TÉCNICO LISBOA

Shape-based Trajectory Clustering

Telmo José Pessoa Pires

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor: Prof. Mário Alexandre Teles Figueiredo

Examination Committee

Chairperson: Prof. João Manuel Lage de Miranda Lemos
Supervisor: Prof. Mário Alexandre Teles de Figueiredo
Member of the Committee: Prof. Jorge dos Santos Salvador Marques

October 2016

“Never give up on a dream just because of the time it will take to accomplish it. The time will pass anyway.”

Earl Nightingale

Acknowledgments

Acknowledgments are an ungrateful task, and it is very hard to thank everyone. Nevertheless, I will try my best.

First of all, I would like to thank my supervisor, Prof. Mário Figureiredo, for his help and supervision during the preparation of this work. Without his help or his assertive remarks, none of this would have been possible.

I would also like to thank Prof. João Miranda Lemos for his assistance when I was searching for a thesis topic. Had not been for his patience and I would not have met Prof. Mário Figueiredo.

Special thanks to Instituto de Telecomunicações (IT) for the access to room 10.21, where most of the work was conducted, and to the SPARSIS project, for providing some of the datasets used in this work.

Last, but not least, I would like to thank my friends and family for the support and encouragement over the last 5 years. Had not been for them and I would not be where I am.

This work was partially supported by FCT grant PTDC/EEI-PRO/0426/2014.

Resumo

A classificação automática de trajetórias tem inúmeras aplicações, que vão desde as ciências naturais, como a zoologia e a meteorologia, até ao planeamento urbano e à análise desportiva, o que tem gerado grande interesse e investigação.

O objetivo deste trabalho é propor e testar experimentalmente um novo método de agrupamento de trajetórias, baseado na sua forma e não na posição, tal como acontece com os métodos até agora desenvolvidos.

O método proposto baseia-se na reamostragem uniforme das trajetórias utilizando *splines*, e na sua caraterização com base nos ângulos das tangentes nos pontos reamostrados. Quantidades angulares introduzem algumas dificuldades para a análise, devido à sua natureza periódica, o que impossibilita a utilização direta de métodos de agrupamento usuais. Para contornar o problema, três métodos foram desenvolvidos/adaptados para aplicação neste trabalho: uma variante do algoritmo k-means, um modelo de misturas de distribuições de Von Mises multivariadas, que é ajustado recorrendo ao algoritmo EM e a factorização esparsa e não negativa de matrizes. Como o número de grupos raramente é conhecido *à priori*, foram também introduzidos métodos para automatizar a sua escolha. Finalmente, estes algoritmos foram testados em dados reais e simulados, nos quais se demonstrou a viabilidade da técnica.

Palavras-chave: Aprendizagem Não Supervisionada, Estatística Direcional, k-means, NMF, Algoritmo EM, Agrupamento de Trajetórias.

Abstract

Automatic trajectory classification has countless applications, ranging from the natural sciences, such as zoology and meteorology, to urban planning and sports analysis, and has generated great interest and investigation.

The purpose of this work is to propose and test new methods for trajectory clustering, based on shape, rather than spatial position, as is the case with previous methods.

The proposed approach starts by uniformly resampling the trajectories using splines, and then characterizes them using the angles of the tangents at the resampled points. Angular data introduces some challenges for analysis, due to its periodic nature, therefore preventing the direct application of common clustering techniques. To overcome this problem, three methods are proposed/adapted: a variant of the k-means algorithm, a mixture model using multivariate Von Mises distributions, which is fitted using the EM algorithm, and sparse nonnegative matrix factorization. Since the number of clusters is seldom known *a priori*, methods for automatic model selection are also introduced. Finally, these techniques are tested on both real and synthetic data, and the viability of this approach is demonstrated.

Keywords: Unsupervised Learning, Directional Statistics, k-means, NMF, EM Algorithm, Trajectory Clustering.

Contents

List of Tables

List of Figures

List of Acronyms

DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DTW	Dynamic Time Warping
EM	Expectation-Maximization
FMM	Finite Mixture Model
GMM	Gaussian Mixture Model
GPS	Global Positioning System
LCSS	Longest Common Sub-Sequence
MAP	Maximum a Posteriori
MDL	Minimum Description Length
ML	Maximum Likelihood
NMF	Nonnegative Matrix Factorization
PCA	Principal Component Analysis
RFID	Radio-frequency identification
SSNMF	Sparse Semi Nonnegative Matrix Factorization
VMM	Von Mises Mixture

Chapter 1

Introduction

1.1 Background

In recent years, an exponential growth in the amount of available data for analysis has been experienced in almost all fields of knowledge. These huge data quantities, far beyond the scope of manual analysis, have stimulated a growing interest in automated methods. One task of primary interest is finding patterns and regularities within such data. This is, however, a non-trivial task and, in some applications, such as image analysis, it is not even obvious how to create an algorithm to accomplish it. To face these challenges, the field of *machine learning* emerged.

Machine learning is closely related to Computer Science, Artificial Intelligence and Statistical Inference and, as Arthur Samuel defined it, is the "field of study that gives computers the ability to learn without being explicitly programmed". Instead, they are programmed to *learn from data*. Its applications range from medicine to finance and it is used in many places of day to day life, such as spam filters, recommender systems in e-commerce platforms, among others.

1.1.1 Supervised Learning

One of the main "branches" of Machine Learning is *supervised learning*. In supervised learning, a collection of examples including the "right answer" to the problem is given, i.e., the data is *labeled*. For example, in the spam filtering problem, the learning algorithm is given a set of examples of e-mail and told, about each one of them, if it is spam or not.

Supervised learning can be subdivided into two different subproblems: classification and regression. In classification, the goal of the algorithm is to categorize the input into a finite number of categories. In regression, the desired output is one or more continuous variables, which depend on some way on the inputs.

1.1.2 Unsupervised Learning

Being dependent on the "right answer" may be a serious limitation, as human experts are required to provide it, which may be impractical for large data sets. *Unsupervised learning* addresses this issue: it tries to discover patterns or a hidden structure in *unlabeled data*, i.e., without being given the "right answer".

Bishop [1] states three main tasks in unsupervised learning: clustering, density estimation, and dimensionality reduction. In clustering, the goal is to group data in a way that the elements within a group are more similar among themselves than to those in other groups. In density estimation, as the name implies, the goal is to estimate the distribution of the input data. Finally, in dimensionality reduction, the goal is to project the given data onto lower dimensions, for example, for visualization, or memory saving purposes.

The applications of unsupervised learning are immense, ranging from image analysis to bioinformatics, and it is a field that will keep growing, as most of the data available is unlabeled. The present thesis will focuses on the clustering problem, in particular on trajectory clustering.

1.2 Motivation

The growing number of tracking devices including, but not limited to, GPS receivers, RFID tags, and tracking cameras, has caused an explosion in the amount of trajectory data available from different sources, thus precluding manual analysis. Even the traces of cellphone calls being handed between towers can be used to track mobility patterns [2, 3]. This has led to a growing interest in automatic trajectory clustering, as a means to perform activity recognition and classification. Such tools and methodologies find applications in various areas, ranging from politics to biology:

- Policy makers can study traffic patterns, human mobility, and even air pollution exposure of populations [4], allowing better urban planning and more informed policy making. Trends spotted in human behavior can also be used in targeted advertising.
- In meteorology, these tools can be used to spot patterns and common behaviors in meteorological phenomena, such as hurricanes, helping improve the accuracy of forecasts [5, 6].
- Biologists and zoologists can study animal movement and their distribution, as well as feeding, habitat, and migratory patterns [7]. Social interactions between species and changes over time, possibly caused by human activities, such as road building, and climate change, can also be analyzed. These techniques could also be used to perform a "raw-mapping" of the terrain, as animals avoid hilly areas, and prefer areas rich in water and food.
- Coaches and professional sports players can use these tools to analyze previous matches and study their behaviors and identify opponents' patterns [8].
- Other uses include automatic outdoor surveillance and abnormal event detection, such as moving in forbidden directions or areas.

The analysis of trajectory data, however, is non-trivial, and there are many problems yet to be solved. Trajectories can be influenced by many factors, some not directly related to the phenomenon in study. Animal tracks, for example, tend to depend on the age of the animals, but can also change due to temporary weather conditions. Hurricane tracks, while depending on storm strength, are very dependent on seasonality. Besides the dependence on application-specific characteristics, trajectory data, where a trajectory is considered as a sequence of space-time data points, presents, by itself, many challenges [9]:

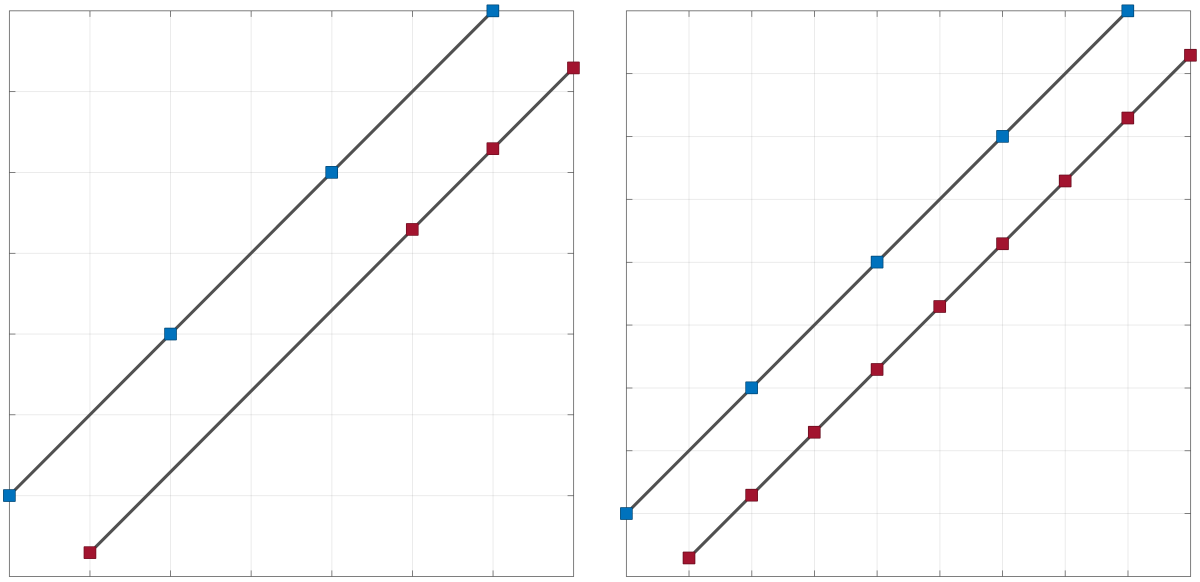
1. Different speeds: differences in the speed of tracked objects cause misalignments in the trajectories, as illustrated in Figure 1.1(a). The object in blue moves at a constant speed, while the object in red moves faster in the beginning and slower in the end, leading to different spacings between points (observations) in the trajectory, assuming a constant sampling rate.

2. Different sampling rates: sampling rate can change between tracks and even within the same track (as a result of, for example, sensor delays). In practice it is equivalent to having objects move at different speeds, with a constant sampling rate.

3. Different lengths: similar trajectories may have a different number of points, as illustrated in 1.1(b). This can be caused by different speeds and/or different sampling rates/methods.

4. Similar motions in different regions: a spacial translation of a trajectory gives a similar one, and it may be of interest to detect that.

5. Noise and outliers: besides being affected by noise, trajectories can also have big, but sparse errors, called outliers. A possible source for this kind of errors is multipath interference in satellite tracking. Good trajectory analysis must be robust to them.



(a) Objects moving at different speeds, leading to misaligned, but similar tracks.

(b) Similar tracks with a different number of points.

Figure 1.1: Some of the challenges faced with trajectory data.

1.3 Previous work and contributions

Due to their many practical applications, many authors have focused on extracting movement patterns from trajectory data. Despite the diversity of available methods, there is no agreement on which one is the best.

The approaches in literature can be divided into two main groups: the distance metric approach and the model based (generative) approach.

1.3.1 Distance metric approaches

When dealing with trajectory data to address tasks such as classification or clustering, the first problem that comes to mind is how to find (dis)similarity measures between trajectories. Numerous reports defining various (dis)similarity measures have been published and are reviewed in the following paragraphs.

In [10, 11], an Euclidean distance was used. In that approach, the dissimilarity between 2 trajectories is proportional to the Euclidean distance between the vectors of points in the 2 trajectories. The authors of [12] proposed a similar metric, but used PCA (principal component analysis) to perform dimensionality reduction, which led to a more robust and faster algorithm, working in a lower dimensional space. Both methodologies require trajectories with the same number of points and uniformly sampled, so a resampling preprocessing step is needed to fulfill these conditions. Without resampling, even if the number of points is the same, these metrics perform poorly as they are very sensitive to speed differences between tracks, which produce misalignments in the trajectories (see Figure 1.1(a)).

To overcome the need for uniform resampling, Pierobon et al. [13] use *dynamic time warping* (DTW), an algorithm that finds the optimal alignment between two time-dependent sequences. Such alignment may stretch or shorten parts of the two sequences to find the best alignment, hence the name *time warping*. DTW has found applications in music and speech recognition [14], and had relative success in pattern identification in financial time series [15]. Applied to trajectory data [13], it is able to deal with trajectories of different lengths (number of points). Since it finds the optimal alignment between two trajectories, it automatically solves the problem of the different speeds. However, since it tries to match all points of the two trajectories, it is very sensitive to noise and outliers, and does not scale well.

To solve the outlier problem with DTW, a similar metric was proposed in [9]: *longest common subsequence* (LCSS). As the name states, LCSS finds the longest "close enough" alignment between two trajectories, allowing them to stretch or shrink, but it can ignore points that are unusually far from the rest (outliers).

Any of the previously described distance functions can and have been used with a myriad of clustering algorithms, such as direct, hierarchical, and spectral clustering. Morris and Trivedi [16] performed a comparative analysis and obtained the best results using LCSS, closely followed by DTW.

While the previous approaches simply defined (dis)similarity measures, a complete framework for trajectory analysis, an algorithm called TRACLUS, was proposed in [17]. TRACLUS partitions trajectories and allows sub-trajectory clustering, instead of clustering the trajectories as a whole. It defines

special metrics that measure 3 types of distance between trajectories: perpendicular, parallel, and angular. The method involves two steps: it first attempts to find an optimal partitioning of the trajectories, using principles from information theory (minimum description length - MDL [18]); then, density-based clustering (DBSCAN) is used. The main drawbacks of TRACLUS are its high sensitivity to noise and the high dependence of the results on its input parameters, which are difficult to set and depend on the dataset. Jiashun [19] attempted to solve these problems and proposed a method that is less sensitive to the parameters.

1.3.2 Model-based approaches

Model-based methods, also known as generative methods, attempt to find a model capable of describing the whole dataset, instead of directly comparing trajectories.

In [20], a clustering method based on a mixture model is proposed. It represents trajectories as polynomial functions of time, whose parameters are estimated by the mixture model. Estimation is performed using the EM (expectation-maximization) algorithm. In [21], some extensions are introduced to allow for spatial and temporal shifts in the data.

A similar approach, which models trajectories using B-splines was proposed in [22]. The methodology is similar to the one in [21], but is optimized for parallel computing.

A different approach, believed to be superior by some authors, is the use of vector fields [23]. Vector fields directly encode information about speed and position of the trajectories, and allow for easy representation of the results.

Ferreira et al. [23] proposed a clustering algorithm based on k-means, but using vector fields. Vector fields are estimated from data using a *Laplacian* penalty to ensure smoothness. Its main advantages are its speed and the ability to work well with partial or incomplete data. Its main disadvantage is getting stuck in poor local optima. The authors proposed an initialization procedure that attempts to reduce such behavior, but offers no theoretical guarantees.

A mixture of vector fields model was proposed in [24]. It is based on the assumption that trajectories can be described by a small number of typical behaviors, modeled as vector fields. It allows the transition between multiple fields based on a probabilistic transition model using Hidden Markov Models, for a more expressive representation. Both the transition model as well as the vector fields are estimated from data, using an EM algorithm.

1.4 Objectives

The main goal of this thesis is to derive, implement, and test methods for trajectory clustering in the far-field scenario, using only shape information. This is a novel approach, whose underlying idea is that, in many applications, position information is not necessary, as similar trajectory shapes mean similar activities.

The methods will be developed for $2D$ trajectories, but aside from this, no other assumption will be made, and trajectories may be corrupted by noise and/or outliers and may have different lengths.

1.5 Thesis Outline and Contributions

Chapter 2 describes the nomenclature and notation used throughout this thesis and introduces concepts from statistics that are used in the following chapters.

Chapter 3 describes the pre-processing step applied to trajectories to characterize their shape. A technique for choosing the number of features used is also presented.

Chapter 4 introduces the k-means clustering algorithm for Euclidean data. It then derives new metrics for using it with circular data.

