize generative latent variable models representing noisy

manifolds underlying the data set. The generative models are combined in a hierarchical mixture representing the full data density. The proposed approach is significantly different from the current manifold learning approaches which typically assume that the whole data set is sampled from a single low dimensional manifold, which may not always be realistic.

As with other manifold learning approaches, parameter selection (e.g. neighborhood size) can be an issue. Model selection approaches can be used to select the appropriate values for a given application, but obviously much more work is required in this direction. In this work we present a proof of concept and show that our multiple manifolds learning framework can be potentially applied in interesting application domains, such as astronomy.

Chapter 6

Conclusions

6.1 Summary of the thesis

Kernel density estimator as a classical non-parametric density estimator has been proved successfully when applied to many practical problems. It requires no prior assumption about the density distribution form and no complex parameter learning process for exploring data. But its computational cost is a big issue, especially when the dataset is very large. After a review of several computational cost reduction strategies proposed in the literature for the classical kernel density estimator, we propose an efficient and sparse density estimation: Fast Parzen Windows (FPW). It could be viewed as a "sloppy Gaussian mixture model". On the one hand, the proposed density estimator (FPW) does not involve any "global" optimization process, which is usually very time consuming, but keeps the parameter learning process simple by estimating the Gaussian components locally. On the other hand, since our FPW components are designed to align along the local low structure of high dimensional data points, with carefully selected neighbor size, the local distribution could be captured by the local component accurately. Preliminary theoretical work also shows that the performance of the local component is relating to unknown true density, the Gaussian form and the local size. Experiments on both synthetic and real galaxy simulation datasets demonstrate that our proposed density estimator could provide a comparable density estimation accuracy, especially when the dataset in high dimensional space is generated from some low dimensional structure up to some noise. A significant improvement on the computational efficiency of our method over other non-parametric and semi-parametric density estimators is also observed.

An significantly improved algorithm efficiency makes applying the density estimator to astronomical research possible. A principled calibration of galaxy disruption simulations based on the density estimation is introduced in Chapter 3. Verified by a series of pseudo-observation datasets about the disrupted galaxy, the methodology based on our efficient and sparse density estimator shows a much more reliable calibration correct rate over the classical chi-square test, which is widely used in the Astronomy.

The other line of our research work is estimating the manifold aligned density through explicit manifold modeling. It's with the the assumption that the data with high dimensionality could be represented by a set of variables with lower dimensionality after some linear or non-linear transformation. Many algorithms were proposed to obtain an explicit manifold form. Of these, generative model based manifold learning algorithm is more attractive because it not only makes the manifold learning process generative to new observation but also provides the possibility to represent more than one manifold in a single density model. We developed a new manifold learning algorithm. It explores and learns the manifold by crawling over the connected manifold along its low dimensional structure. The learnt manifold is represented by graphs that could initialize the manifold modeling algorithm: Generative topographic mapping (GTM) easily. Compared with the classical GTM algorithm initialization, which is based on linear PCA, our proposed manifold learning algorithm could captures the underlying non-linear manifold correctly.

In contrast to the general assumption (that is the data points observed in one set is generated from a single low dimensional structure) in manifold learning algorithms, we proposed a learning framework in a scenario that the observed data points are actually lying along several different manifolds with different dimensionalities. Using a single GTM model to represent an explicit manifold embedded, the overall density estimated along multiple manifolds is a hierarchical mixture model of these GTMs. To initialize these manifold associated GTMs accurately, we utilized the following two steps: the first step is filtering the datasets according to the estimated intrinsic dimensionality of each data point. Then, we iteratively detect and learn the embedded manifolds for each filtered subset by employing the manifold learning approach proposed for initializing a single GTM. The effectiveness of the presented hierarchical mixture model is verified on synthetic datasets consisting of data points generated from more

than one manifold. We also demonstrate that the more than one 2-D structure could be detected and learnt from the galaxy disruption simulations by using our proposed framework.

6.2 Future work

In this section, we briefly highlight the possible future work of this thesis. One part of the work is about continuing the theoretical analysis work of our proposed efficient and sparse density estimator: Fast Parzen Windows. The FPW method employs full covariance Gaussian components to capture the local density distribution and represents the overall density by the mixture of these Gaussian component. Preliminary theoretical work presented in the thesis shows that the main contributed component in FPW has the similar properties of the locally parametric density estimator [1].

To continue the theoretical analysis, we would like to investigate the contribution of other components in the FPW. Therefore, we could obtain the accuracy of the whole FPW algorithm instead of only a single component.

Further work about the multiple manifolds aligned density estimation could be carried out along two different lines. First, work about proving the algorithm convergence, improving the computational efficiency have not been touched yet.

The other line is about the application of our proposed multiple manifold density estimation. In both Chapter 4 and Chapter 5, we showed the application of our manifold(s) aligned manifold density estimation on estimating the density of disrupted galaxy with explicit 2-D manifolds. However, as shown in Chapter 4, due to lack of the the prior knowledge about the distribution of the "shell" data points in the vicinity of the main galaxy M31, it is difficult to improve the learnt the manifold through fitting the GTM model to the observations. Similar problem appeared in the experiments of Chapter 5. Our proposed initializing framework for hierarchical mixture model is capable of detecting and learning more than one interesting 2-D structure formed by the data points, but a further refining the density learning process is required. Accordingly, we proposed the following future work along this line:

- With closer collaboration with astronomers, we will develop a better model fitting process by taking account of the prior knowledge about the galaxy density distribution observed by the astronomers. Consequently, the manifolds learnt

from the hierarchical mixture model could represent the low dimensional structures more accurately.

- Then, we could use our multiple manifolds density estimator as a tool to automatically detect and learn the 2-D manifolds in batches of different disrupted galaxy simulations. More than that, we expect that different types of 2-D manifolds could be separated automatically as well.