Chapter 5 introduces Finite Mixture Models and derives the equations for fitting mixtures of multidimensional Von Mises distributions. It also presents a method for model selection based on information theory concepts.

Chapter 6 introduces Nonnegative Matrix Factorization and its relationship with clustering. It then proceeds to adapt it for clustering circular data.

Chapter 7 presents the obtained results, in both synthetic and real data.

Chapter 8 wraps up this work and discusses achievements and suggestions for future work.

The main contribution of this thesis is the introduction of a novel approach to trajectory clustering, based on shape, rather than position. This is accomplished by means of a pre-processing step, that extracts shape information, and adapted versions of three commonly used clustering methods.

Chapter 2

Theoretical Background

This chapter presents some useful concepts and notation used throughout this thesis. First, we establish the adopted notation. We then define the clustering problem and state how it can be useful for trajectories. Next, basic statistic concepts, such as maximum likelihood, and a statistical distribution for circular data are reviewed. Finally, a key concept from information theory is presented.

2.1 Notation

In this short subsection, we summarize the main notational choices made when writing this thesis:

- Scalars and vectors are denoted by lowercase letters and matrices by uppercase letters. To distinguish scalars and vectors, the latter are written in bold.
- For random variables, uppercase refers to the variables themselves, while lowercase refers to a particular realization. For example, $p_X(x)$ is the probability density function of random variable X evaluated at $X = x$.
- Finally, following machine learning nomenclature, we call the datasets we are working with *training sets*, and a particular sample, *training sample*. Although the order of the samples is arbitrary, they are usually numbered, and $x^{(i)}$ refers to the ith sample.

2.2 Clustering

Clustering is the task of grouping objects, in a way that elements in the same group are more similar among themselves than with those in other groups. In some contexts, each group can be represented by a "typical" (e.g., mean) element, that roughly characterizes it.

Figure 2.1 shows a dataset that is clearly composed of 3 main groups, each with a different color. The goal of a clustering algorithm is to find such groups in high dimensional and noisy data.

Many different definitions of what constitutes a cluster exist, and based on them, different algorithms have been proposed, such as k-means, hierarchical clustering, DBSCAN, etc [25]. Despite this diversity,

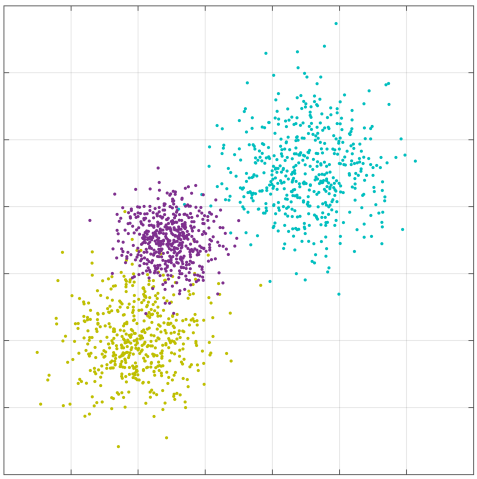


Figure 2.1: Figure showing a dataset with 3 differently colored clusters.

there is no method decidedly better than others, and the choice depends on the application.

Applied to trajectory analysis, clustering aims at finding groups of similar trajectories, so that different activities and types of motion can be recognized.

2.3 Statistical Inference

In this section, the basics of parameter estimation and statistical inference are briefly reviewed.

2.3.1 Maximum Likelihood

Given a data set $\{x^{(1)}, \ldots, x^{(m)}\}$ with m samples observed from a population X with probability density function (pdf) $p_X(x; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is an unknown parameter vector that characterizes X, the likelihood function is defined as

$$\mathcal{L}(\boldsymbol{\theta}, x^{(1)}, \ldots, x^{(m)}) = p(x^{(1)}, \ldots, x^{(m)}; \boldsymbol{\theta}) = \prod_{i=1}^{m} p_X(x^{(i)}; \boldsymbol{\theta}), \tag{2.1}$$

where the second equality assumed the observations are independent and identically distributed (i.i.d.), so the joint pdf equals the product of the individual pdf's.

Maximum likelihood (ML) estimation is a method for estimating the parameters of a statistical model given experimental data. Its reasoning is that the most likely value for $\boldsymbol{\theta}$ is that which maximizes the likelihood of the observations:

$$\hat{\boldsymbol{\theta}}_{ML} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\arg\max}\, \mathcal{L}(\boldsymbol{\theta}, x^{(1)}, \ldots, x^{(m)}). \tag{2.2}$$

ML provides a systematic way for estimating parameter vectors and it can be shown that, when a ML

estimator exists, there is no asymptotically better estimator [26].

Many important density functions involve exponentials, so it is usually helpful to take logarithms and simplify the expressions. As the logarithm is a strictly increasing function, maximizing the likelihood or the log-likelihood gives the same result, making Eq. (2.2) equivalent to

$$\hat{\boldsymbol{\theta}}_{ML} = \arg\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \ln(\mathcal{L}(\boldsymbol{\theta}, x^{(1)}, \ldots, x^{(m)})). \tag{2.3}$$

2.3.2 Bayes Theorem

In classical estimation, parameter vector $\boldsymbol{\theta}$ is assumed to be deterministic, yet unknown. The Bayesian approach, on the other side, assumes $\boldsymbol{\theta}$ is a random variable, whose particular realization must be inferred from the observed data.

This method is directly based on Bayes' Theorem, which provides a method for updating knowledge about the parameters:

$$p(\boldsymbol{\theta}|x^{(1)}, \ldots, x^{(m)}) = \frac{p_X(x^{(1)}, \ldots, x^{(m)}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p_X(x^{(1)}, \ldots, x^{(m)}|\ \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}} \tag{2.4}$$

The left-hand side of this equation is the *posterior pdf*, i.e., *the probability density function of the parameter vector computed at $\boldsymbol{\theta}$, given the observations*. On the right-hand side, $p_X(x|\boldsymbol{\theta})$ is the likelihood function and $p(\boldsymbol{\theta})$ is the *prior pdf*, expressing prior knowledge about the parameters. The term in denominator is a normalizing constant, that assures the posterior integrates to 1.

2.3.3 Maximum a Posteriori

Maximum a posteriori (MAP) estimation is, similarly to ML, a method for estimating parameters of statistical models, but it employs an augmented objective function: it seeks the maximizer of the posterior pdf, rather than the likelihood function.

The denominator term in Bayes' theorem is constant (with respect to the parameters), so it may be dropped in the maximization. Once again, logarithms are usually applied to simplify the resulting expressions, so the final formula for the MAP estimator is

$$\hat{\boldsymbol{\theta}}_{MAP} = \arg\max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \left[\ln(p(x^{(1)}, \ldots, x^{(m)}|\boldsymbol{\theta})) + \ln(p(\boldsymbol{\theta}))\right]. \tag{2.5}$$

Clearly, if the prior is uniform, $\ln(p(\boldsymbol{\theta}))$ is constant, and MAP reduces to ML.

2.4 Directional Statistics

Circular data, such as angles, time of day, and directions (north, south, ...) arises in many practical applications and poses some challenges, as samples differing of multiples of the period represent the same value. The fundamental property of this kind of data is that the beginning and the end of the scale coincide, so it can be represented in a circle, hence the name.

When dealing with such data, usual statistical methods and distributions can not be directly applied. Directional Statistics, a subfield of statistics, is the discipline that focuses on it.

2.4.1 Univariate Von Mises Distribution

From a statistical inference standpoint, the Von Mises distribution is one of the most useful distributions. It is a continuous unimodal distribution on the circle and is a close approximation to the wrapped normal, a circular analogue of the normal distribution. Due to its simplicity, it is the most used circular distribution in practical applications.

The Von Mises pdf is given by

$$p_X(x;\mu,\kappa) = \frac{e^{\kappa\cos(x-\mu)}}{2\pi I_0(\kappa)}, \tag{2.6}$$

where μ is its mean and $\kappa > 0$ is the concentration parameter, which is analogous to the inverse variance of the Gaussian distribution. The Von Mises pdf reduces to the uniform distribution on the circle when $\kappa \to 0$, and to a Gaussian with mean μ and variance $1/\kappa$ when $\kappa \to \infty$. $I_0(\kappa)$ denotes the modified Bessel function of the first kind and order 0, and acts as a normalizing constant so the pdf integrates to 1:

$$I_0(\kappa) = \frac{1}{2\pi}\int_0^{2\pi} e^{\kappa\cos\omega} d\omega. \tag{2.7}$$

Figure 2.2 shows a Von Mises pdf on the unit circle. Its shape is very close to a normal distribution, but wrapped around the unit circle.

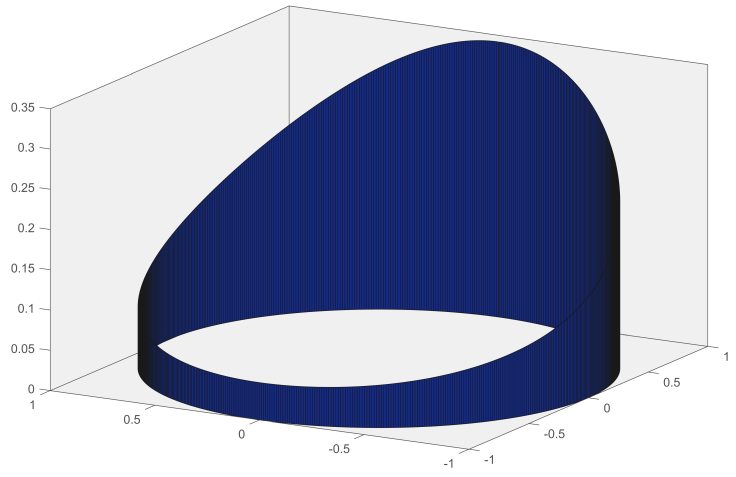


Figure 2.2: Plot of the univariate Von Mises distribution on the unit circle.

2.4.2 Multivariate Von Mises Distribution

A multivariate extension of the Von Mises distribution was proposed in [27], known as the multivariate sine distribution. Instead of being a distribution on a circle, as the univariate case, it is the distribution

on a hyper-dimensional torus. For d-dimensional data, the pdf is

$$p_X(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\kappa},\boldsymbol{\Lambda}) = T(\boldsymbol{\kappa},\boldsymbol{\Lambda})^{-1}\exp\left(\boldsymbol{\kappa}^T c(\boldsymbol{x},\boldsymbol{\mu}) + \frac{1}{2}s(\boldsymbol{x},\boldsymbol{\mu})^T\boldsymbol{\Lambda}\; s(\boldsymbol{x},\boldsymbol{\mu})\right), \tag{2.8}$$

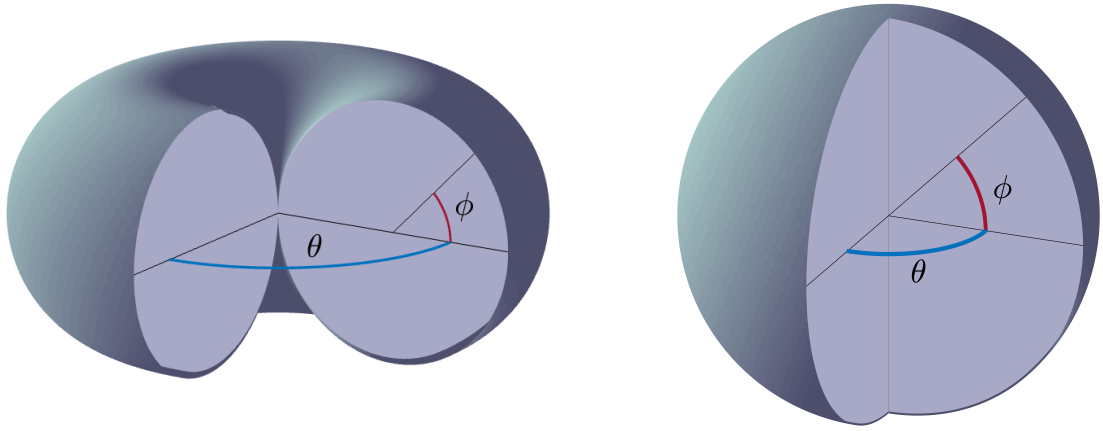
where

$$c(\boldsymbol{x},\boldsymbol{\mu})^T = [\cos(x_1-\mu_1),\ \ldots\ ,\ \cos(x_d-\mu_d)]\,,$$

$$s(\boldsymbol{x},\boldsymbol{\mu})^T = [\sin(x_1-\mu_1),\ \ldots\ ,\ \sin(x_d-\mu_d)]\,,$$

and $T(\boldsymbol{\kappa},\boldsymbol{\Lambda})^{-1}$ is a normalizing constant, which is unknown in explicit form for $d > 2$. Similarly to the univariate case, $\boldsymbol{\mu}$ and $\boldsymbol{\kappa}$ are the mean and concentration parameter of the distribution, but here they are d-dimensional vectors; x_p and μ_p are the pth coordinates of these vectors; $\boldsymbol{\Lambda}$ is a $d \times d$ symmetric matrix, where entry Λ_{ij} measures the dependence between x_i and x_j, and whose diagonal contains zeros. Matrix $\boldsymbol{\Lambda}$ can be seen as the analogous to the covariance matrix in a Gaussian distribution.

Another multivariate extension of the Von Mises distribution exists: the Von Mises-Fisher distribution. The main difference between these generalizations is that the sine distribution generalizes the Von Mises distribution to a multi-dimensional torus, while the Von Mises-Fisher generalizes it to a multi-dimensional sphere. This difference can be easily understood with the help of Figure 2.3. While in the case of the torus, both angles θ and ϕ can freely take any value in $[0, 2\pi[$, *independently* of one another, in the case of the sphere, this does not happen. If, for example, θ varies in $[0, 2\pi[$, ϕ is limited to $[-\pi/2, \pi/2]$, because otherwise the representation is *redundant*.



(a) Parameterization of a torus.

(b) Parameterization of a sphere.

Figure 2.3: Angular parameterization of the torus and the sphere. In the torus, both angles can vary freely in $[0, 2\pi[$ (or any other interval with length 2π), while in the sphere only one can.

For the uses intended in this thesis, the sine distribution must be used, as we want each dimension (feature) of the vector describing a trajectory to be independent of the others, and thus, to freely vary in $[0, 2\pi[$.

2.5 Minimum Description Length Principle

The Minimum Description Length (MDL) principle, is a concept from information theory that provides a systematic, formal way to address problems of model selection [18]. It is based on the idea that regularities within data can be used to compress it: the more regularities, the more compressed it can be. In other words, if there is a good model of the data, it can be compressed to lower lengths. Using the lengths as a sort of cost, one can use MDL to choose the complexity of a model. In our case, this is the number of clusters present in a given dataset.

The standard form of MDL has two components:

- The first is the length of the hypothesis, represented as $L(H)$. It is the length spent in encoding the model (hypothesis) used in data compression.
- The second is the length of the data encoded with the hypothesis, $L(D|H)$. Using a hypothesis that captures many regularities in the data, one can compressed it to smaller lengths.

The final MDL cost is the sum of these components, $MDL_{cost} = L(H) + L(D|H)$. This way, MDL formalizes a key concept in model selection: a trade-off between model order and model size. Bigger models allow better encodings of the data, but are, by definition, bigger. Choosing the minimum MDL cost is choosing a compromise between model size and model accuracy. The unit in which these lengths are in is irrelevant, as long as it is consistent. Depending on the base of the logarithm used in computing the lengths, the units can be bits (base 2), nats (base e), hartleys (base 10), etc.

Chapter 3

Data Pre-processing

The main goal of this thesis is to cluster trajectories according to shape. Since a trajectory is usually given as a sequence of points, a pre-processing step is needed in order to capture its shape.

In this work, the shape of a trajectory is characterized as a sequence of angles: the angles of the tangents to the trajectory, sampled at uniformly spaced points, as illustrated in Figure 3.1. These angles are measured relative to the horizontal, but this reference is arbitrary. As shown below, the sequence of angles is invariant under changes in location and scale of the trajectory.

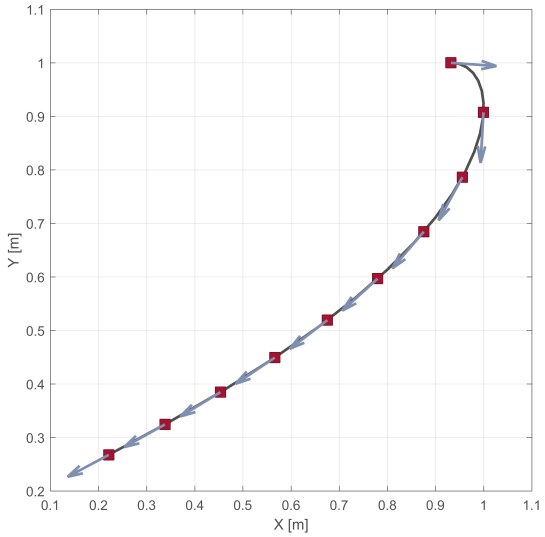


Figure 3.1: Uniformly resampled trajectory and respective tangents. Shape is characterized by the angles the tangent vectors make with the horizontal.

The uniform resampling step is not novel (see [10, 11, 12]), and allows us to cope with the main challenges of trajectory data stated in section 1.2:

1. Different lengths: due to many factors, trajectories usually have different lengths (number of data points). By resampling to a fixed number of points, this issue is solved.

2. Trajectory misalignment: due to variations in the speed of tracked objects, or in sampling rates, raw

trajectory data may suffer from misalignment between points. Uniform resampling assures that all trajectories are similarly sampled.

The use of shape instead of spatial location allows for the detection of similar motions in different regions, as any algorithm will be invariant to space translation of data. Besides, in many cases of interest, position information is redundant, as trajectories exhibit *spacial locality*: similar trajectories in the same area, due to constraints of the terrain, tend to exhibit similar shapes.

One way to resample data is to fit a mathematical function to it, and then use this function as a predictor of the values such data would take at other points. In this work, cubic splines are used for this purpose.

3.1 Interpolating and Smoothing Splines

3.1.1 Interpolating Splines

In spline interpolation, the interpolating function is a piecewise polynomial [28]. This has two main advantages: first, polynomials can be easily differentiated and integrated, making the interpolation and evaluation processes easy. Second, by using a piecewise approach, the order of the interpolating polynomial can be kept low (otherwise the more complex the curve, the higher the required degree). This avoids Runge's phenomenon, a problem in which oscillation occurs between points interpolated using high degree polynomials.

The most commonly used splines are cubic, i.e., using 3rd degree polynomials. The interpolant function is chosen to be twice continuously differentiable, i.e, to have continuous first and second derivatives, producing smooth curves. Many different, yet equivalent formulations exist for splines. The notation used here closely follows [28].

Given a set of data, $x(\tau_1), \ldots, x(\tau_m)$, the values of a function x evaluated at $\tau_1, \ldots, \tau_m$, the interpolant, f, is constructed to agree with x on these values. On each interval $[\tau_i, \tau_{i+1}]$, we have $f(\tau) = P_i(\tau)$, where $P_i(x)$ is a 3rd degree polynomial, which can be written as

$$P_i(\tau) = c_{1,i} + c_{2,i}(\tau - \tau_i) + c_{3,i}(\tau - \tau_i)^2 + c_{4,i}(\tau - \tau_i)^3. \tag{3.1}$$

With

$$\begin{aligned}
c_{1,i} = \quad & P_i(\tau_i) && = x(\tau_i), \\
c_{2,i} = \quad & P_i'(\tau_i) && = s_i, \\
c_{3,i} = \quad & P_i''(\tau_i)/2 && = \frac{\frac{x(\tau_{i+1})-x(\tau_i)}{\tau_{i+1}-\tau_i} - s_i}{\tau_{i+1} - \tau_i} - c_{4,i} \cdot (\tau_{i+1} - \tau_i), \\
c_{4,i} = \quad & P_i'''(\tau_i)/6 && = \frac{s_i + s_{i+1} - 2\frac{x(\tau_{i+1})-x(\tau_i)}{\tau_{i+1}-\tau_i}}{(\tau_{i+1} - \tau_i)^2},
\end{aligned} \tag{3.2}$$

where P_i', P_i'' and P_i''' denote the 1st, 2nd, and 3rd derivatives of polynomial $P_i(\tau)$ and s_i is the value of

the 1st derivative of x at τ_i.

Using these coefficients, $f(\tau_i) = x(\tau_i),\ i = 1, \ldots, m$, so only the s_i's (slopes) remain to be chosen. Cubic splines use the condition that f should be twice continuously differentiable to determine these slopes, $s_2, \ldots, s_{m-1}$. Using $P''_{i-1}(\tau_i) = P''_i(\tau_i)$ for all $i = 2, \ldots, m-1$, we get

$$2c_{3,i-1} + 6c_{4,i-1}(\tau_{i-1} - \tau_{i-2}) = 2c_{3,i}. \tag{3.3}$$

Using the previous equations, we get a system of $m-4$ equations and $m-2$ unknowns. For it to be determined, two additional equations need to be specified. These are the values of s_1 and s_m, that is, the slopes at the end points. When there is no specific information about their values, a frequent choice is $s_1 = s_m = 0$, also called *free-end* conditions, resulting in the so-called *natural splines*. These are the two additional conditions used in this work, despite the fact they may reduce the precision of the spline.

3.1.2 Noise and smoothing splines

As stated in section 1.2, an issue with trajectory data is noise. In real data, noise is always present, and a systematic approach to dealing with it must be devised. One such approach is using *smoothing splines*, instead of *interpolating splines*. Interpolating splines force the interpolant to match the given data, that is, $f(\tau_i) = x(\tau_i),\ i = 1, \ldots, m$, which may not be a good practice when data is corrupted by noise. Smoothing splines, on the other hand, fit a smooth function f that *closely* follows the given data, but may not pass through the given points.

A smoothing spline is fitted by minimizing the following cost [28]:

$$p \sum_{i=1}^{m} w_i (x(\tau_i) - f(\tau_i))^2 + (1-p) \int (D^2 f(\tau))^2 d\tau, \tag{3.4}$$

where $p \in [0, 1]$ is a trade-off parameter, w_i are weights typically set to 1, as will be the case in this thesis, and $D^2 f(\tau)$ is the second derivative of function f.

Parameter p controls the trade-off between the two terms of previous equation: the first term is the error incurred by the spline and the second is a *roughness* measure. By an appropriate choice of p, a good compromise between accuracy and smoothness can be achieved. This can be observed in Figure 3.2. For $p = 1$, the second term in Eq. (3.4) disappears, yielding an interpolating spline. On the other hand, for $p = 0$, the first term disappears, and the smoothest possible spline, a straight line, is obtained. Carefully tuning p, through trial and error (or some other criterion, such as cross validation), a good compromise can be obtained. For the data in Figure 3.2, this is $p \approx 0.95$. As can be seen, despite the highly noisy observations, a good and smooth approximation is obtained.

The smoothing spline minimizing Eq. (3.4) can be obtained by an analytical solution (see [28]). Many numerical implementations of these algorithms are readily available online. `MATLAB` provides such tools in the Curve Fitting toolbox, which was used in this thesis.

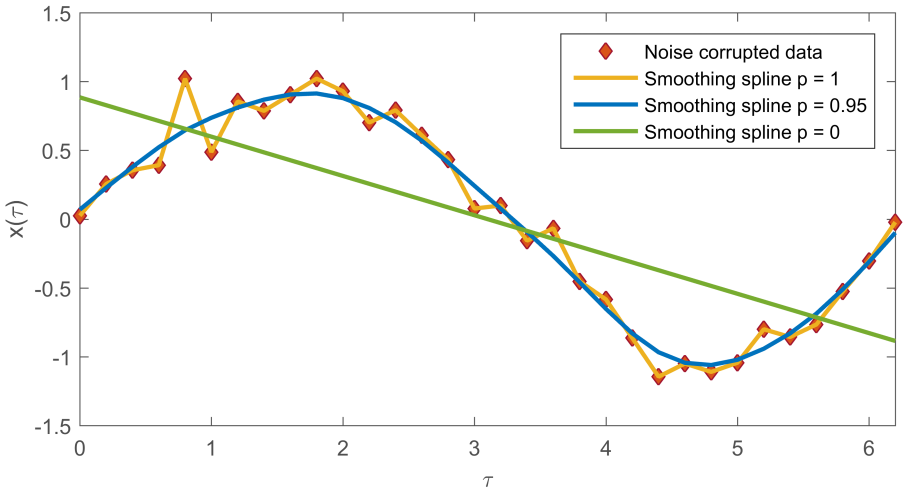


Figure 3.2: Noise corrupted data and smoothing splines for $p = 1$, $p = 0.95$ and $p = 0$.

3.2 Multidimensional splines

Splines, as reviewed above, allow fitting *unidimensional* functions. Trajectories, however, are multi-dimensional data[1], and so some adaptations must made.

Suppose a set of 2-dimensional trajectories, $\{t^{(1)}, \ldots, t^{(m)}\}$ is given. Each trajectory $t^{(i)}$ is composed of a sequence of pairs of points, that is, $t^{(i)} = \{(x_1^{(i)}, y_1^{(i)}), \ldots, (x_{d_i}^{(i)}, y_{d_i}^{(i)})\}$, where d_i is the length of each trajectory (number of points), which may differ within the same dataset. Since any $2D$ trajectory can be decomposed into two sequences of points, the x and y coordinates, natural cubic splines can be fitted to each coordinate, allowing us to describe arbitrarily shaped trajectories.

As seen in previous sections, fitting splines requires two inputs: a dependent and an independent variable, that is, $x(\tau_1), \ldots, x(\tau_m)$ and $\tau_1, \ldots, \tau_m$. In the case of trajectories, the independent variable must be the same for both coordinates. Although a natural choice for the independent variable is time, in many datasets, only spatial location is given. In these cases, the independent variable must be estimated from data. One way of doing so is to use [29]

$$\tau_{p+1} = \tau_p + \sqrt{(x_{p+1}^{(i)} - x_p^{(i)})^2 + (y_{p+1}^{(i)} - y_p^{(i)})^2}, \tag{3.5}$$

with $\tau_1 = 0$, which estimates the τ_i's from the distance between consecutive points. Using this parameter, splines can be fitted independently for both the x and y dimensions.

3.3 Trajectory resampling

Having fitted splines to both coordinates of a trajectory, uniform resampling can be accomplished by evaluating the splines at uniformly spaced τ's. If a uniform resampling to d points is desired, splines are

[1] Bi-dimensional, in the case of this work.

evaluated at

$$\tau_i = \tau_1 + \frac{\tau_m - \tau_1}{d-1} \cdot (i-1)\,, \quad i = 1, \ldots, d,$$

where τ_1 and τ_m are the τ's calculated according to Eq. (3.5).

Although this choice for τ_i guarantees uniform resampling in τ, it does not guarantee that the resampled points are uniformly spaced, due to noise. If some part of a trajectory is noisier than the rest, the distances between consecutive points will increase, and so will the τ_i's. Consequently, this "noisier part" will appear to be longer in terms of τ. Since the resampling is uniform spacing in τ, a zone "lengthier" in τ will have more points than it should, that is, the points will be closer than desired. This is illustrated in Figure 3.3. Despite these small fluctuations, this approach worked well in practice, as will be seen in the following chapters.

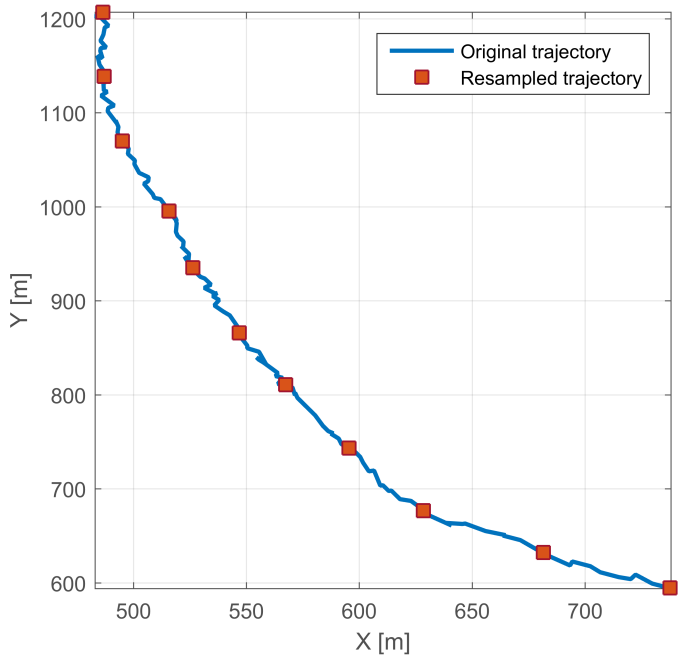


Figure 3.3: Example trajectory, where points are uniformly sampled but not uniformly spaced, although the differences are small.

In this work, we are interested in characterizing the shape of trajectories by a sequence of angles. So, instead of evaluating the splines, we are interested in their derivatives. This is done analytically, since we have an expression for the splines. `MATLAB` provides functions `fnval` and `fnder` to evaluate a spline and compute its derivatives at the specified points. Having determined the first derivatives, the angle of the tangent to the trajectory in each point is given by

$$\omega_p^{(i)} = \operatorname{atan2}(Df_y^{(i)}(\tau_p), Df_x^{(i)}(\tau_p))\,, \quad \omega_p^{(i)} \in [-\pi, \pi]\,,$$

where $\operatorname{atan2}(\cdot)$ is the 4-quadrant inverse tangent function and $Df_y^{(i)}(\tau_p)$ and $Df_x^{(i)}(\tau_p)$ refer to the first derivatives of the fitted splines, evaluated at τ_p. This way, we get a d-dimensional vector $\boldsymbol{\omega}^{(i)}$ describing the trajectory.

This approach assures that besides being invariant to spacial translation, trajectories are also invariant to scaling, since the distance between consecutive samples will be scaled by the same factor. Another interesting property of this formulation is that it allows trajectories to be invariant to rotation with a small modification: by working with the differences between consecutive angles, instead of the angles themselves. As a result, since we are only interested in the *change* of these angles, the orientation of a trajectory does not influence the results.

3.3.1 Choosing p and d

In order to fit smoothing splines to trajectories, two input parameters must be specified: p, the *smoothing* parameter, and d, the number of points of the resampled trajectories.

Regarding p, no systematic procedure for its choice is available. The usual approach, which is used in this thesis, is to try several values until a good compromise is found: smooth trajectories are desired, but not so smooth as to lose important characteristics.

The choice of d, on the other hand, can be automated. In the paper introducing TRACLUS [17], its authors propose a method for partitioning trajectories around points of rapid behavior change, called *characteristic points*. This is illustrated in Figure 3.4: a trajectory composed of the points $p_1, \ldots, p_8$ is represented in full line, while its partition, composed of characteristic points $p_{c_1}, \ldots, p_{c_4}$ is represented in dashed line.

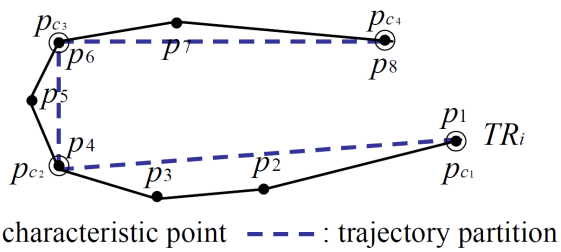


Figure 3.4: Example of a trajectory and its partition. Image from [17].

The number of characteristic points of a trajectory is a lower limit to d: smaller d cannot possibly describe the trajectory without information loss. Thus, d should be higher than the number of characteristic points, as the d points will be uniformly spaced, while the characteristic points may not be. So, in order to fully characterize a trajectory, we use d around 5 times (an order of magnitude) the maximum number of characteristic points of any trajectory in the given dataset.

When trying to find the characteristic points of a trajectory, a trade-off between precision and conciseness arises. By choosing all points of a trajectory as characteristic points, precision is maximized (the original trajectory is kept), but conciseness is minimized. On the other hand, when only the first and last points are kept, conciseness is maximized, but precision is minimized. This trade-off is formalized using the MDL principle.

The MDL cost, or MDL length, of a given partitioning consists of two components, the length of the

hypothesis, $L(H)$ and the length of the data when encoded with the hypothesis, $L(D|H)$, given by

$$L(H) = \sum_{j=1}^{par_i - 1} \log_2(length(p_{c_j} p_{c_{j+1}})),$$

$$L(D|H) = \sum_{j=1}^{par_i - 1} \sum_{k=c_j}^{c_{j+1}-1} \left[\log_2(d_\perp(p_{c_j} p_{c_{j+1}}, p_k p_{k+1})) + \log_2(d_\theta(p_{c_j} p_{c_{j+1}}, p_k p_{k+1}))\right],$$

where par_i is the number of characteristic points in trajectory i, $length(\cdot)$ is the euclidean length of a segment, $p_a p_b$ is the segment defined by points p_a and p_b and $d_\perp$ and d_θ are the perpendicular and angular distances, defined as

$$d_\perp(s_i e_i, s_j e_j) = \frac{l_{\perp 1}^2 + l_{\perp 2}^2}{l_{\perp 1} + l_{\perp 2}}$$

$$d_\theta(s_i e_i, s_j e_j) = \begin{cases} length(s_j e_j) \times \sin\theta & 0 \leq \theta < \pi/2 \\ length(s_j e_j), & \pi/2 \leq \theta \leq \pi \end{cases}$$

$l_{\perp 1}$ and $l_{\perp 2}$ are the distances illustrated in Figure 3.5.