.1 Equation derivation of local parametric density esitmator

It was shown [1] that maximizing the local likelihood function (2.22) is equivalent to solving

$$\begin{aligned} V_n(x,\theta) &= \int K_h(t-x)v(x,t,\theta)\{dF_n(t) - f(t,\theta)dt\} \\ &= \frac{1}{N}\sum_{i=1}^{N} K_h(x_i - x)v(x,x_i,\theta) \\ &\quad - \int K_h(t-x)v(x,t,\theta)f(t,\theta)dt = 0 \end{aligned} \tag{1}$$

with the weight function

$$v(x,t,\theta) = u(t,\theta) = \frac{\partial \log f(t,\theta)}{\partial \theta}.$$

Considering the local model with parameters a, μ and δ

$$f(t;a,\mu,\delta) = \frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2})\right\}$$

for f around x. Accordingly, the weight functions are $\frac{\partial \log f(t;a,\mu,\delta)}{\partial a}$, $\frac{\partial \log f(t;a,\mu,\delta)}{\partial \mu}$ and $\frac{\partial \log f(t;a,\mu,\delta)}{\partial \delta}$. Hence, there are three equations to solve:

$$\begin{aligned} &\frac{1}{N}\sum_{i=1}^{N} K_h(x_i - x)\begin{pmatrix} \frac{1}{a} \\ \frac{(x_i-\mu)}{\delta^2} \\ \frac{1}{\delta}\{\frac{(x_i-\mu)^2}{\delta^2} - 1\} \end{pmatrix} \\ &\quad = \int K_h(t-x)\begin{pmatrix} \frac{1}{a} \\ \frac{(t-\mu)}{\delta^2} \\ \frac{1}{\delta}\{\frac{(t-\mu)^2}{\delta^2} - 1\} \end{pmatrix} \frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\} dt. \end{aligned} \tag{2}$$

The first equation in Eq. (2) is

$$\frac{1}{Na}\sum_{i=1}^{N} K_h(x_i - x) = \frac{1}{\sqrt{2\pi}a}\int K_h(t-x)\frac{a}{\delta}\exp\{-\frac{(t-\mu)^2}{2\delta^2}\}dt. \tag{3}$$

If we remove the same constant $\frac{1}{a}$ from both sides of the equation, the left hand side of the new equation is actually $\tilde{f}(x)$. With $K_h(t-x) = \frac{1}{\sqrt{2\pi}h}\exp\left\{-\frac{(t-x)^2}{2h^2}\right\}$, the right

hand side term can be written as follows:

$$\begin{aligned}
&\int K_h(t-x)\frac{a}{\sqrt{2\pi}\delta}\exp\{-\frac{(t-\mu)^2}{2\delta^2}\}dt \\
=&\frac{a}{2\pi h\delta}\int \exp\{-\frac{(t-x)^2}{2h^2}\}\exp\{-\frac{(t-\mu)^2}{2\delta^2}\}dt \\
=&\frac{a}{2\pi h\delta}\int \exp\{-\frac{(t-x)^2\delta^2+(t-\mu)^2h^2}{2h^2\delta^2}\}dt \\
=&\frac{a}{2\pi h\delta}\int \exp\{E(t)\}dt,
\end{aligned} \tag{4}$$

where $E(t)$ is defined as:

$$\begin{aligned}
E(t) =& -\frac{(t-x)^2\delta^2+(t-\mu)^2h^2}{2h^2\delta^2} \\
=& -\frac{(\delta^2+h^2)t^2-2(x\delta^2+h^2\mu)t+h^2\mu^2+\delta^2x^2}{2h^2\delta^2} \\
=& -\frac{\delta^2+h^2}{2h^2\delta^2}\left(t^2-\frac{2\delta^2x+2h^2\mu}{\delta^2+h^2}t\right)+\frac{\delta^2x^2+h^2\mu^2}{2\delta^2h^2} \\
=& -\frac{\delta^2+h^2}{2h^2\delta^2}\left(t-\frac{\delta^2x+2h^2\mu}{\delta^2+h^2}\right)^2+\frac{(\delta^2x+h^2\mu)^2}{(2h^2\delta^2)(\delta^2+h^2)}-\frac{\delta^2x^2+h^2\mu^2}{2\delta^2h^2} \\
=& -\frac{\delta^2+h^2}{2h^2\delta^2}\left(t-\frac{\delta^2x+2h^2\mu}{\delta^2+h^2}\right)^2+\frac{-(x-\mu)^2}{2(\delta^2+h^2)}.
\end{aligned} \tag{5}$$

Then, replacing $E(t)$ in (4) with Eq. (5), we obtain:

$$\begin{aligned}
\frac{a}{2\pi h\delta}\int \exp\{E(t)\}dt =& -\frac{a}{2\pi h\delta}\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\} \\
&\int \exp\left\{-\frac{\delta^2+h^2}{2h^2\delta^2}\left(t-\frac{\delta^2x+2h^2\mu}{\delta^2+h^2}\right)^2\right\}dt \\
=&\frac{a}{2\pi h\delta}\sqrt{\frac{\pi 2h^2\delta^2}{\delta^2+h^2}}\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\} \\
=&\frac{a}{\sqrt{2\pi}(\delta^2+h^2)^{1/2}}\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\}.
\end{aligned} \tag{6}$$

Therefore, the first equation in (2) becomes:

$$\tilde{f}(x) = \frac{a}{\sqrt{2\pi}(\delta^2+h^2)^{1/2}}\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\}. \tag{7}$$

Then we consider the second equation in (2):

$$\begin{aligned}&\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)\frac{(x_i-\mu)}{\delta^2}\\&\quad=\int K_h(t-x)\frac{(t-\mu)}{\delta^2}\frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt\end{aligned}\tag{8}$$