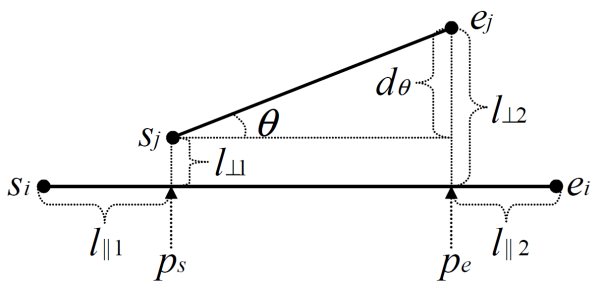


Figure 3.5: Definition of distance functions from TRACLUS. Image from [17].

The best partition is that which minimizes the MDL cost, $MDL_{cost} = L(H) + L(D|H)$. Finding the optimal partitioning, though, is computationally unfeasible, but an approximate (locally optimal) algorithm is provided in the original paper [17]. The approximate algorithm loops through all points in a trajectory and computes $MDL_{par}(p_{start}, p_{curr})$, the MDL cost assuming p_{start} and p_{curr} are the only characteristic points in the interval $p_{start}, \ldots, p_{curr}$, and $MDL_{nopar}(p_{start\_index}, p_{curr\_index})$, the MDL cost assuming there are no characteristic points in that interval (i.e., preserving the original trajectory). If $MDL_{par} \leq MDL_{nopar}$, then choosing p_{curr} as a characteristic point makes the MDL cost lower. As in our case there is no interest in the characteristic points themselves, only in their number, this procedure is adapted, as show in algorithm 1.

This algorithm is very sensitive to noise, and chooses a big number of characteristic points in noisy trajectories. Keskin [30] proposed a small modification, in an attempt to tackle this problem, but it is not very effective. As a workaround, in this thesis, the following method was used: first a "good" p was chosen. Then, splines were fitted using d set to twice the length of the trajectory, to assure its shape was well captured. Algorithm 1 was then run in the smoothed trajectories, and the spline fitting step was re-run using d equal to 5 times the largest number of characteristic points chosen.

Algorithm 1: Approximate Trajectory Partitioning.

Input: A trajectory, as a sequence of points, $p_1, \ldots, p_m$
Output: N_c, the number of characteristic points

$N_c = 1$ `// Add the starting point`
$start\_index = 1$
$length = 1$
while $start\_index + length \leq m$ **do**
  $curr\_index = start\_index + length$
  $cost_{par} = MDL_{par}(p_{start\_index}, p_{curr\_index})$
  $cost_{nopar} = MDL_{nopar}(p_{start\_index}, p_{curr\_index})$
  if $cost_{par} > cost_{nopar}$ **then**
    `// Partition at the previous point`
    $N_c = N_c + 1$
    $start\_index = curr\_index - 1$
    $length = 1$
  else
    $length = length + 1$
  end
end
$N_c = N_c + 1$ `// Add the ending point`

3.4 Summary

The pre-processing step described in this chapter takes as input the trajectories, $\{t^{(1)}, \ldots t^{(m)}\}$, the smoothing parameter, p and, optionally, the number of features, d. If d is not provided, the number of characteristic points is computed for each trajectory in the dataset, and the maximum, N_c, is chosen. Finally, each trajectory is fitted with splines and resampled with $d = 5N_c$ points, and the angles of the tangents are computed, producing the processed trajectories, $\{\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(m)}\}$.

Since this thesis focuses on trajectory clustering using shape, from this point on, whenever we mention trajectories we mean the trajectories *after* pre-processing, i.e., the set $\{\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(m)}\}$, where each trajectory $\boldsymbol{\omega}^{(i)}$ is a sequence of angles $\{\omega_1^{(i)}, \ldots, \omega_d^{(i)}\}$.

Chapter 4

K-means

The k-means algorithm is one of the most famous cluster analysis techniques available [31]. Proposed by Lloyd [32], it is a *local search* approach, offering no global optimality guarantees. In fact, it has been shown to give arbitrarily bad results in certain conditions. Despite this[1], a 2002 survey elected this over 50 year old algorithm as the most used clustering algorithm in scientific and industrial applications [34]. The two most likely reasons for this are its simplicity and speed, as will be seen in the following sections.

In this chapter, the k-means algorithm for Euclidean data is introduced. Then, following some motivation remarks, it is adapted for circular data: a new metric is defined and a way of updating the centroids is introduced.

4.1 The original algorithm

k-means is a simple and intuitive method: given a set of points and the number of desired clusters, it alternates between an assignment step and a centroid update step, until it finds a locally optimal clustering assignment. The algorithm stores a set of k centroids, $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$, and proceeds as follows:

1. In the assignment step, it sets $c^{(i)} := \arg\min_j \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_j\|^2$ for all i, where $c^{(i)} = j$ if the ith input sample belongs to cluster j. In other words, it assigns each point to the closest centroid in terms of squared Euclidean distance.

2. In the centroid update step, it sets the centroid $\boldsymbol{\mu}_j$ to $\boldsymbol{\mu}_j := \frac{\sum_{i=1}^m 1\{c^{(i)}=j\}\boldsymbol{x}^{(i)}}{\sum_{i=1}^m 1\{c^{(i)}=j\}}$ for all j, where $1\{c^{(i)} = j\}$ is an indicator function, taking the value 1 if $c^{(i)} = j$ and 0 otherwise. This moves each centroid to the mean of the points assigned to it, which justifies the name of the algorithm.

These steps are repeated until a convergence criteria is met. For example, until the cluster assignments do not change, or by looking at the variation of the *distortion function*, which k-means is proven

[1] One has to take into account that the problem of finding clusters minimizing intra-cluster distance is NP-hard, making it difficult to solve this problem efficiently [33].

to minimize locally:

$$J(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}, c^{(1)}, \ldots, c^{(m)}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k) = \sum_{i=1}^{m} \|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{c^{(i)}}\|^2 \tag{4.1}$$

Since k-means minimizes the distortion function, it is guaranteed to converge, and so the $c^{(i)}$'s and $\boldsymbol{\mu}_j$'s will also (usually) converge. However, although rare in practice, it is possible for the algorithm to oscillate indefinitely between a few different clusterings, all with the same J [35].

4.2 Measuring distances

Despite working well for many practical applications, Euclidean distance, on which the original k-means[2] is based, does not work well with angular data. For example, 0 and 2π rad are the same angle, so a distance measure between them should output 0. Obviously, this is not the case with Euclidean distance.

For k-means to work with circular data, and thus with trajectory shapes, some changes must be made. More specifically, both the distance function, or *metric*, used and the way of computing the centroids must be adapted for circular data.

There are many ways to measure distances between 2 angles. A typical one is drawing the angles in the unit circle and finding the distance between the corresponding unit vectors/points. Figure 4.1 illustrates a situation where 2 angles as represented as points A and B on the unit circle. The distance between them is given by the length of the red line, which is the chord joining A and B. This way, small differences (close vectors) lead to small chord lengths, while antipodal vectors lead to maximum chord lengths. If we call $\Delta\omega$ to the difference between both angles, the length of the chord is given by: $2\sin\frac{\Delta\omega}{2}$.

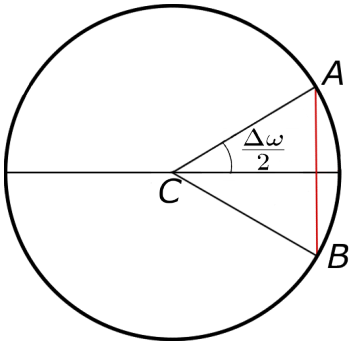


Figure 4.1: Relationship between angular difference and chord in a unit circle.

Since $\sin$ is an odd function, if $\Delta\omega$ is negative, $2\sin\frac{\Delta\omega}{2}$ will also be negative, which is not desired, as distances must be nonnegative. One way to deal with this is to take the absolute value. This, however, leads to a *non-smooth* function around $\Delta\omega = 0$, as can be seen in Figure 4.2, that treats small angular

[2]It is important to note that k-means has already been generalized to non-Euclidean metrics, in the so called k-medoids algorithm [36].

differences (around 0) differently from large ones (odd multiples of π)[3].

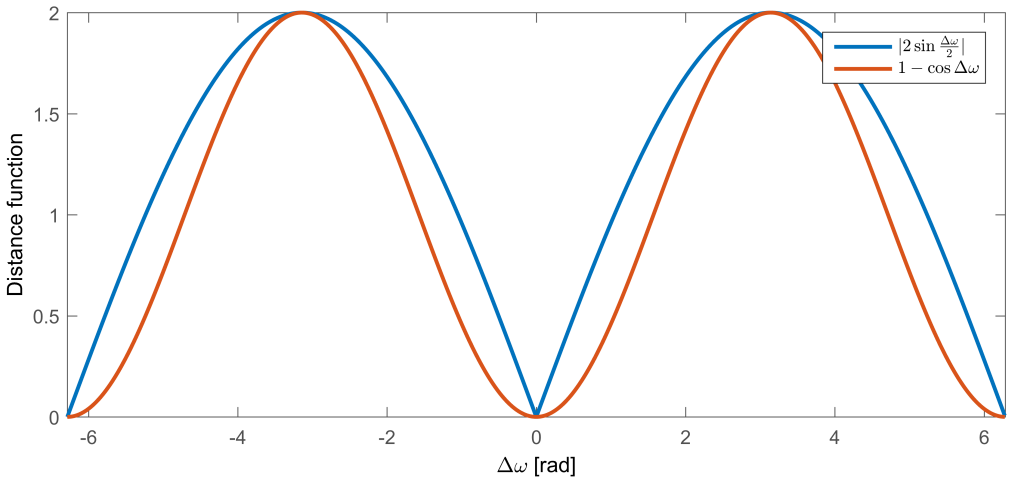


Figure 4.2: Distances between angles: using the absolute value of the chord, in blue, or $1-\cos\Delta\omega$, in red.

The kink around $\Delta\omega = 0$ can be avoided by using the square, instead of the absolute value:

$$\begin{aligned}
&\left(2\sin\frac{\Delta\omega}{2}\right)^2 = 4\sin^2\frac{\Delta\omega}{2} = 2\left(\sin^2\frac{\Delta\omega}{2}+\sin^2\frac{\Delta\omega}{2}\right)\\
&= 2\left(\sin^2\frac{\Delta\omega}{2}+1-\cos^2\frac{\Delta\omega}{2}\right) = 2\left(1-\left(\cos^2\frac{\Delta\omega}{2}-\sin^2\frac{\Delta\omega}{2}\right)\right)\\
&= 2\left(1-\cos\Delta\omega\right) \propto 1-\cos\Delta\omega,
\end{aligned}$$

where the second to last equality used the trigonometric identity $\cos\alpha = \cos^2\frac{\alpha}{2} - \sin^2\frac{\alpha}{2}$. Since this function will only be used to compare distances, the constant "2" is irrelevant, and can be dropped. Thus, the final metric is $1-\cos\Delta\omega$, a "smooth" function (see Figure 4.2), as opposed to the absolute value of the chord.

This is not the only way to measure distances between angular quantities. One other approach is the length of the shorter arc [37]:

$$D(\Delta\omega) = min\left(|\Delta\omega|, 2\pi - |\Delta\omega|\right).$$

In this work, both metrics were tested. Since no difference was found in the results, $1-\cos\Delta\omega$ was chosen, as it seems a more "natural" choice.

Until now, we were only concerned with scalar data. The extension to d-dimensional data, such as shapes of trajectories, is trivial:

$$D^2(\boldsymbol{\omega^{(1)}}, \boldsymbol{\omega^{(2)}}) = \sum_{p=1}^{d}\left(1-\cos(\omega_p^{(1)}-\omega_p^{(2)})\right), \tag{4.2}$$

where the $\boldsymbol{\omega^{(i)}}$'s are angular vectors and $\omega_p^{(i)}$ is the pth component of vector i. The notation D^2 is used

[3]This function is periodic, so the same reasoning applies for multiples of 2π

to indicate this metric is a squared value (of the chord).

4.3 Updating the centroids

The centroid update step in k-means must also be adapted for circular data. The arithmetic mean can not be used, due to the periodic nature of circular data. For example, the arithmetic mean of 0 and 2π rad is π, clearly a bad choice. Also, any measure of mean must be *independent of the origin of the angular coordinates*, i.e., the mean must give the same result (up to a multiple of 2π) independently of the data being in the interval $[0, 2\pi[$ or any other, which does not hold for the Euclidean mean.

One typical solution in the literature [1] is the following: given a sequence of angles, $\{\omega^{(1)}, \ldots, \omega^{(m)}\}$, they can be viewed as a set of unit vectors, where vector i is $(\cos\omega^{(i)}, \sin\omega^{(i)})$. The average of these vectors, $\bar{\boldsymbol{x}}$, does not depend on the origin of the coordinates, and can be used to find the mean of the angles, $\bar{\omega}$:

$$\bar{\boldsymbol{x}} = \frac{1}{m}\sum_{i=1}^{m}(\cos\omega^{(i)}, \sin\omega^{(i)}) = \left(\frac{1}{m}\sum_{i=1}^{m}\cos\omega^{(i)}, \frac{1}{m}\sum_{i=1}^{m}\sin\omega^{(i)}\right).$$

Defining the mean of the angles as the angle corresponding to the mean vector, we obtain[4]

$$\bar{\omega} = \mathrm{atan2}\left(\frac{\frac{1}{m}\sum_{i=1}^{m}\sin\omega^{(i)}}{\frac{1}{m}\sum_{i=1}^{m}\cos\omega^{(i)}}\right) \tag{4.3}$$

It can be shown that this formula for the mean is the maximum likelihood estimator of the mean of a set of angles following a Von Mises distribution [1]. Given the duality between $2D$ vectors and complex numbers, $\bar{\omega}$ can be computed in a more direct way:

$$\bar{\omega} = \arg\left(\frac{1}{m}\sum_{i=1}^{m}e^{j\omega^{(i)}}\right), \tag{4.4}$$

where $\arg$ returns the argument of a complex number.

The extension of this result to vectorial data is straightforward: the mean of an angular vector is the angular mean of each of its coordinates.

[4] Once again, $\mathrm{atan2}$ refers to the 4-quadrant inverse tangent function.

4.4 Algorithm Description

Using the formulas derived in previous sections, k-means can be adapted for circular data, producing the following algorithm:

Algorithm 2: $circular\ k-means$ algorithm.

Input: Data set, $\{\boldsymbol{\omega^{(1)}}, \ldots, \boldsymbol{\omega^{(m)}}\}$, and number of clusters, k

Output: Cluster centroids, $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$, and cluster assignments, $c^{(1)}, \ldots, c^{(m)}$

Initialize k cluster centroids: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$

repeat

  foreach *i* **do**

    Set $c^{(i)} := \arg\min_j D(\boldsymbol{\omega^{(i)}}, \boldsymbol{\mu_j})$ `// Assign centroids to samples`

  end

  foreach *j* **do**

    Set $\mu_j^p = \arg\left(\frac{\sum_{i=1}^m 1\{c^{(i)}=j\} e^{j\omega_p^{(i)}}}{\sum_{i=1}^m 1\{c^{(i)}=j\}}\right)$ for $p = 1, \ldots, d$ `// Update each coordinate of the centroids`

  end

until *convergence*

4.4.1 Initialization and stopping criteria

For the algorithm to be completely described, initialization and stopping criteria must be specified.

One of the most critical issues with k-means is initialization. When the algorithm starts, it needs k cluster centroids to be set. A typical way of initializing them is to randomly select k samples from the training data and use them as the centroids. The main disadvantage of this approach is that it can be shown to lead to arbitrarily poor clustering results [38]. The reason for this is that the algorithm may get stuck at a local minimum. Since k-means' results are heavily dependent on the initial centroids, leaving the initialization purely to chance seems a risky option.

Since the problem is NP-hard, an optimal initialization procedure would require exponential time[5]. Loosening the optimality requirement, fast initialization methods with good theoretical guarantees have been devised, such as k-means++ [38]. Better guarantees were found for other methods, but at the expense of a much slower initialization.

The idea behind a good initialization is to choose an initial set of clusters as spread-out as possible. k-means++ accomplishes this goal by choosing centroids randomly from training data, but with weights according to their distance to other centroids. It is fast (linear in k) and it can be shown that the converged solution is at most $\log k$ worse than the optimal. The exact procedure is as follows:

1. Choose the first centroid, $\boldsymbol{\mu}_1$ uniformly at random from the training data.
2. Take a new center $\boldsymbol{\mu}_i$ from the data with probability $\frac{D^2(\boldsymbol{\omega})}{\sum_{\boldsymbol{\omega}' \in \Omega} D^2(\boldsymbol{\omega}')}$. Ω is the set of all the training trajectories.

[5] Unless P=NP, but this is another topic, beyond the scope of this work.

3. Repeat previous step until all clusters have been chosen.

After this, k-means is run until the assignments no longer change between two iterations[6]. To avoid the (extremely rare) possibility of the algorithm oscillating between assignments, the number of steps is limited to 100.

4.5 Model Selection

Until now, the number of clusters, k, was assumed to be known and given as an input to the algorithm. In real-world situations, though, this is often not the case, and it is necessary to estimate k.