Similarly, by removing the same term $\frac{1}{\delta^2}$ from the both sides, the equation reads:

$$\begin{aligned}&\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-\mu)\\&\quad=\int K_h(t-x)\frac{a(t-\mu)}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt,\end{aligned}\tag{9}$$

where the left hand side can be expanded as follows:

$$\begin{aligned}&\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-\mu)\\=&\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-x+x-\mu)\\=&\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-x)+\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x-\mu).\end{aligned}\tag{10}$$

Since the first derivative of $\tilde{f}(x)$ is

$$\tilde{f}'(x)=\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-x),$$

Eq. (10) could be written in terms of $\tilde{f}(x)$ as

$$h^2\tilde{f}'(x)+(x-\mu)\tilde{f}(x).$$

Also, we expand the right hand side into two terms:

$$\begin{aligned}
&\int K_h(t-x)\frac{a(t-\mu)}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt \\
&\quad=\int K_h(t-x)\frac{a(t-x+x-\mu)}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt \\
&\quad=\int K_h(t-x)\frac{a(t-x)}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt \\
&\qquad+\int K_h(t-x)\frac{a(x-\mu)}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt.
\end{aligned} \tag{11}$$

From (3), we know that

$$\tilde{f}(x)=\int K_h(t-x)\frac{a(x-\mu)}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt, \tag{12}$$

we could simplify (9) to be

$$h^2\tilde{f}'(x)=\int K_h(t-x)\frac{a(t-x)}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt \tag{13}$$

by removing two the equal terms

$$(x-\mu)\tilde{f}(x)$$

and

$$\int K_h(t-x)\frac{a(x-\mu)}{\sqrt{2\pi}\delta}\exp\{-\frac{(t-\mu)^2}{2\delta^2}\}dt$$

from (10) and (11)), respectively.

With the specific form of the smooth kernel $K_h(t-x)$, we rewrite (13) in the following form:

$$\begin{aligned}
h^2\tilde{f}'(x)=&\frac{a}{2\pi\delta h}\int\exp\left\{-\frac{(t-x)^2\delta^2+(t-\mu)^2h^2}{2\delta^2h^2}\right\}(t-x)dt \\
=&\frac{a}{2\pi\delta h}\int\exp\left\{E(t)\right\}(t-x)dt.
\end{aligned} \tag{14}$$

According to the new form of $E(t)$ derived in (5), Eq. (14) becomes:

$$\begin{aligned}
h^2\tilde{f}'(x)=&\frac{a}{2\pi\delta h}\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\} \\
&\int\exp\left\{-\frac{\delta^2+h^2}{2\delta^2h^2}\left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}\right)^2\right\}(t-x)dt.
\end{aligned} \tag{15}$$

We use a constant $\mathbf{C}$ to represent the constant term in (15), i.e.,

$$\mathbf{C} = \frac{a}{2\pi\delta h}\exp\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\}. \tag{16}$$

We continue simplify (15) as follows:

$$\begin{aligned} h^2\tilde{f}'(x) =& \mathbf{C}\int \exp\left\{-\frac{\delta^2+h^2}{2\delta^2h^2}(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2})^2\right\} \\ & \left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}+\frac{x\delta^2+\mu h^2}{\delta^2+h^2}-x\right)dt \\ =& \mathbf{C}\int \exp\left\{-\frac{\delta^2+h^2}{2\delta^2h^2}(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2})^2\right\}\left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}\right)dt \\ & + \mathbf{C}\int \exp\left\{-\frac{\delta^2+h^2}{2\delta^2h^2}(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2})^2\right\}\left(\frac{x\delta^2+\mu h^2}{\delta^2+h^2}-x\right)dt \\ =& -\mathbf{C}\sqrt{\frac{\pi 2\delta^2h^2}{\delta^2+h^2}}\frac{h^2(x-\mu)}{\delta^2+h^2}. \end{aligned} \tag{17}$$

Bringing the constant $\mathbf{C}$ (16) back to (17), we obtain the following equation from the second equation of (2):

$$\tilde{f}'(x) = -\frac{a(x-\mu)}{(\delta^2+h^2)^{3/2}}\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\}.$$

Finally, we analyze the third equation in (2):

$$\begin{aligned} & \frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)\frac{(x_i-\mu)^2}{\delta^2} \\ & \quad = \int K_h(t-x)\frac{a(t-\mu)^2}{\sqrt{2\pi}\delta^3}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt \end{aligned} \tag{18}$$

Removing $\frac{1}{\delta^2}$ from both sides of the equation, (18) is updated to be:

$$\begin{aligned} & \frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-\mu)^2 \\ & \quad = \int K_h(t-x)\frac{a(t-\mu)^2}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{2\delta^2}\right\}dt. \end{aligned} \tag{19}$$

Then, we expand the left-hand side of Eq. (19) in the following way:

$$\begin{aligned}
&\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-\mu)^2 \\
&\quad=\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-x+x-\mu)^2 \\
&\quad=\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-x)^2+\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x-\mu)^2 \\
&\qquad+2\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-x)(x-\mu).
\end{aligned} \tag{20}$$

Since the second derivative of $\tilde{f}(x)$ is

$$\tilde{f}''(x)=\frac{1}{h^4N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-x)^2-\frac{1}{h^2N}\sum_{i=1}^{N}K_h(x_i-x),$$

we obtain the representation of the first term in(20) in terms of $f''(x)$ and $f(x)$ as:

$$\begin{aligned}
\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x)(x_i-x)^2 =&h^4\tilde{f}''(x)+h^2\frac{1}{N}\sum_{i=1}^{N}K_h(x_i-x) \\
=&h^4\tilde{f}''(x)+h^2\tilde{f}(x).
\end{aligned} \tag{21}$$

Also, from (3) and (13), the second and third terms in the right side of (20) can be simplified as

$$2h^2(x-\mu)\tilde{f}'(x)+(x-\mu)^2\tilde{f}(x). \tag{22}$$

The right hand side of (19) can be derived and splitted into three terms:

$$\begin{aligned}
&\int K_h(t-x)(t-\mu)^2\frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{\delta^2}\right\}dt \\
&\quad=\int K_h(t-x)\frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{\delta^2}\right\}(t-x+x-\mu)^2dt \\
&\quad=\int K_h(t-x)\frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{\delta^2}\right\}(t-x)^2dt \\
&\qquad+2\int K_h(t-x)\frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{\delta^2}\right\}(t-x)(x-\mu)dt \\
&\qquad+\int K_h(t-x)\frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{\delta^2}\right\}(x-\mu)^2dt.
\end{aligned} \tag{23}$$

According to (3) and (13), the sum of the second and the third terms on the right side

of (20) represented as Eq. (22) is equal to the sum of the second and third terms in (23):

$$\begin{aligned} &2(x-\mu)\int K_h(t-x)\frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{\delta^2}\right\}(t-x)dt \\ &+(x-\mu)^2\int K_h(t-x)\frac{a}{\sqrt{2\pi}\delta}\exp\left\{-\frac{(t-\mu)^2}{\delta^2}\right\}dt. \end{aligned} \tag{24}$$

Therefore, we update Eq. (18) to be

$$h^4\tilde{f}''(x)+h^2\tilde{f}(x)=\frac{a}{\sqrt{2\pi}\delta}\int K_h(t-x)\exp\left\{-\frac{(t-\mu)^2}{\delta^2}\right\}(x-t)^2dt \tag{25}$$

by removing these equal terms.

Considering the specific format of the kernel $K_h(t-x)$ on right hand side of Eq. (25), we have:

$$h^4\tilde{f}''(x)+h^2\tilde{f}(x)=\frac{a}{2\pi\delta}\int\exp\left\{E(t)\right\}(x-t)^2\Big\}dt. \tag{26}$$

By replacing $E(t)$ with (5), it gives

$$h^4\tilde{f}''(x)+h^2\tilde{f}(x)=\mathbf{C}\int\exp\left\{-\frac{\delta^2+h^2}{2\delta^2h^2}\left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}\right)^2\right\}(t-x)^2dt, \tag{27}$$

where $\mathbf{C}$ is defined earlier.

Rewriting the right hand of Eq. (27), we obtain the following three terms:

$$\begin{aligned} &\mathbf{C}\int\exp\left\{-\frac{\delta^2+h^2}{2\delta^2h^2}\left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}\right)^2\right\}\left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}\right)^2dt \\ &2\mathbf{C}\int\exp\left\{-\frac{\delta^2+h^2}{2\delta^2h^2}\left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}\right)^2\right\}\left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}\right)\left(\frac{-h^2(x-\mu)}{\delta^2+h^2}\right)dt \\ &\mathbf{C}\int\exp\left\{-\frac{\delta^2+h^2}{2\delta^2h^2}\left(t-\frac{x\delta^2+\mu h^2}{\delta^2+h^2}\right)^2\right\}\left(\frac{h^2(x-\mu)}{\delta^2+h^2}\right)^2dt. \end{aligned} \tag{28}$$

Furthermore, we have

$$\begin{aligned} h^4\tilde{f}''(x)+h^2\tilde{f}(x)=&\mathbf{C}\left[\frac{1}{2}\sqrt{\frac{\pi(2\delta^2h^2)^3}{(h^2+\delta^2)^3}}+\left(\frac{h^2(x-\mu)}{\delta^2+h^2}\right)^2\sqrt{\frac{\pi(2\delta^2h^2)}{(h^2+\delta^2)}}\right] \\ =&\frac{a}{\sqrt{2\pi}}\left[\frac{\delta^2h^2}{(h^2+\delta^2)^{3/2}}+\frac{h^4(x-\mu)^2}{(h^2+\delta^2)^{5/2}}\right]\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\}. \end{aligned} \tag{29}$$

Therefore, the third function becomes:

$$h^2\tilde{f}(x) + h^4\tilde{f}''(x) = \frac{a}{\sqrt{2\pi}}\left[\frac{\delta^2 h^2}{(\delta^2+h^2)^{3/2}} + \frac{h^4(x-\mu)^2}{(\delta^2+h^2)^{5/2}}\right]\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\}.$$

From the (2), we obtain the following three equation:

$$\begin{aligned}
\tilde{f}(x) &= \frac{a}{\sqrt{2\pi}(h^2+\delta^2)^{1/2}}\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\} && (30)\\
\tilde{f}'(x) &= -\frac{a(x-\mu)}{\sqrt{2\pi}(\delta^2+h^2)^{3/2}}\exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\} && (31)\\
h^2\tilde{f}(x) + h^4\tilde{f}''(x) &= \frac{a}{\sqrt{2\pi}}\left[\frac{\delta^2 h^2}{(\delta^2+h^2)^{3/2}} + \frac{h^4(x-\mu)^2}{(\delta^2+h^2)^{5/2}}\right] \\
&\quad \exp\left\{-\frac{(x-\mu)^2}{2(\delta^2+h^2)}\right\}. && (32)
\end{aligned}$$

The estimated local model with the estimated parameters is

$$\hat{f}(x) = \hat{f}(x;\hat{a},\hat{\mu},\hat{\delta}) = \frac{\hat{a}}{\sqrt{2\pi}\hat{\delta}}\exp\left\{-\frac{(x-\hat{\mu})^2}{2\hat{\delta}^2}\right\}. \tag{33}$$

According to Eq. (30), we can represent the estimated $\hat{a}$ as

$$\hat{a} = \tilde{f}(x)\sqrt{h^2+\delta^2}\exp\left\{\frac{(x-\mu)^2}{2(h^2+\delta^2)}\right\}. \tag{34}$$