One common method used in conjunction with k-means is the so-called elbow method. This method involves plotting the distortion function for several values of k. This function will always decrease with the number of clusters, but after a certain point, hopefully the right k, the rate of decrease will get significantly smaller. Such a point is called an elbow, and is illustrated in Figure 4.3.

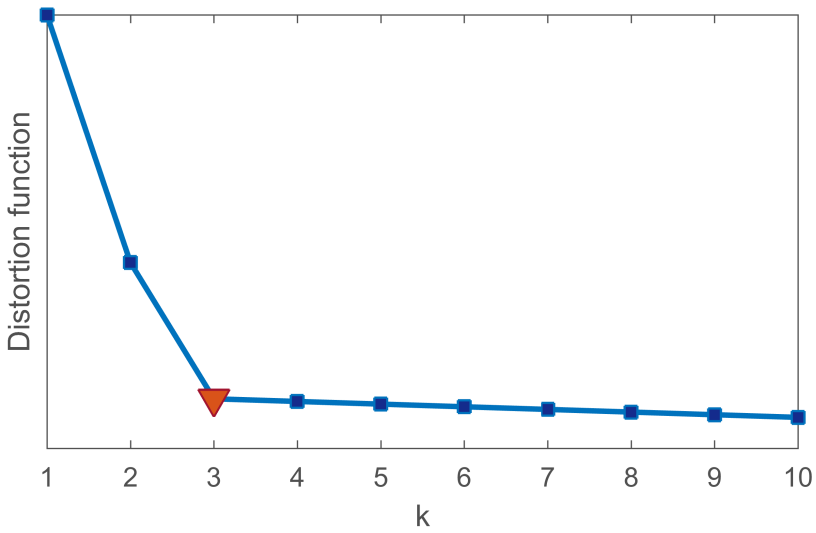


Figure 4.3: Example of a elbow in a plot of the distortion function. In this case, there is a clear elbow at $k = 3$, indicating there may be 3 clusters in the dataset.

The distortion function is just the sum of the distances of each data sample to the respective centroid. In the case of "regular" k-means, this is given by Eq. (4.1). For circular k-means, it is

$$J(\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(m)}, c^{(1)}, \ldots, c^{(m)}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k) = \sum_{i=1}^{m} \sum_{p=1}^{d} \left(1 - \cos(\omega_p^{(i)} - \mu_{c^{(i)}}^p)\right). \tag{4.5}$$

A disadvantage of this approach is that an elbow is not guaranteed to exist, and so model selection using this criteria may not be possible. Another problem is that it is difficult to say what is an elbow and what is not.

[6]This was done in all the tests performed in the Results chapter.

Chapter 5

Finite Mixture Models

In many practical applications, a single unimodal distribution does not capture well enough the intricacies of data, such as it being concentrated in various groups. This is well illustrated in Figure 5.1, which shows a dataset that no unimodal distribution can properly fit, since there are two “high-concentration” regions.

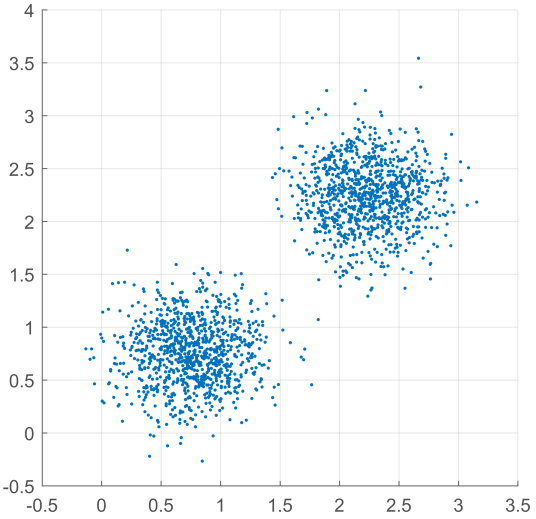


Figure 5.1: Data set with two clumps, making any unimodal distribution a poor fit.

Finite mixture models (FMM), or finite mixtures, address this shortcoming, modeling data using a convex combination of multiple unimodal distributions, thus allowing multiple modes. They are powerful probabilistic tools for modeling possibly multi-dimensional data, and their applications include many areas, such as computer vision, signal analysis, machine learning, economics, and bioinformatics.

Finite mixtures allow a formal treatment of clustering, but with a little twist in relation with k-means. In k-means, each sample can only belong to one cluster, corresponding to what is called *hard clustering*. Finite mixtures, on the other hand, perform *soft clustering*, allowing a sample to belong to multiple clusters, assigning a probability to each case. Another difference is that FMM’s are *generative models*, that is, they model the data generation process in order to categorize it, while k-means and matrix fac-

torization (introduced in the following chapter) are *discriminative methods* that categorize data directly.

5.1 Mathematical Formulation

Mixture models make use of *latent variables*, i.e., variables that are not directly observable from the training set, but that can be inferred from it. Given a d-dimensional training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$, which is to be grouped in k clusters, a latent variable $z^{(i)}$ may be defined for each sample $\boldsymbol{x}^{(i)}$, taking values from 1 to k, with $z^{(i)} = j$ meaning sample $\boldsymbol{x}^{(i)}$ belongs to cluster j. The $z^{(i)}$'s are assumed to follow a multinomial distribution, $z^{(i)} \sim \text{Multinomial}(\boldsymbol{\alpha})$, which define the mixture probabilities:

$$p(z^{(i)} = j) = \alpha_j,$$

where $\sum_{j=1}^{k} \alpha_j = 1$ and $\alpha_j \geq 0,\ \forall j = 1, \ldots, k$.

We can now model the distribution of $\boldsymbol{x}^{(i)}$ using conditional probabilities:

$$p(\boldsymbol{x}^{(i)}) = \sum_{j=1}^{k} p(\boldsymbol{x}^{(i)} | z^{(i)} = j) p(z^{(i)} = j).$$

This formula is general and independent of the probability distribution of the $\boldsymbol{x}^{(i)}$'s given the $z^{(i)}$'s. By choosing $p(\boldsymbol{x}^{(i)}|z^{(i)})$, different kinds of mixtures are obtained. These distributions are characterized by a set of parameters, usually denoted $\boldsymbol{\theta}$, whereby this pdf is written as $p(x^{(i)}|z^{(i)}; \boldsymbol{\theta})$. A very common choice is the normal distribution, with the resulting model called Gaussian mixture model (GMM), which is probably the most often used type of FMM. A GMM, composed of 3 univariate Gaussians with different means and variances is exemplified in Figure 5.2.

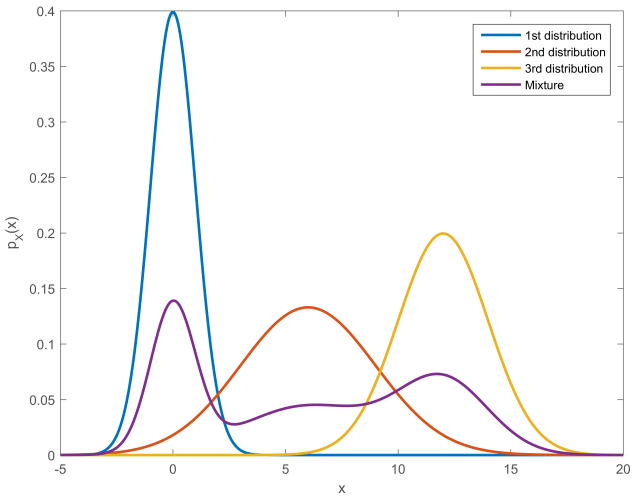


Figure 5.2: 3 univariate Gaussians, with different means and covariances and their mixture, using $\alpha_1 = \alpha_2 = \alpha_3 = {}^1/_3$. As can be observed, in this case the final mixture has 3 modes, one for each univariate distribution. It is important to note, however, that a k component GMM may have less than k modes.

Assuming the samples from the training data set are independent and identically distributed (i.i.d.),

the joint pdf, also referred as the likelihood function, can be written as

$$p(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}|\boldsymbol{\alpha}, \boldsymbol{\theta}) = \prod_{i=1}^{m} p(\boldsymbol{x}^{(i)}|\boldsymbol{\alpha}, \boldsymbol{\theta}) = \prod_{i=1}^{m} \sum_{j=1}^{k} \alpha_j p(\boldsymbol{x}^{(i)}|z^{(i)} = j, \boldsymbol{\theta}). \tag{5.1}$$

The sum over j, necessary since the $z^{(i)}$'s are unknown, makes the estimation problem very challenging to carry out directly. This is typically solved using the EM (expectation-maximization) algorithm, which provides an iterative locally-optimal procedure to maximize the likelihood function with respect to $\boldsymbol{\theta}$ and $\alpha_1, \ldots, \alpha_k$ [39].

5.2 Mixtures of Von Mises

Assume a trajectory $\boldsymbol{\omega}$ can be written as a mean trajectory, $\boldsymbol{\mu}$, corrupted by noise, $\boldsymbol{\epsilon}$, i.e., $\boldsymbol{\omega} = \boldsymbol{\mu} + \boldsymbol{\epsilon}$. If $\boldsymbol{\epsilon}$ follows a multivariate sine Von Mises distribution, then a set of trajectories $\{\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(m)}\}$ can be modeled as a mixture of multivariate Von Mises:

$$p(\boldsymbol{\omega}|\boldsymbol{\theta}) = \sum_{j=1}^{k} \alpha_j p(\boldsymbol{\omega}^{(i)}|z^{(i)} = j; \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j, \boldsymbol{\Lambda}_j), \tag{5.2}$$

where k is the number of clusters, $\boldsymbol{\mu}_j$, $\boldsymbol{\kappa}_j$, and $\boldsymbol{\Lambda}_j$ are, respectively, the mean, concentration parameter and the matrix measuring the dependences among variables of distribution j.

5.3 Expectation-Maximization Algorithm

Latent variable models introduce some challenges in the statistical inference of their parameters. Since latent variables are not observed, ML and MAP estimation can not be carried out directly. This is where the EM algorithm comes in: it provides an iterative procedure, with two steps, expectation (E-step) and maximization (M-step), to infer the statistical parameters of latent variable models. In the E-step, the algorithm attempts to "guess" the values of the latent variables for each input sample. In the M-step, it updates the parameters of the model based on the guesses made in the E-step. This procedure has been shown to always monotonically improve the log-likelihood/log-posterior. There are many different, yet equivalent, ways of deriving the EM equations. In this work, we follow [35].

Suppose there is a set of data, $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with which a latent variable model is to be fitted. The EM algorithm works as follows:

1. E-step: for all i, compute $Q_i(z^{(i)}) = p(z^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\theta})$.
2. M-step: update the estimates of the parameter vector, $\boldsymbol{\theta}$: $\boldsymbol{\theta} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\arg\max} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \ln \frac{p(x^{(i)}, z^{(i)}, \boldsymbol{\theta})}{Q_i(z^{(i)})}$.

Above, $Q_i(z^{(i)})$ is the conditional probability function of the latent variables of sample i and Θ is the parameter space. This works for both ML and MAP estimation. These two steps are repeated until convergence.

We now proceed to derive the EM equations for a Von Mises Mixture (VMM) algorithm, using ML estimation.

5.4 E-Step

In the case of a Von Mises FMM, the E-step is straightforward:

$$w_j^{(i)} = Q_i(z^{(i)} = j) = p(z^{(i)} = j|\boldsymbol{\omega}^{(i)}; \boldsymbol{\alpha}, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k, \boldsymbol{\kappa}_1, \ldots, \boldsymbol{\kappa}_k, \boldsymbol{\Lambda}_1, \ldots, \boldsymbol{\Lambda}_k) \tag{5.3}$$

From this point on, to simplify notation, $\boldsymbol{M}$, $\boldsymbol{K}$, and $\boldsymbol{L}$ will be used in place of $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$, $\{\boldsymbol{\kappa}_1, \ldots, \boldsymbol{\kappa}_k\}$, and $\{\boldsymbol{\Lambda}_1, \ldots, \boldsymbol{\Lambda}_k\}$, respectively.

From Bayes' Theorem:

$$p(z^{(i)} = j|\boldsymbol{\omega}^{(i)}; \boldsymbol{\alpha}, \boldsymbol{M}, \boldsymbol{K}, \boldsymbol{L}) = \frac{p(\boldsymbol{\omega}^{(i)}|z^{(i)} = j; \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j, \boldsymbol{\Lambda}_j)p(z^{(i)} = j; \boldsymbol{\alpha})}{p(\boldsymbol{\omega}^{(i)}; \boldsymbol{M}, \boldsymbol{K}, \boldsymbol{L})},$$

and so the E-step reduces to computing, for every i and j,

$$w_j^{(i)} = \frac{\alpha_j p(\boldsymbol{\omega}^{(i)}|z^{(i)} = j; \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j, \boldsymbol{\Lambda}_j)}{\sum_{l=1}^{k} \alpha_l p(\boldsymbol{\omega}^{(i)}|z^{(i)} = l; \boldsymbol{\mu}_l, \boldsymbol{\kappa}_l, \boldsymbol{\Lambda}_l)}, \tag{5.4}$$

where $p(\boldsymbol{\omega}^{(i)}|z^{(i)} = j; \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j, \boldsymbol{\Lambda}_j)$ is a multivariate Von Mises pdf.

5.5 M-Step

The derivation of the equations for the M-step is significantly more tedious. The following function needs to be maximized with respect to the parameters of the model, $\boldsymbol{\alpha}, \boldsymbol{M}, \boldsymbol{K}, \boldsymbol{L}$:

$$\begin{aligned}
&\sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \ln \frac{p(\boldsymbol{\omega}^{(i)}, z^{(i)}; \boldsymbol{\alpha}, \boldsymbol{M}, \boldsymbol{K}, \boldsymbol{L})}{Q_i(z^{(i)})} \\
=&\sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} \ln \frac{p(\boldsymbol{\omega}^{(i)}|z^{(i)} = j; \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j, \boldsymbol{\Lambda}_j)p(z^{(i)} = j; \boldsymbol{\alpha})}{w_j^{(i)}} \\
=&\sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} \ln \frac{T(\boldsymbol{\kappa}_j, \boldsymbol{\Lambda}_j)^{-1} \exp\left(\boldsymbol{\kappa}_j^T c(\boldsymbol{\omega}^{(i)}, \boldsymbol{\mu}_j) + \frac{1}{2} s(\boldsymbol{\omega}^{(i)}, \boldsymbol{\mu}_j)^T \boldsymbol{\Lambda}_j \, s(\boldsymbol{\omega}^{(i)}, \boldsymbol{\mu}_j)\right) \alpha_j}{w_j^{(i)}}.
\end{aligned} \tag{5.5}$$

As stated in section 2.4.2, $T(\boldsymbol{\kappa}_j, \boldsymbol{\Lambda}_j)$ is a normalizing constant unknown in explicit form for $d > 2$. This presents a challenge for ML or MAP estimation, and it requires a numerical optimization method to be solved. Some of these methods are maximum pseudolikelihood or the method of moments [27].

Working with these methods in high-dimensional spaces is delicate, and a simplifying assumption is made to avoid it: each feature of a trajectory, that is, each component of the $\boldsymbol{\omega}^{(i)}$ vector, is assumed to be *independent* of the others. This means that all $\boldsymbol{\Lambda}_j$ matrices are null matrices, and the multivariate Von Mises distribution is equivalent to the product of d (the number of features) univariate Von Mises

distributions:

$$p(\boldsymbol{\omega}^{(i)}|z^{(i)}=j,\boldsymbol{\mu}_j,\boldsymbol{\kappa}_j,\boldsymbol{\Lambda}_j)=\prod_{p=1}^{d}\frac{e^{\kappa_j^p\cos(\omega_p^{(i)}-\mu_j^p)}}{2\pi I_0(\kappa_j^p)}, \tag{5.6}$$

where μ_j^p, κ_j^p, and $\omega_p^{(i)}$ refer to the pth component of vectors $\boldsymbol{\mu}_j$, $\boldsymbol{\kappa}_j$ and $\boldsymbol{\omega}^{(i)}$, respectively[1]. This leads to $T(\boldsymbol{\kappa}_j,\boldsymbol{\Lambda}_j)=T(\boldsymbol{\kappa}_j)=\prod_{p=1}^{d}2\pi I_0(\kappa_j^p)$, allowing the direct use of ML and MAP methods.