Combining (34) with (33), $\hat{f}(x)$ can be further written as:

$$\begin{aligned}
\hat{f}(x;\hat{a},\mu,\delta) =& \tilde{f}(x)\frac{\sqrt{h^2+\delta^2}}{\delta}\exp\left\{\frac{(x-\mu)^2}{2(h^2+\delta^2)}\right\}\exp\left\{-\frac{(x-\mu)^2}{2\delta^2}\right\} \\
=& \tilde{f}(x)\sqrt{\frac{h^2+\delta^2}{\delta^2}}\exp\left\{-\frac{(x-\mu)^2}{2(h^2+\delta^2)}\frac{h^2}{\delta^2}\right\} \\
=& \tilde{f}(x)\sqrt{\frac{h^2+\delta^2}{\delta^2}}\exp\left\{-\frac{(x-\mu)^2}{2(h^2+\delta^2)^2}\frac{h^2(h^2+\delta^2)}{\delta^2}\right\} \\
=& \tilde{f}(x)Q^{-1/2}\exp\left\{-\frac{(x-\mu)^2}{2(h^2+\delta^2)^2}Q^{-1}h^2\right\}
\end{aligned} \tag{35}$$

where

$$Q = \frac{\delta^2}{\delta^2+h^2}.$$

Dividing Eq. (30) by Eq. (31), we obtain

$$\frac{\tilde{f}'(x)}{\tilde{f}(x)} = -\frac{x-\mu}{h^2+\delta^2}. \tag{36}$$

Similarly, dividing (31) by (32), we obtain

$$\begin{aligned}\frac{h^2\tilde{f}(x)+h^4\tilde{f}''(x)}{\tilde{f}(x)} =& a\left[\frac{\delta^2h^2}{(\delta^2+h^2)^{3/2}}+\frac{h^4(x-\mu)^2}{(\delta^2+h^2)^{5/2}}\right]\frac{(h^2+\delta^2)^{1/2}}{a}\\ =&\left[\frac{\delta^2h^2}{(\delta^2+h^2)}+\frac{h^4(x-\mu)^2}{(\delta^2+h^2)^2}\right].\end{aligned} \tag{37}$$

With (36), Eq. (37) can be written as:

$$\frac{\tilde{f}''(x)}{\tilde{f}(x)} = \frac{(x-\mu)^2}{(\delta^2+h^2)^2} - \frac{1}{\delta^2+h^2}.$$

Therefore, Q can be represented in terms of $f(x)$, $f'(x)$ and $f''(x)$ as follows:

$$Q = \frac{\delta^2}{\delta^2+h^2} = 1-\frac{h^2}{\delta^2+h^2} = 1-h^2\left\{\left(\frac{\tilde{f}'(x)}{\tilde{f}(x)}\right)^2-\frac{\tilde{f}''(x)}{\tilde{f}(x)}\right\},$$

and we can write $\hat{f}(x)$ as

$$\begin{aligned}\hat{f}(x) =&\tilde{f}(x)Q^{-1/2}\exp\{-\frac{1}{2}(\frac{\tilde{f}(x')}{\tilde{f}(x)})^2Q^{-1}h^2\}\\ =&\tilde{f}(x)\sqrt{\frac{\tilde{f}^2(x)}{\tilde{f}^2(x)+h^2\tilde{f}''(x)\tilde{f}(x)-h^2\tilde{f}'^2(x)}}\\ &\exp\{-\frac{1}{2}\frac{h^2\tilde{f}'^2(x)}{\tilde{f}^2(x)+h^2\tilde{f}''(x)\tilde{f}(x)-h^2\tilde{f}'^2(x)}\}.\end{aligned} \tag{38}$$

.2 Equation derivations of Fast Parzen Windows

It is straightforward to get

$$\alpha_j = \tilde{f}(s_j). \tag{39}$$

For μ_j, we have

$$\begin{aligned}
\mu_j =& \frac{\sum_{i=1}^{N} K_h(x_i - s_j)(x_i - s_j) + \sum_{i=1}^{N} K_h(x_i - s_j)s_j}{\sum_{n=1}^{N} K_h(x_n - s_j)} \\
=& \frac{\sum_{i=1}^{N} K_h(x_i - s_j)(x_i - s_j)}{\sum_{n=1}^{N} K_h(x_n - s_j)} + s_j \\
=& \frac{h^2 \tilde{f}'(s_j)}{\tilde{f}(s_j)} + s_j.
\end{aligned} \tag{40}$$

Note that the last step in Eq. (40) used

$$f'(s_j) = \frac{1}{h^2} \sum_{i=1}^{N} K_h(x_i - s_j)(x_i - s_j). \tag{41}$$

As to δ_j^2, we expand it to

$$\begin{aligned}
\delta_j^2 =& \frac{\sum_{i=1}^{N} K_h(x_i - s_j)(x_i - s_j + s_j - \mu_j)^2}{\sum_{n=1}^{N} K_h(x_n - s_j)} \\
=& \frac{\sum_{i=1}^{N} K_h(x_i - s_j)[(x_i - s_j)^2 + 2(s_j - \mu_j)(x_i - s_j) + (s_j - \mu_j)^2]}{\sum_{n=1}^{N} K_h(x_n - s_j)} \\
=& \frac{\sum_{i=1}^{N} K_h(x_i - s_j)(x_i - s_j)^2}{N\tilde{f}(s_j)} + \frac{2(s_j - \mu_j)\sum_{i=1}^{N} K_h(x_i - s_j)(x_i - s_j)}{N\tilde{f}(s_j)} \\
&+ \frac{(s_j - \mu_j)^2 \tilde{f}(s_j)}{\tilde{f}(s_j)}.
\end{aligned} \tag{42}$$

The first term in (42) can be

$$\frac{\sum_{i=1}^{N} K_h(x_i - s_j)(x_i - s_j)^2}{N\tilde{f}(s_j)} = \frac{h^2 \tilde{f}(s_j) + h^4 \tilde{f}''(s_j)}{\tilde{f}(s_j)}, \tag{43}$$

where the second derivative of $\tilde{f}(s_j)$ is

$$\tilde{f}''(s_j) = \frac{1}{h^4 N} \sum_{i=1}^{N} K_h(x_i - s_j)(x_i - s_j)^2 - \frac{1}{h^2 N} \sum_{i=1}^{N} K_h(x_i - s_j). \tag{44}$$

Bringing (40) to the second and third terms in (42), we obtain

$$\begin{aligned}\frac{2(s_j-\mu_j)}{N\tilde{f}(s_j)}&\sum_{i=1}^{N}K_h(x_i-s_j)(x_i-s_j)\\&=-2\frac{h^2\tilde{f}'(s_j)}{\tilde{f}(s_j)}\frac{\sum_{i=1}^{N}K_h(x_i-s_j)(x_i-s_j)}{\tilde{f}(s_j)}\\&=-2\frac{h^2\tilde{f}'(s_j)}{\tilde{f}(s_j)}\frac{h^2\tilde{f}'(s_j)}{\tilde{f}(s_j)}\\&=-\frac{2h^4\tilde{f}'^2(s_j)}{\tilde{f}^2(s_j)}\end{aligned}\tag{45}$$

and

$$\begin{aligned}\frac{(s_j-\mu_j)^2\tilde{f}(s_j)}{\tilde{f}(s_j)}&=\left(-\frac{h^2\tilde{f}'(s_j)}{\tilde{f}(s_j)}\right)^2\frac{\tilde{f}(s_j)}{\tilde{f}(s_j)}\\&=\frac{h^4\tilde{f}'^2(s_j)}{\tilde{f}^2(s_j)}.\end{aligned}\tag{46}$$

Therefore, Eq. (42) is the sum of Eqs. (43), (45) and (46), i.e.,

$$\delta_j^2=\frac{h^2\tilde{f}^2(s_j)+h^4\tilde{f}''(s_j)\tilde{f}(s_j)-h^4\tilde{f}'^2(s_j)}{\tilde{f}^2(s_j)}.\tag{47}$$

Using the estimated parameters of jth component of FPW-S, α_j, μ_j and δ_j, the estimated jth compnonent reads

$$\begin{aligned}\hat{g}(s_j;\alpha_j,\mu_j,\delta_j)=&\frac{1}{\sqrt{2\pi}}\frac{f(\tilde{s}_j)}{\sum_{q=1}^{M}\tilde{f}(s_q)}\frac{1}{\delta}\exp\left\{-\frac{(\mu_j-s_j)^2}{2\delta^2}\right\}\\=&\frac{1}{\sqrt{2\pi}}\frac{f(\tilde{s}_j)}{\sum_{q=1}^{M}\tilde{f}(s_q)}\frac{1}{\delta}\exp\left\{-\frac{(\frac{h^2\tilde{f}'(s_j)}{\tilde{f}(s_j)})^2}{2\delta^2}\right\}\\=&\frac{1}{\sqrt{2\pi}}\frac{f(\tilde{s}_j)}{\sum_{q=1}^{M}\tilde{f}(s_q)}\sqrt{\frac{\tilde{f}^2(s_j)}{h^2\tilde{f}^2(s_j)+h^4\tilde{f}''(s_j)\tilde{f}(s_j)-h^4\tilde{f}'^2(s_j)}}\\&\exp\left\{-\frac{h^2\tilde{f}'^2(s_j)}{h^2\tilde{f}^2(s_j)+h^4\tilde{f}''(s_j)\tilde{f}(s_j)-h^4\tilde{f}'^2(s_j)}\right\}.\end{aligned}\tag{48}$$

References

[1] N. L. Hjort and M. C. Jones. Locally parametric nonparametric density estimation. *The Annals of Statistics*, 24(4):1619–1647, 1996.

[2] G. E. Hinton and T. J. Sejnowski. *Unsupervised Learning: Foundations of Neural Computation*. MIT Press, 1999.

[3] P. Vincent and Y. Bengio. Manifold Parzen windows. In *Advances in Neural Information Processing Systems 15*, pages 825–832, 2003.

[4] C. M. Bishop, M. Svensen, and C. K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998.

[5] B. W. Silverman. *Density Estimation*. Chapman and Hall, 1986.

[6] D. W. Scott. *Multivariate Density Estimation. Theory, Practice and Visualization*. Wiley, 1992.

[7] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[8] B. S. Everitt and D. J. Hand. *Finite Mixture Distributions*. Chapman and Hall, 1981.

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

[10] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[11] M. P. Wand and M. C. Jones. *Kernel smoothing*. CRC Press, 1995.

[12] D. W. Scott. On optimal and data-based histograms. *Biometrika*, 66(3):605–610, 1979.

[13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[14] E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[15] S. R. Sain and D. W. Scott. On locally adaptive density estimation. *Journal of the American Statistical Association*, 91(436):1525–1534, 1996.

[16] S. R. Sain. Multivariate locally adaptive density estimation. *Journal of the American Statistical Association*, 91:1525–1534, 1999.

[17] L. Greengard and J. Strain. The fast Gauss transform. *SIAM Journal of Scientific Statistical Computing*, 12(1):79–94, 1991.

[18] L. Greengard and X. Sun. A new version of the fast Gauss transform. *Documenta Mathematica*, ICM(1):575–584, 1998.

[19] B. J. C. Baxter and G. Roussos. A new error estimate of the fast Gauss transform. *SIAM Journal on Scientific Computing*, 24(1):257–259, 2002.