Substituting Eq. (5.6) into Eq. (5.5), we finally get

$$\sum_{i=1}^{m}\sum_{j=1}^{k}w_j^{(i)}\left(\sum_{p=1}^{d}\left[-\ln(2\pi I_0(\kappa_j^p))+\ln e^{\kappa_j^p\cos(\omega_p^{(i)}-\mu_j^p)}\right]+\ln\alpha_j-\ln w_j^{(i)}\right). \tag{5.7}$$

5.5.1 Estimation of $\boldsymbol{\mu}$

The maximization with respect to μ_l^q (qth component of $\boldsymbol{\mu}_l$) is straightforward. Taking partial derivatives,

$$\begin{aligned}
&\frac{\partial}{\partial\mu_l^q}\left(\sum_{i=1}^{m}\sum_{j=1}^{k}w_j^{(i)}\sum_{p=1}^{d}\left[\ln(2\pi I_0(\kappa_j^p))+\ln e^{\kappa_j^p\cos(\omega_p^{(i)}-\mu_j^p)}+\ln\alpha_j-\ln w_j^{(i)}\right]\right)\\
=&\frac{\partial}{\partial\mu_l^q}\left(\sum_{i=1}^{m}\sum_{j=1}^{k}w_j^{(i)}\sum_{p=1}^{d}\left[-\ln e^{\kappa_j^p\cos(\omega_p^{(i)}-\mu_j^p)}\right]\right)\\
=&\sum_{i=1}^{m}w_l^{(i)}\frac{\partial}{\partial\mu_l^q}\left[\kappa_l^q\cos(\omega_q^{(i)}-\mu_l^q)\right]\\
=&-\kappa_l^q\sum_{i=1}^{m}w_l^{(i)}\sin\left(\omega_q^{(i)}-\mu_l^q\right).
\end{aligned} \tag{5.8}$$

Setting the previous expression to 0 and using the identity $\sin(\alpha-\beta)=\sin\alpha\cos\beta-\cos\alpha\sin\beta$:

$$\begin{aligned}
&-\kappa_l^q\sum_{i=1}^{m}w_l^{(i)}\sin\left(\omega_q^{(i)}-\mu_l^q\right)=0\Leftrightarrow\sum_{i=1}^{m}w_l^{(i)}\left(\sin\omega_q^{(i)}\cos\mu_l^q-\cos\omega_q^{(i)}\sin\mu_l^q\right)=0\\
\Leftrightarrow&\sin\mu_l^q\sum_{i=1}^{m}w_l^{(i)}\cos\omega_q^{(i)}=\cos\mu_l^q\sum_{i=1}^{m}w_l^{(i)}\sin\omega_q^{(i)}\Leftrightarrow\tan\mu_l^q=\frac{\sum_{i=1}^{m}w_l^{(i)}\sin\omega_q^{(i)}}{\sum_{i=1}^{m}w_l^{(i)}\cos\omega_q^{(i)}}.
\end{aligned} \tag{5.9}$$

Consequently the update rule for μ_l^q is

$$\mu_l^q=\operatorname{atan2}\left(\frac{\sum_{i=1}^{m}w_l^{(i)}\sin\omega_q^{(i)}}{\sum_{i=1}^{m}w_l^{(i)}\cos\omega_q^{(i)}}\right). \tag{5.10}$$

5.5.2 Estimation of $\boldsymbol{\alpha}$

Grouping together the terms of Eq. (5.7) that depend on α_j, we get

$$\sum_{i=1}^{m}\sum_{j=1}^{k}w_j^{(i)}\ln\alpha_j.$$

[1] This notation may get confusing, but it is the standard in literature.

In this case, there is an additional constraint: the α_j's must sum to 1. To deal with it, the Lagrangian is introduced:

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\sum_{j=1}^{k} w_j^{(i)} \ln \alpha_j + \beta \left(\sum_{j=1}^{k} \alpha_j - 1 \right), \tag{5.11}$$

where β is the Lagrange multiplier. Taking derivatives with respect to α_l:

$$\frac{\partial \mathcal{L}(\boldsymbol{\alpha})}{\partial \alpha_l} = \sum_{i=1}^{m} \frac{w_l^{(i)}}{\alpha_l} + \beta$$

Setting to zero and solving, we get: $\alpha_l = -\frac{\sum_{i=1}^{m} w_l^{(i)}}{\beta}$. Since, by constraint, the α_j's must sum to 1:

$$\sum_{j=1}^{k} \alpha_j = \sum_{j=1}^{k} -\frac{\sum_{i=1}^{m} w_j^{(i)}}{\beta} = -\sum_{i=1}^{m} \frac{\overbrace{\sum_{j=1}^{k} w_l^{(i)}}^{=1}}{\beta} = -\frac{1}{\beta}\sum_{i=1}^{m} 1 = 1 \Leftrightarrow \beta = -m,$$

where we used the fact $\sum_{j=1}^{k} w_l^{(i)}$ must be 1, since it is a probability distribution. Substituting this in the formula for α_l we get

$$\alpha_l = \frac{1}{m}\sum_{i=1}^{m} w_l^{(i)}. \tag{5.12}$$

Another condition that must be satisfied is $\alpha_l \geq 0$. In this case, even though this condition was not explicitly included in the Lagrangian, it automatically holds, as the $w_l^{(i)}$'s are probabilities (and hence, non-negative).

5.5.3 Estimation of $\boldsymbol{\kappa}$

For the M-step to be completely specified, an update rule for $\boldsymbol{\kappa}_l$ must be derived. Rearranging Eq. (5.7) and dropping the terms that do not depend on $\boldsymbol{\kappa}_j$, we get

$$\sum_{i=1}^{m}\sum_{j=1}^{k} w_j^{(i)} \sum_{p=1}^{d} \left[-\ln(2\pi I_0(\kappa_j^p)) + \kappa_j^p \cos(\omega_p^{(i)} - \mu_j^p) \right].$$

Computing the derivative with respect to the qth component of $\boldsymbol{\kappa}_l$, we get

$$\sum_{i=1}^{m} w_l^{(i)} \frac{\partial}{\partial \kappa_l^q} \left(-\ln(2\pi I_0(\kappa_l^q)) + \kappa_l^q \cos(\omega_q^{(i)} - \mu_l^q) \right) = \sum_{i=1}^{m} w_l^{(i)} \left(-\frac{1}{I_0(\kappa_l^q)} \frac{\partial I_0(\kappa_l^q)}{\partial \kappa_l^q} + \cos(\omega_q^{(i)} - \mu_l^q) \right). \tag{5.13}$$

$$\frac{\partial I_0(\kappa_l^q)}{\partial \kappa_l^q} = \frac{\partial}{\partial \kappa_l^q} \left(\frac{1}{2\pi} \int_0^{2\pi} e^{\kappa_l^q \cos\omega} d\omega \right) = \frac{1}{2\pi} \int_0^{2\pi} \frac{\partial e^{\kappa_l^q \cos\omega}}{\partial \kappa_l^q} d\omega = \frac{1}{2\pi} \int_0^{2\pi} \cos\omega \cdot e^{\kappa_l^q \cos\omega} d\omega = I_1(\kappa_l^q),$$

where I_1 denotes the modified Bessel function of the first kind and order 1. Substituting this result in Eq.

(5.13) and setting to 0:

$$\sum_{i=1}^{m} w_l^{(i)} \left(-\frac{I_1(\kappa_l^q)}{I_0(\kappa_l^q)} + \cos(\omega_q^{(i)} - \mu_l^q) \right) = 0 \Leftrightarrow \frac{I_1(\kappa_l^q)}{I_0(\kappa_l^q)} \sum_{i=1}^{m} w_l^{(i)} = \sum_{i=1}^{m} w_l^{(i)} \cos(\omega_q^{(i)} - \mu_l^q).$$

Finally, defining function $A_1(\cdot) = \frac{I_1(\cdot)}{I_0(\cdot)}$ and its inverse, $A_1^{-1}(\cdot)$, the update rule for κ_l^q is

$$\kappa_l^q = A_1^{-1} \left(\frac{\sum_{i=1}^{m} w_l^{(i)} \cos(\omega_q^{(i)} - \mu_l^q)}{\sum_{i=1}^{m} w_l^{(i)}} \right) \tag{5.14}$$

No analytical expression exists for A_1^{-1}, but it can be closely approximated by the following expression [40]:

$$A_1^{-1}(\bar{R}) = \begin{cases} 2\bar{R} + \bar{R}^3 + \frac{5}{6}\bar{R}^5 & , \quad 0 \leq \bar{R} < 0.53 \\ -0.4 + 1.39\bar{R} + \frac{0.43}{1-\bar{R}} & , \quad 0.53 \leq \bar{R} < 0.85 \\ \frac{1}{\bar{R}^3 - 4\bar{R}^2 + 3\bar{R}} & , \quad 0.85 \leq \bar{R} < 1. \end{cases} \tag{5.15}$$

5.6 Constraining $\boldsymbol{\kappa}_j$

Similarly to what is sometimes done with mixtures of Gaussians, where the covariance matrices are constrained to be diagonal, it may be of interest to reduce the freedom of the mixture model, to help avoid *overfitting* in problems with small datasets. This can be accomplished by forcing all components of each $\boldsymbol{\kappa}_j$ vector to be the equal, i.e, forcing the same κ in each dimension.

To derive the M-step update rule for $\boldsymbol{\kappa}_l$ with this constraint, we take Eq. (5.7) and set $\kappa_j = \kappa_j^1 = \kappa_j^2 = \ldots = \kappa_j^d$:

$$\begin{aligned} &\sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} \left(\sum_{p=1}^{d} \left[-\ln(2\pi I_0(\kappa_j)) + \ln e^{\kappa_j \cos(\omega_p^{(i)} - \mu_j^p)} \right] + \ln \alpha_j - \ln w_j^{(i)} \right) \\ &\propto \sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} \left(-d \ln(2\pi I_0(\kappa_j)) + \kappa_j \sum_{p=1}^{d} \cos(\omega_p^{(i)} - \mu_j^p) \right), \end{aligned}$$

where the terms independent of κ_j were dropped as they don't influence the maximization. Differentiating and equating to zero,

$$\begin{aligned} &\sum_{i=1}^{m} w_l^{(i)} \frac{\partial}{\partial \kappa_l} \left(-d \ln(2\pi I_0(\kappa_l)) + \kappa_l \sum_{p=1}^{d} \cos(\omega_p^{(i)} - \mu_l^p) \right) = 0 \\ \Leftrightarrow\ & dA_1(\kappa_l) \sum_{i=1}^{m} w_l^{(i)} = \sum_{i=1}^{m} w_l^{(i)} \sum_{p=1}^{d} \cos(\omega_p^{(i)} - \mu_l^p) = \sum_{p=1}^{d} \sum_{i=1}^{m} w_l^{(i)} \cos(\omega_p^{(i)} - \mu_l^p) \\ \Leftrightarrow\ & \kappa_l = A_1^{-1} \left(\frac{1}{d} \sum_{p=1}^{d} \frac{\sum_{i=1}^{m} w_l^{(i)} \cos(\omega_p^{(i)} - \mu_l^p)}{\sum_{i=1}^{m} w_l^{(i)}} \right) \end{aligned} \tag{5.16}$$

yields the expression for a constrained κ_l[2]. It is interesting to note that the argument of A_1^{-1} in Eq. (5.16) is nothing more than the mean of what would have been the arguments of A_1^{-1} for each dimension in

[2]When we write κ_l, we refer to any component of the $\boldsymbol{\kappa}_l$ vector, as they are all the same.

the unconstrained case.

From this point on, to distinguish between VMM with unconstrained and constrained $\boldsymbol{\kappa}_l$'s, we call the former unconstrained VMM, and the latter constrained VMM.

5.7 Prior on the concentration parameter

Up to this point, ML estimation was used. Though it works well for the α_j's and $\boldsymbol{\mu}_j$'s, it is problematic for the $\boldsymbol{\kappa}_j$'s, due to the asymptotic behavior of A_1^{-1}, which goes to infinity as its argument approaches 1. This phenomenon, illustrated by the blue curve in Figure 5.3, causes numerical issues and the algorithm may fail to converge.

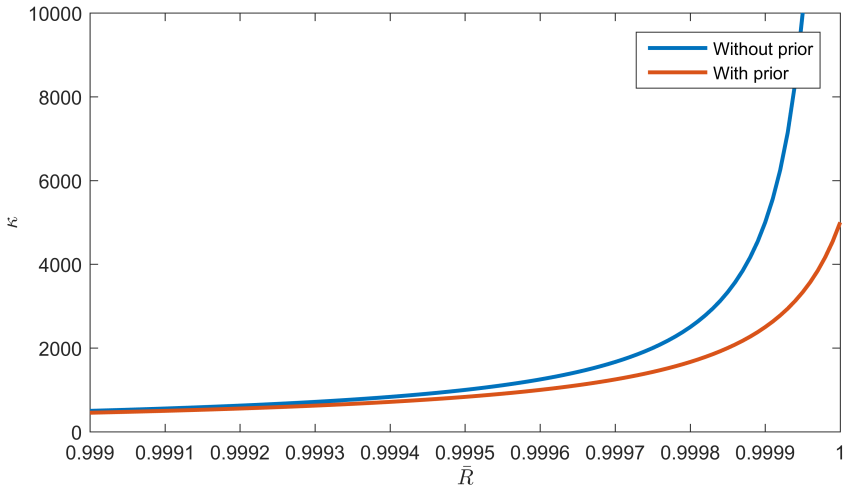


Figure 5.3: Estimation of κ, with (red) and without (blue) a prior. The blue curve explodes in the vicinity of 1, leading to numerical issues.

One way to deal with this is to limit the range of values the $\boldsymbol{\kappa}_j$'s can take, which can be accomplished with MAP estimation. Instead of maximizing the likelihood, we maximize the posterior pdf, which allows the introduction of *prior information* about the parameters, thus preventing the $\boldsymbol{\kappa}_j$'s from exploding.

Using a prior introduces information about a parameter. This way, we can use inference as if we had a larger dataset. When choosing a prior, that is, when choosing how to encode prior information on the inference problem, there are two main goals: it must adequately describe the knowledge of the unknown parameters and it should produce an analytically tractable posterior. The first goal can be achieved by any distribution that attributes lower probabilities to high $\boldsymbol{\kappa}_j$'s. The second, using a *conjugate prior* [41].

Conjugate priors are statistical distributions that guarantee a closed-form expression for the posterior. By choosing a prior conjugate to the likelihood function, a complicated calculation is transformed in an "easy" one. The conjugate prior for the concentration parameter of a univariate Von Mises distribution is [41]

$$p(\kappa; c, R_0) = \begin{cases} \frac{1}{K} \frac{e^{\kappa R_0}}{I_0(\kappa)^c} & , \quad \kappa > 0 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{5.17}$$

where $K = \int_0^\infty \frac{e^{\kappa R_0}}{I_0(\kappa)^c}$ is a normalizing constant, $c > 0$, and $R_0 \leq c$ are parameters that control the shape

of the prior. If R_0 is negative, higher probabilities are attributed to lower κ's, while the opposite is true when R_0 is positive. The c parameter also controls the likelihood attributed to high κ's: the higher c, the lower the probability attributed to high κ's. With the correct choice of their values, a behavior similar to the curve in red in Figure 5.3 can be achieved, thus limiting the maximum κ and solving the numerical issues.

We now adapt the EM algorithm to use a posterior. In the E-step, nothing is changed, but some modifications are needed in the M-step: instead of $p(\boldsymbol{\omega}^{(i)}, z^{(i)} = j; \boldsymbol{\alpha}, \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j)$, we maximize

$$p(\boldsymbol{\alpha}, \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j | \boldsymbol{\omega}^{(i)}, z^{(i)} = j) = \beta \; p(\boldsymbol{\omega}^{(i)}, z^{(i)} = j; \boldsymbol{\alpha}, \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j) p(\boldsymbol{\kappa}_j),$$

with normalizing constant β so it integrates to 1 and $p(\boldsymbol{\kappa}_j) = \prod_{p=1}^{d} p(\kappa_j^p; c, R_0)$. This assumes that all $\boldsymbol{\kappa}_j$'s, as well as their components are independent. It also assumes, for simplicity, the same c and R_0 for all κ_j^p's.

Substituting the previous equation in Eq. (5.5) gives

$$\sum_{i=1}^{m} \sum_{j=1}^{k} w_j^{(i)} \left(\ln \beta + \sum_{p=1}^{d} \left[-\ln(2\pi I_0(\kappa_j^p)) + \kappa_j^p \cos(\omega_p^{(i)} - \mu_j^p) + \kappa_j^p R_0 - c \ln I_0(\kappa_j^p) \right] + \ln \alpha_j - \ln w_j^{(i)} - \ln K \right). \tag{5.18}$$

Dropping the terms independent of κ_j^p[3], computing derivative and setting to 0:

$$\begin{aligned}
&\sum_{i=1}^{m} w_l^{(i)} \frac{\partial}{\partial \kappa_l^q} \left(-\ln(2\pi I_0(\kappa_l^q)) + \kappa_l^q \cos(\omega_q^{(i)} - \mu_l^q) + \kappa_l^q R_0 - c \ln I_0(\kappa_l^q) \right) = 0 \\
\Leftrightarrow & - A_1(\kappa_l^q) \sum_{i=1}^{m} w_l^{(i)} + \sum_{i=1}^{m} w_l^{(i)} \cos(\omega_q^{(i)} - \mu_l^q) + R_0 \sum_{i=1}^{m} w_l^{(i)} - c A_1(\kappa_l^q) \sum_{i=1}^{m} w_l^{(i)} = 0 \\
\Leftrightarrow & (1+c) A_1(\kappa_l^q) = \frac{\sum_{i=1}^{m} w_l^{(i)} \cos(\omega_q^{(i)} - \mu_l^q)}{\sum_{i=1}^{m} w_l^{(i)}} + R_0 \\
\Leftrightarrow & \kappa_l^q = A_1^{-1} \left(\frac{1}{1+c} \left[\frac{\sum_{i=1}^{m} w_l^{(i)} \cos(\omega_q^{(i)} - \mu_l^q)}{\sum_{i=1}^{m} w_l^{(i)}} + R_0 \right] \right).
\end{aligned} \tag{5.19}$$

It is interesting to note how this prior reduces the likelihood of high κ_l^p's: it makes sure the maximum argument for A_1^{-1} is $\frac{1+R_0}{1+c}$. This way, with an appropriate choice of c and R_0, we guarantee a maximum value for κ_l^p, keeping the numerical methods stable.