[20] C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis. Improved fast Gauss transform and efficient kernel density estimation. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 464, Washington, DC, USA, 2003. IEEE Computer Society.

[21] A. G. Gray. Nonparametric density estimation: toward computational tractability. In *SIAM International Conference on Data Mining*, 2003.

[22] J. M. Kubica, J. Masiero, A. Moore, R. Jedicke, and A. J. Connolly. Variable KD-tree algorithms for spatial pattern search and discovery. *Neural Information Processing Systems*, 2005.

[23] Andrew Moore Alexander Gray. N-body problems in statistical learning. In Todd K. Leen and Thomas G. Dietterich, editors, *Advances in Neural Information Processing Systems 13*. MIT Press, 2001.

[24] V.C. Raykar, C. Yang, R. Duraiswami, and N.A. Gumerov. Fast computation of sums of Gaussians in high dimensions. Technical report, UMIACS TR 2005-69, Department of Computer Science, University of Maryland, College Park, 2005.

[25] I. Murray. Gaussian processes and fast matrix-vector multiplies, 2009. Presented at the Numerical Mathematics in Machine Learning workshop at the 26th International Conference on Machine Learning (ICML 2009), Montreal, Canada.

[26] Z. Zhang and H. Zha. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM Journal on Scientific Computing*, 26(1):313–338, 2005.

[27] J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23(9):881–890, 1974.

[28] P. J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

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

[30] J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications. Beverly Hills. CA, 1977.

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

[32] T. Kohonen, editor. *Self-Organizing Maps*. Springer, 1995. Berlin/Heidelberg.

[33] J. B. Tenenbaum, V. D. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[34] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[35] L.J.P. van der Maaten, E.O. Postma, and H.J. van den Herik. Dimensionality reduction: A comparative review. Technical report, Tilburg University, 2009. TiCC-TR 2009-005.

[36] B. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*, 26:17–32, Feb 1981.

[37] B. Kegl, A. Krzyzak, T. Linder, and K. Zeger. Principal curves: learning and convergence. In *Proceedings of the IEEE International Symposium on Information Theory*, page 387, 1998.

[38] K.-R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12:181–201, 2001.

[39] A.J. Smola B. Scholkopf and K. R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[40] S. Mika, B. Schölkopf, A. J. Smola, K. R. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de–noising in feature spaces. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*. MIT Press, 1999.

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

[42] A. Ultsch. Self-organizing neural networks for visualization and classification. In O. Opitz, B. Lausen, and R. Klar, editors, *Information and Classification*, pages 307–313, 1993.

[43] H. Yin. ViSOM- a novel method for multivariate data projection and structure visualization. *IEEE Transactions on Neural Networks*, 13(1):237–243, 2002.

[44] J. Vesanto. SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126, 1999.

[45] L. K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2003.

[46] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, June 2003.

[47] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceeding of National Academy of Science United States of America*, 100(10):5591–5596, 2003.

[48] X. He and P. Niyogi. Locality preserving projections. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[49] V. D. Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *In Advances in Neural Information Processing Systems 15*, volume 15, pages 705–712, 2003.

[50] M. Balasubramanian and E.L. Schwartz. The Isomap algorithm and topological stability. *Science*, 295(5552):7, 2002.

[51] H. Zha and Z. Zhang. Isometric embedding and continuum ISOMAP. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 864–871, 2003.

[52] C. M. Bishop. Latent variable models. *Learning in Graphical Models*, pages 371–403, 1999.

[53] M. Tipping and C. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

[54] M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999.

[55] S. Roweis, L. K. Saul, and G. E. Hinton. Global coordination of local linear models. In *Advances in Neural Information Processing Systems 14*, pages 889–896. MIT Press, 2002.

[56] J. J. Verbeek, N. Vlassis, and B. Krose. Coordinating principal component analyzers. In *Proceeding of International Conference on Artificial Neural Networks*, pages 914–919. Springer, 2002.

[57] Y. W. Teh and S. T. Roweis. Automatic alignment of hidden representations. In *Advances in Neural Information Processing Systems 15*, pages 841–848, 2003.

[58] J. J. Verbeek, S. T. Roweis, and N. Vlassis. Non-linear CCA and PCA by alignment of local models. In *Advances in Neural Information Processing Systems 15*, pages 297–304. MIT Press, 2003.

[59] M. Brand. Charting a manifold. In *Advances in Neural Information Processing Systems 15*, pages 961–968. MIT Press, 2003.

[60] C. M Bishop, M. S., and C. KI Williams. Developments of the generative topographic mapping. *Neurocomputing*, 21(1):203–224, 1998.

[61] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Springer, 1991.

[62] P. Mitra, C. A. Murthy, and S. K. Pal. Density-based multiscale data condensation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:734–747, 2002.

[63] D. W. Scott and S. J. Sheather. Kernel density estimation with binned data. *Communications in Statistics - Theory and Methods*, 14:1353–1359, 1985.

[64] P. Hall and M. P. Wand. On the accuracy of binned kernel density estimators. *Journal of Multivariate Analysis*, 56:165–184, 1996.

[65] M. Girolami and C. He. Probability density estimation from optimally condensed data samples. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:1253–1264, 2003.

[66] K. Zhang and J. T. Kwok. Simplifying mixture models through function approximation. In *Advances in Neural Information Processing Systems 18*, pages 825–832, Englewood Cliffs, NJ, 2006.

[67] J. Goldberger and S. Roweis. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems 17*, pages 505–512, 2005.

[68] X. Hong, S. Chen, and C. J. Harris. A forward-constrained regression algorithm for sparse kernel density estimation. *IEEE Transactions on Neural Networks*, 19(1):193–198, 2008.

[69] J. Catlett. Megainduction: Machine learning on very large databases, 1991. Unpublished doctoral dissertation, University of Sydney, Australia.