When it is desired to use a constrained VMM, as discussed in the previous section, the equation is similar. Making $\kappa_j = \kappa_j^1 = \kappa_j^2 = \ldots = \kappa_j^d$ in Eq. (5.18), it is easy to obtain

$$\kappa_l = A_1^{-1} \left(\frac{1}{1+c} \left[\frac{1}{d} \sum_{p=1}^{d} \frac{\sum_{i=1}^{m} w_l^{(i)} \cos(\omega_p^{(i)} - \mu_l^p)}{\sum_{i=1}^{m} w_l^{(i)}} + R_0 \right] \right). \tag{5.20}$$

Once again, as in Eq. (5.16), the argument of A^{-1} in this case is the mean of the arguments of A_1^{-1} in Eq. (5.19).

[3] β is a constant with respect to the parameters. By Bayes theorem, it is obtained by integrating out the parameters, to make the posterior integrate to 1, and so it does not affect the maximization.

5.8 Mixture Algorithm Description

The basic algorithm is as follows:

Algorithm 3: Mixture of Von Mises for trajectory clustering algorithm.

Input: $\{\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(m)}\}$ and the number of clusters, k

Output: $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$, $\{\boldsymbol{\kappa}_1, \ldots, \boldsymbol{\kappa}_k\}$ and $\boldsymbol{\alpha}$

Initialize $\boldsymbol{\alpha}$, $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$, $\{\boldsymbol{\kappa}_1, \ldots, \boldsymbol{\kappa}_k\}$

repeat

`// E-step`

foreach *i, j* **do**

Set $w_j^{(i)} := \frac{\alpha_j p(\boldsymbol{\omega}^{(i)}|z^{(i)}=j;\boldsymbol{\mu}_j,\boldsymbol{\kappa}_j)}{\sum_{j=1}^{k} \alpha_j p(\boldsymbol{\omega}^{(i)}|z^{(i)}=l;\boldsymbol{\mu}_j,\boldsymbol{\kappa}_j)}$

end

`// M-step`

Update α_j with Eq. (5.12) for $j = 1 \ldots, k$

Update μ_j^p with Eq. (5.10) for $j = 1 \ldots, k$ and $p = 1, \ldots, d$

Update κ_j^p with Eq. (5.14), (5.16), (5.19), or (5.20) for $j = 1 \ldots, k$ and $p = 1, \ldots, d$

until *convergence condition is satisfied*

`// Compute cluster assignments`

foreach $i = 1, \ldots, m$ **do**

$c^{(i)} = \arg\max_{j=1,\ldots,k} w_j^{(i)}$

end

After running the algorithm, the statistical parameters of the model can be used to classify trajectories. This is done by computing the probability:

$$p(z^{(i)} = j|\boldsymbol{\omega}^{(i)}; \boldsymbol{\alpha}, \boldsymbol{M}, \boldsymbol{K}) = w_j^{(i)},$$

i.e., using Eq. (5.4). This gives the probability of a trajectory belonging to each cluster, and classification is done by assigning a trajectory to the cluster with highest probability.

For the algorithm's description to be complete, an initialization procedure and a termination condition must be specified, which will be addressed next.

5.8.1 Algorithm initialization

Similarly to k-means, EM may get stuck in local optima. This means poor initial estimates for parameters $\boldsymbol{\alpha}$, $\boldsymbol{M}$ and $\boldsymbol{K}$ may cause poor results. The initialization procedure used in this work tries to reduce this problem. Parameters are initialized as follows:

- Since no information is known about cluster assignments, a natural initialization for the α_j's, the probability of being in each cluster, is to assume them equally likely, that is, $\alpha_j = \frac{1}{k}$, for $j = 1, \ldots, k$. One could also assign to α_j the fraction of samples of the input dataset assigned to cluster j by

k-means. However, our approach gives EM more "liberty", and tries to prevent it from getting stuck in the same local optimum as k-means.

- In EM, the choice of the initial $\boldsymbol{\mu}_j$'s is as critical as the choice of centroids was in k-means. A typical choice [42], and the one used in this work, is running k-means using k-means++ initialization and using the obtained centroids as the initial estimates for EM. This way, EM may improve solutions provided by k-means.

- Finally, for the $\boldsymbol{\kappa}_j$'s, the global κ, i.e., the κ of the whole data set in all dimensions is calculated and assigned to κ_j^p[4]. Although they could be initialized using the $\boldsymbol{\kappa}$ of the clusters found by k-means, this approach gives EM more "liberty".

5.8.2 Termination condition

Since EM is shown to converge to a local maximum of the log-likelihood (or log-posterior, in case of MAP estimation), a natural condition to use is to terminate the algorithm when the increase is less than a fraction ϵ of the current value of the log-likelihood. In this work, $\epsilon = 10^{-4}$ was used in all tests.

The log-likelihood is given by

$$\ln p(\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(m)} | \boldsymbol{\alpha}, \boldsymbol{M}, \boldsymbol{K}) = \ln \prod_{i=1}^{m} \sum_{j=1}^{k} \alpha_j p(\boldsymbol{\omega}^{(i)} | z^{(i)} = j, \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j) = \sum_{i=1}^{m} \ln \sum_{j=1}^{k} \alpha_j p(\boldsymbol{\omega}^{(i)} | z^{(i)} = j, \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j), \tag{5.21}$$

where $\alpha_j p(\boldsymbol{\omega}^{(i)} | z^{(i)} = j, \boldsymbol{\mu}_j, \boldsymbol{\kappa}_j)$ are the unnormalized $w_j^{(i)}$'s, meaning no additional steps are necessary to calculate it: it can use the $w_j^{(i)}$'s before normalization.

Although a termination condition involving the prior would be more correct when using MAP estimation, in this thesis the algorithm is terminated when log-likelihood converges, regardless of the method. The reason is that, no matter what method is used, when log-likelihood has converged, the posterior has likely also converged, and the results should be very similar.

5.9 Model Selection

One advantage of using probabilistic methods is the ability to use formal approaches for model selection, i.e., to systematically choose the number of clusters. One such approach is the minimum description length principle [43].

The idea behind the MDL principle is that a good model should capture patterns within a dataset, and thus could be used to encode it within a small length. Since the description of the model itself must also be encoded, more complex models require bigger lengths. Thanks to this, MDL formalizes a trade-off between model accuracy (that favors more complex models, as they can capture patterns within data better than simpler ones) and model "size".

[4] This is similar to what is usually done with Gaussian Mixtures, where the variance of the whole dataset is computed.

There are two components of the length that must be computed: the length of the model, $L(H)$, and the length of data encoded with the model, $L(D|H)$. From information theory [43], if some data α has a pdf $p(\alpha;\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a parameter vector, its optimal coding length is given by

$$L(\alpha) = -\ln p(\alpha;\boldsymbol{\theta}).$$

In the case of a set of trajectories $\{\boldsymbol{\omega}^{(1)},\ldots,\boldsymbol{\omega}^{(m)}\}$, their joint pdf is the likelihood function, $p(\boldsymbol{\omega}^{(1)},\ldots,\boldsymbol{\omega}^{(m)}|\boldsymbol{\alpha},\boldsymbol{M},\boldsymbol{K})$, and so its length is

$$L(D|H) = -\ln p(\boldsymbol{\omega}^{(1)},\ldots,\boldsymbol{\omega}^{(m)}|\boldsymbol{\alpha},\boldsymbol{M},\boldsymbol{K}) = -\sum_{i=1}^{m}\ln\sum_{j=1}^{k}\alpha_j p(\boldsymbol{\omega}^{(i)}|z^{(i)}=j,\boldsymbol{\mu}_j,\boldsymbol{\kappa}_j), \tag{5.22}$$

that is, the log-likelihood function, where the last step used Eq. (5.21). This means that the higher the likelihood, that is, the better the model describes the data, the lower the length.

The encoding of the model is more delicate, and it depends on how the parameters, which are real numbers, are encoded with finite precision. Making some approximations it can be shown that the optimal coding length for the parameters is, approximately [43]

$$L(H) = \frac{1}{2}\sum_{i=1}^{c}\ln m_i,$$

where c is the number of parameters of the model and m_i is the number of samples from the dataset used in estimating the ith parameter of the model. As this term increases with c, it penalizes more complex models. Assuming all m data samples are used to estimate each parameter, this reduces to $\frac{c}{2}\ln m$ and the final MDL cost is

$$MDL_{cost} = L(D|H) + L(H) = -\sum_{i=1}^{m}\ln\sum_{j=1}^{k}\alpha_j p(\boldsymbol{\omega}^{(i)}|z^{(i)}=j,\boldsymbol{\mu}_j,\boldsymbol{\kappa}_j) + \frac{c}{2}\ln m. \tag{5.23}$$

In the case of VMM's, c can take two possible values. In both unconstrained and constrained VMM's, the model estimates the α_j's and $\boldsymbol{\mu}_j$'s, which account for $k + d \times k$ parameters, where d is the dimension of the parameter vector. When the $\boldsymbol{\kappa}_j$'s are unconstrained, each has d degrees of freedom and there are k $\boldsymbol{\kappa}_j$'s, so $c = k \times (2d+1)$. When they are constrained, $c = k \times (d+2)$.

Having fully specified the MDL cost, the selection of the number of clusters is straightforward: several possible values for k are tried, and the one minimizing the MDL cost is chosen. Since EM is susceptible to local optima, it is run several times for each value of k, and only the best solution (the one with highest likelihood) is considered.

Chapter 6

Matrix Factorization

Matrix factorization and low-rank techniques have recently found many applications, ranging from image segmentation and reconstruction [44] to face recognition and text analysis [45], and even recommender systems [46]. The idea behind them is that, in many cases, high-dimensional data lies in a low-dimensional space. In real world data, this is not exactly true because of noise, but in many cases it is *approximately* true.

The goal of matrix factorization is to (approximately) factorize a data matrix, V, as a product of two low-rank matrices, W and H, i.e., $V \approx W \times H$. This can be interpreted as writing the columns of the V matrix as linear combinations of a small set of columns, the columns of W, with weights that are in the columns of H. For this reason, W is usually called the centroid matrix, and H the weight matrix. Matrix factorization is illustrated in Figure 6.1, where the columns of V are (approximately) linear combinations of the 2 columns of W.

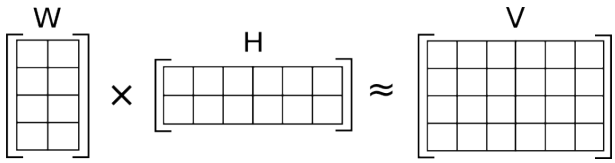


Figure 6.1: Illustration of matrix factorization. Each column of V is (approximately) a combination of the columns of W, with the weights in H. Image from [47].

For data analysis, the V matrix, a $d \times m$ matrix, is usually obtained by arranging m d-dimensional data samples as its columns. If we are looking for a rank k approximation, matrices W and H must be, respectively, $d \times k$ and $k \times m$, and so we get:

$$\boldsymbol{v}_i \approx \sum_{j=1}^{k} \boldsymbol{w}_j \cdot H_{ji}, \tag{6.1}$$

where $\boldsymbol{v}_i$ is the ith column of V, $\boldsymbol{w}_j$ the jth column of W and H_{ji} the entry (j, i) of H.

The factorization itself is usually posed as an optimization problem. A cost function penalizes the error $V - WH$ and must be minimized. In the literature, mainly two cost functions are used: the squared Frobenius norm and the Kullback–Leibler divergence [48]. In the case of the Frobenius norm, the opti-

mization problem is

$$\min_{W,H} \|V - WH\|_F^2, \tag{6.2}$$

where $\|A\|_F^2 = \sum_{ij} A_{ij}^2$ is the squared Frobenius norm of matrix A. For simplicity, this work will focus only on this cost function.

6.1 Nonnegative Matrix Factorization

Nonnegative matrix factorization (NMF) is a special case of matrix factorization where no element of matrices V, W and H can be negative. One big advantage of this kind of decomposition comes from its *interpretability*.

Constraining the combination weights, that is, the entries of H, to be nonnegative, leads to decomposing the columns of V as an *additive linear combination* of W's columns. This way, the combination process can be thought of as combining parts, the columns of W, to form the whole, leading to a *parts-based representation*. Nonnegative weights are necessary for this representation, as negative weights could be used to "delete" other components. There is a great interest in parts-based representations as the human brain is believed to work in a similar way [45].

6.1.1 Sparse NMF

Another important property of NMF, and the most interesting one, from the perspective of this thesis, is its relationship with k-means. It has been shown that, with the appropriate constraints, k-means can be written as a NMF problem [49] and NMF and its variants have been shown to work well in many clustering problems [48].

Interpreting the columns of the W matrix as the centroids of the clusters, H can be interpreted as the cluster assignments. The larger a weight is, the more important the corresponding centroid is for a training sample. This can be used for clustering, assigning each sample to the cluster that has the highest weight.

In the case of k-means, it performs what is called *hard-clustering*, that is, any sample belongs to exactly one cluster. This is equivalent to approximating each sample by only one centroid. In other words, to represent k-means as matrix factorization, each column of the H matrix must be composed only of zeros, except for one element, which must be 1. Unfortunately, forcing this constraint on NMF leads to a computationally intractable problem [48], and the constraint on the columns must be relaxed to make it solvable.

One way to do this is by using Sparse-NMF [50]. Sparse-NMF introduces one extra term to the cost function (besides the Frobenius norm) to force sparsity on the columns of H:

$$\min_{W \geq 0, H \geq 0} \left[\|V - WH\|_F^2 + \eta \|W\|_F^2 + \beta \sum_{i=1}^{m} |\boldsymbol{h}_i|^2 \right], \tag{6.3}$$

where $|\boldsymbol{h}_i|$ is the ℓ_1-norm of the ith column of H[1], while η and β are tuning parameters: $\eta > 0$ controls the size of the elements of W[2], while $\beta > 0$ controls the trade-off between sparseness and accuracy of the approximation.

One common way of solving the above problem is using the Alternating Non-Negativity Constrained Least Squares algorithm [48, 50]. Unfortunately, the optimization problem (6.3) is non-convex, thus there are no guarantees of finding the globally optimal solution. This, however, is rarely a problem, as the clustering results tend to be *consistent*, that is, the algorithm tends to produce the same assignments despite the initialization [50].

Since sparsity is not strictly forced, the columns of H may have more than one non-zero element. This means that, similarly to the Von Mises Mixture algorithm, Sparse-NMF performs soft-clustering, and the columns of H can be interpreted as posterior cluster membership probabilities, after normalization [48].

6.1.2 Negative Data

In many cases of interest, the data matrix has negative entries, rendering standard NMF unusable. To overcame those issues, semi-NMF was been proposed [51]. In semi-NMF, both matrices V and W are unconstrained, allowing negative data to be treated, while still requiring the entries of H to be nonnegative, so that the parts-based representation property is not lost.

A similar variation to Sparse-NMF has been proposed, called Sparse Semi-NMF (SSNMF) [52]. It solves the same optimization problem, but with an unconstrained W matrix.

When dealing with negative data, it is especially advantageous to use SSNMF rather than semi-NMF for clustering. Since the data matrix has negative components, the centroid matrix will also have negative components that can be used to cancel out others. While this may lead to better approximations (lower errors), it leads to poorer clustering results. SSNMF, on the other hand, will try to use as few columns of W as possible, due to the sparseness constraints. This way, SSNMF tries to choose them so that every column of V is closely approximated by few components, typically one or two, leading to better clustering results.

6.2 Application to circular data

Given the close relationship of NMF and its variants with k-means, it is natural to wonder if it can be used to group trajectories by their shapes. The challenge is getting it to work with circular data.

Due to their periodic nature, some caution is needed when writing an angular vector as a linear combination of other angular vectors. Following a reasoning similar to the one used in Chapter 4, when computing the mean centroid in k-means, if angles are converted to unit vectors, problems arising from the periodic nature of circular data disappear.