[70] D. Huang and T. W. S. Chow. Enhancing density-based data reduction using entropy. *Neural computation*, 18(2):470–495, 2006.

[71] L. Holmstrom. The accuracy and the computational complexity of a multivariate binned kernel density estimator. *Journal of Multivariate Analysis*, 72:264–309, 2000.

[72] P. Hall. The influence of rounding errors on some nonparametric estimators of a density and its derivatives. *SIAM Journal on Applied Mathematics*, 42, 1982.

[73] M. Pawlak. Kernel density estimation with generalized binning. *Scandinavian Journal of Statistics*, 26(4):539–561, 1999.

[74] A. J. Smola and B. Schökopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of 14th International Conference on Machine Learning (ICML'00)*, pages 911–918, 2000.

[75] P. B. Nair, A. Choudhury, and A. J. Keane. Some greedy learning algorithms for sparse regression and classification with mercer kernels. *Journal of Machine Learning Research*, 3:781–801, 2002.

[76] P. Sun and X. Yao. Sparse approximation through boosting for learning large scale kernel machines. *IEEE Transactions on Neural Networks*, 21(6):883–894, 2010.

[77] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[78] S. Chen, S. A. Billings, and W. Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50:1873–1896, 1989.

[79] C. R. Loader. Local likelihood density estimation. *The Annals of Statistics*, 24(4):1602–1618, 1996.

[80] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[81] M. A. Fardal, A. Babul, J. J. Geehan, and P. Guhathakurta. Investigating the andromeda stream - ii. orbital fits and properties of the progenitor. *Monthly Notices of the Royal Astronomical Society*, 366:1012–1028, Mar 2006.

[82] M. A. Fardal, P. Guhathakurta, A. Babul, and A. W. McConnachie. Investigating the andromeda stream - iii. a young shell system in m31. *Monthly Notices of the Royal Astronomical Society*, 380:15–32, sep 2007.

[83] R. F. G. Wyse. The merging history of the milky way disk. *Astrophysics*, 2000.

[84] B. Moore, N. Katz, G. Lake, A. Dressler, and A. Oemler. Galaxy harassment and the evolution of clusters of galaxies. *Nature*, 379:613, 1996.

[85] G. W. Snedecor and W. G. Cochran. *Statistical Methods*. Iowa State University Press, 1989.

[86] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[87] C. Ding, D. Zhou, X. He, and H. Zha. R1-PCA: Rotational invariant L1-norm principal component analysis for robust subspace factorization. In *Proceedings of the 23rd international conference on Machine learning*, pages 281–288, 2006.

[88] N. Kwak. Principal component analysis based on L1-Norm maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1672–1680, 2008.

[89] D. D. Ridder and V. Franc. Robust subspace mixture models using t-distributions. In *Proceedings of the 14th British Machine Vision Conference*. BMVA, 2003.

[90] J. Gao, P. W. Kwan, and Y. Guo. Robust multivariate L1 principal component analysis and dimensionality reduction. *Neurocomputing*, 72(4-6):1242–1249, 2009. Brain Inspired Cognitive Systems (BICS 2006) / Interplay Between Natural and Artificial Computation (IWINAC 2007).

[91] D. Skocaj, A. Leonardis, and H. Bischof. Weighted and robust learning of subspace representations. *Pattern Recognition*, 40(5):1556 –1569, 2007.

[92] Alfredo Vellido. Missing data imputation through GTM as a mixture of t-distributions. *Neural Networks*, 19(10):1624–1635, 2006.

[93] Z. Khan and F. Dellaert. Robust generative subspace modeling: The subspace t distribution. Technical report, GVU Center, College of Computing, Georgia, 2004.

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

[95] P. Tino and I. Nabney. Hierarchical GTM: constructing localized non-linear projection manifolds in a principled way. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 2001.

[96] C. Williams. A MCMC approach to hierarchical mixture modelling. In *Advances in Neural Information Processing Systems 12*, pages 680–686. MIT Press, 2000.

[97] A. Kaban, P. Tino, and M. Girolami. A general framework for a principled hierarchical visualization of multivariate data. In *Proceedings of the Third International Conference on Intelligent Data Engineering and Automated Learning*, pages 518–523. Springer-Verlag, 2002.

[98] K. Fukunaga. Intrinsic dimensionality extraction. *Classification, Pattern Recognition and Reduction of Dimensionality*, 2:347–360, 1982. Handbook of Statistics, P.R. Krishnaiah and L.N. Kanal.

[99] Francesco Camastra. Data dimensionality estimation methods: A survey. *Pattern Recognition*, 36:2945–2954, 2003.

[100] F. Camastra and A. Vinciarelli. Estimating the intrinsic dimension of data with a fractal-based approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(10):1404–1407, 2002.

[101] E. Levina and P. Bickel. Maximum likelihood estimation of intrinsic dimension. In *Advances in Neural Information Processing Systems 17*, 2005.

[102] K. Fukunaga and D.R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(2):165–171, 1979.

[103] G.V. Trunk. Statistical estimation of the intrinsic dimensionality of a noisy signal collection. *IEEE Transactions on Computers*, 25:165–171, 1976.

[104] K. Pettis, T. Bailey, T. Jain, and R. Dubes. An intrinsic dimensionality estimator from near-neighbor information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(1):25–37, 1979.

[105] P.J. Verveer and R. Duin. An evaluation of intrinsic dimensionality estimators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(1):81–86, 1995.

[106] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 3:507–522, 1994.

[107] J. Bruske and G. Sommer. Intrinsic dimensionality esitmation with optimally topology preserving maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:572–575, May 1998.

[108] P. Mordohai and G. Medioni. Unsupervised dimensionality estimation and manifold learning in high-dimensional spaces by tensor voting. In *International Joint Conference on Artifcial Intelligence*, pages 798–803, 2005.