[1] The reason for using ℓ_1-norm is its *ability* to produce sparse solutions [36].

[2] While in some applications it may be of interest to control the size of W's elements, our goal is just to control sparsity, so we set $\eta = 0$.

If we stack the set of trajectories, $\{\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(m)}\}$, as columns of the $\boldsymbol{\Omega}$ matrix, and use the duality between $2D$ vectors and complex numbers, we can define matrix V as[3]

$$V = e^{j\boldsymbol{\Omega}} = \begin{bmatrix} | & | & & | \\ e^{j\boldsymbol{\omega}^{(1)}} & e^{j\boldsymbol{\omega}^{(2)}} & \cdots & e^{j\boldsymbol{\omega}^{(m)}} \\ | & | & & | \end{bmatrix},$$

where j is the imaginary unit and $e^{j\boldsymbol{\omega}^{(i)}}$ is the complex vector:

$$e^{j\boldsymbol{\omega}^{(i)}} = \left[e^{j\omega_1^{(i)}}, \ldots, e^{j\omega_d^{(i)}}\right]^T = \left[\cos\omega_1^{(i)} + j\sin\omega_1^{(i)}, \ldots, \cos\omega_d^{(i)} + j\sin\omega_d^{(i)}\right].$$

As usual, $\omega_p^{(i)}$ is the pth entry of trajectory $\boldsymbol{\omega}^{(i)}$.

Using this new matrix, we no longer have to deal with circular data and may use SSNMF for clustering. Since, by definition, the columns of V are approximated as a linear combinations of those of W, W will also be a complex matrix.

From Eq. (6.1), the pth component of the ith column of V is $V_{pi} = e^{j\omega_p^{(i)}} \approx \sum_{j=1}^{k} W_{pj}H_{ji}$. We can interpret matrix factorization geometrically by drawing $e^{j\omega_p^{(i)}}$ in the unit circle (Figure 6.2). Assuming $\sum_{j=1}^{k} W_{pj}H_{ji}$ also lies on the unit circle, as illustrated on Figure 6.2, then the difference between these 2 quantities is the red vector, whose length is the chord associated with the difference between angles $\omega_p^{(i)}$ and $\arg\left(\sum_{j=1}^{k} W_{pj}H_{ji}\right)$.

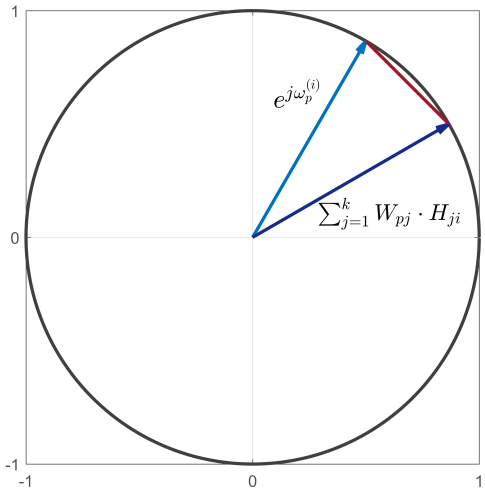


Figure 6.2: Relation between chord and complex representation of the angles. The error (difference) between the blue vectors is the chord corresponding to their angular difference.

This means that minimizing $\|V_{pi} - \sum_{j=1}^{k} W_{pj}H_{ji}\|^2$ is equivalent to minimizing the square of the chord, which, as stated in Chapter 4, is a natural measure of the difference between angular quantities. For this interpretation to be true, $\|\sum_{j=1}^{k} W_{pj} \cdot H_{ji}\|$ must be 1. While we cannot make it hold strictly, it will be approximately true, as the factorization algorithm is trying to minimize the square error, that is, it is

[3]This is a clear abuse of notation, as the exponential of a matrix is being used as the matrix of the exponentials of its entries.

trying to find a similar vector, $V_{pi} \approx \sum_{j=1}^{k} W_{pj} \cdot H_{ji}$.

A last, small modification to this formulation can be made to avoid complex-valued data. This is of interest since most NMF (and variants') toolboxes were developed for real-valued data. For each coordinate, we minimize the squared distance between vectors

$$\|V_{pi} - \sum_{j=1}^{k} W_{pj} \cdot H_{ji}\|^2.$$

Dividing the previous equation into real and imaginary parts, we obtain

$$\|V_{pi} - \sum_{j=1}^{k} W_{pj} \cdot H_{ji}\|^2 = \left(\Re\{V_{pi} - \sum_{j=1}^{k} W_{pj} \cdot H_{ji}\}\right)^2 + \left(\Im\{V_{pi} - \sum_{j=1}^{k} W_{pj} \cdot H_{ji}\}\right)^2,$$

with $\Re\{\cdot\}$ and $\Im\{\cdot\}$ representing the real and imaginary parts, respectively. This means the computation of the error in the real and imaginary parts is independent, and can be done separately. Consequently, we can divide the matrices into real and imaginary parts and stack them upon each other and obtain the same result as when using complex-valued data, that is, instead of factorizing a complex matrix, we can factorize

$$V' = \begin{bmatrix} \Re\{e^{j\mathbf{\Omega}}\} \\ \Im\{e^{j\mathbf{\Omega}}\} \end{bmatrix}. \tag{6.4}$$

The centroid matrix will change by a similar transformation and the top and bottom halves of the matrix must be concatenated, to compute the complex centroids. Since we are only interested in the argument of those complex numbers, the final centroid matrix is

$$W = \arg\left(W_1' + jW_2'\right), \tag{6.5}$$

where W_1' and W_2' are, respectively, the top and bottom halves of the W' matrix, the centroid matrix obtained by factorizing V'. In this context, $\arg(\cdot)$ returns the argument of each entry of the input matrix. Since the H matrix just contains the combination weights, it is not changed.

Given the affinity between Sparse-NMF and k-means, this NMF variant is a natural choice for trajectory clustering. Since the V' matrix can have negative elements, SSNMF must be used. For the purpose of this thesis, "The NMF MATLAB Toolbox" was used [52].

6.3 Algorithm Description

The complete clustering algorithm using SSNMF is as follows:

Algorithm 4: Sparse NMF applied to clustering of trajectories.

Input: Set of trajectories, $\{\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(m)}\}$, number of desired clusters, k and η and β parameters

Output: Clustering assignments, $c^{(i)}$'s, and, if needed, centroid matrix, W

1 Compute V' matrix from Eq. (6.4).

2 Using "The NMF MATLAB Toolbox", find W' and H minimizing:

$$\min_{W',H\geq 0}\left[\|V' - W' \times H\|_F^2 + \eta\|W'\|_F^2 + \beta\sum_{i=1}^{m}|\boldsymbol{h}_i|^2\right]$$

3 Compute W from Eq. (6.5). `// Compute cluster assignments`

4 **foreach** $i = 1, \ldots, m$ **do**

5 $\quad c^{(i)} = \arg\max_{j=1,\ldots,k} H_{ji}$

6 **end**

Besides selecting p and the number of features (d) when pre-processing the data, and k, the number of clusters, the use of this algorithm requires two additional parameters, η and β. The η parameter, attempts to constrain the elements of the centroid matrix. Since this is not of interest for our applications, we set $\eta = 0$. Parameter β, on the other hand, is an important tuning parameter, as it controls sparsity. Kim and Park [50] suggests it should be in the interval $[0.1, 0.5]$, and so, for the purposes of this work, we use $\beta = 0.1$.

6.4 Model selection

Because SSNMF is a non-parametric model, the MDL principle cannot be directly used in conjunction with it. Fortunately, other methods are available, such as the elbow method, similar to the used with k-means, and the so-called consistency method.

The elbow method works the same way as for k-means, but uses a different cost function: $J(V', W', H) = \|V' - W'H\|_F^2$. The consistency method is based on the fact that SSNMF is a consistent algorithm, that is, an algorithm that gives similar clustering results despite the initialization [50]. Using this fact, the following method was devised [50]:

1. For each k being considered, compute the consistency matrix:

$$C_k(i,j) = 1 \iff i \text{ and } j \text{ are in the same cluster}$$

2. Repeat the previous step multiple times, and compute, for each k, the mean consistency matrix.

3. Finally, compute the consistency of each clustering using

$$\rho_k = \frac{1}{n^2} \sum_{i=1}^{m} \sum_{j=1}^{m} 4 \left(\hat{C}_k(i,j) - \frac{1}{2} \right)^2 \ , \ \ 0 \leq \rho_k \leq 1,$$

and choose k where ρ_k drops.

The reasoning behind this choice for k is as follows: when k is smaller than the "true" number of clusters, the algorithm may or may not be consistent; when k is the "true" number of clusters, the algorithm should be consistent; when k is larger than the "true" number of clusters, consistency should decrease.

Chapter 7

Results

In this chapter, the clustering algorithms described in detail in the previous chapters, k-means, unconstrained and constrained VMM's, and SSNMF are tested and compared, on both synthetic and real datasets. To do this, four synthetic datasets designed to show the main characteristics of shape clustering were created. Two real datasets were made available by project SPARSIS, and correspond to surveillance tracking at the IST Alameda campus.

Tests were conducted using a regular laptop, with a 2 GHz i7 Intel processor and 8GB of RAM, running Windows 7. `MATLAB` version R2015b was used. In all the tests, unless otherwise stated, the number of features was automatically chosen to be ~ 5 times the maximum number of characteristic points found in the respective dataset, as explained in Chapter 3.

7.1 Synthetic Datasets

The performance of clustering algorithms (and other unsupervised learning algorithms) can only be correctly evaluated when the "true" number of clusters is known. Since this is not possible with real data, where none or even multiple answers may be possible, artificial data must be generated. Besides, synthetic data provides a greater degree of control over experiments. For the purposes of evaluating the clustering algorithms proposed, 4 artificial datasets were created.

7.1.1 Noiseless Trajectories

First, noiseless trajectories are considered. The performance of the proposed algorithms is analyzed using the following two datasets:

- Roundabout dataset (Figure 7.1): this example models a roundabout where vehicles enter at the bottom and circulate counterclockwise. There are 4 possible exits, each corresponding to a different type of trajectory. Each of these 4 clusters is composed of 20 noiseless trajectories. Even though they are noiseless, the generated trajectories are not exactly equal. The radius of the roundabout, its center and the start and ending points were all perturbed with zero mean Gaussian

noise, with 0.5 m standard deviation, making their shapes a little different.

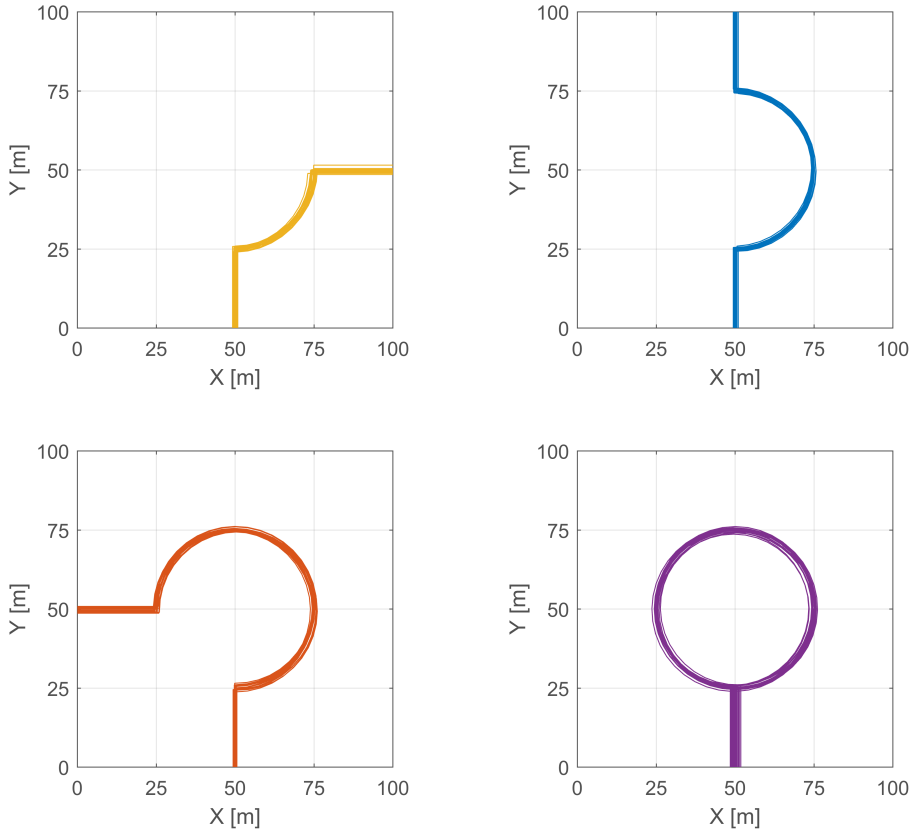


Figure 7.1: Roundabout synthetic dataset and its 4 clusters.

- Circles dataset (Figure 7.2): this example was designed to illustrate the special characteristics of trajectory shape clustering. It illustrates 100 objects performing complete circular movements, 25 per circle. The top two circles correspond to objects circulating counterclockwise, while the bottom two, to objects circulating clockwise. The radius and center of each trajectory were perturbed with zero mean Gaussian noise, with a standard deviation of 0.5 m, so the positions and radius vary between trajectories. Also, each trajectory was generated to start at a random orientation in the interval $[0, 2\pi[$. As stated in chapter 3, the preprocessing step makes any algorithm invariant to spatial translations and scaling. In this case, we are also interested in making it invariant to rotations, so that clustering ignores the starting and ending orientations of the trajectories. This is achieved by clustering the *differences between angles*, not the angles themselves. Given the blindness to translation, scaling and rotation, the dataset has two clusters: one composed by the top two circles, rotating counterclockwise, and another by the bottom two, rotating clockwise.

Since these two datasets are essentially noiseless, no smoothing was done in the preprocessing step, i.e., $p = 1$. The number of characteristic points found was 10 in both cases, and so the number of features used was 50.

Each of our four algorithms (k-means, both unconstrained and constrained VMM's, and SSNMF) was run on these two datasets and given the right number of clusters. This was repeated 1000 times for each algorithm and each dataset and the number of times the right clustering was found was counted. The

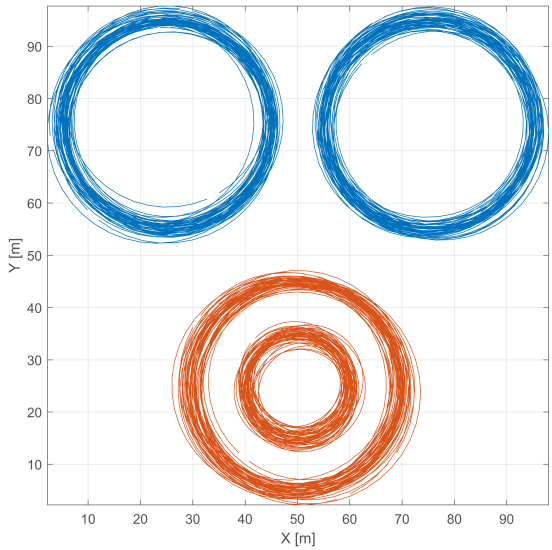


Figure 7.2: Circles dataset and its 2 clusters. The two top circles rotate counterclockwise, while the bottom two rotate clockwise.

results are shown in Table 7.1. For a better comparison, both VMM's were initialized with the centroids found by k-means and not with an independent initialization, so any improvement over it could be found[1].

Table 7.1: Percentage of times the optimal assignment was achieved out of 1000 trials in the noiseless datasets. The results signaled with an '*' use a prior.

Dataset	k-means	unconstrained VMM	constrained VMM	SSNMF
Roundabout	96.7%	96.7%	96.7%	100.0%
Circles	100.0%	100.0%*	100.0%*	100.0%

In the roundabout example, all clustering algorithms were able to find the correct clusters in most of the trials. It is interesting to note that none of the VMM's improved over k-means. In the circles example, both k-means and SSNMF found the correct clusters in all trials. It is also worth to mention SSNMF correctly found the correct clusters 100% of the time in both datasets, showing it is indeed a consistent algorithm. In the case of VMM's, both failed to converge. This happened because the trajectories were *too similar*[2], making the concentration parameters tend to infinity, and thus failing to converge.

To deal with this issue, a limit on the κ's must be introduced. This is done with the aid of a prior, i.e., using MAP estimation. The conjugate prior of the Von Mises distribution (see Eq. (5.17)) has two tuning parameters: R_0 and c. Since we intend to prevent the algorithm to choose too high κ's, we want to make high κ's less likely than low κ's, and so the R_0 parameter, on the exponent of the exponential, must be negative. For the lack of a better way of choosing them, we make $R_0 = -c$. Limiting the maximum κ to

[1] This was also performed on the tests with the noisy synthetic datasets.

[2] In fact, since the algorithms are invariant to translation, scaling and, in this case, rotation, all trajectories within each cluster are nearly identical after pre-processing.

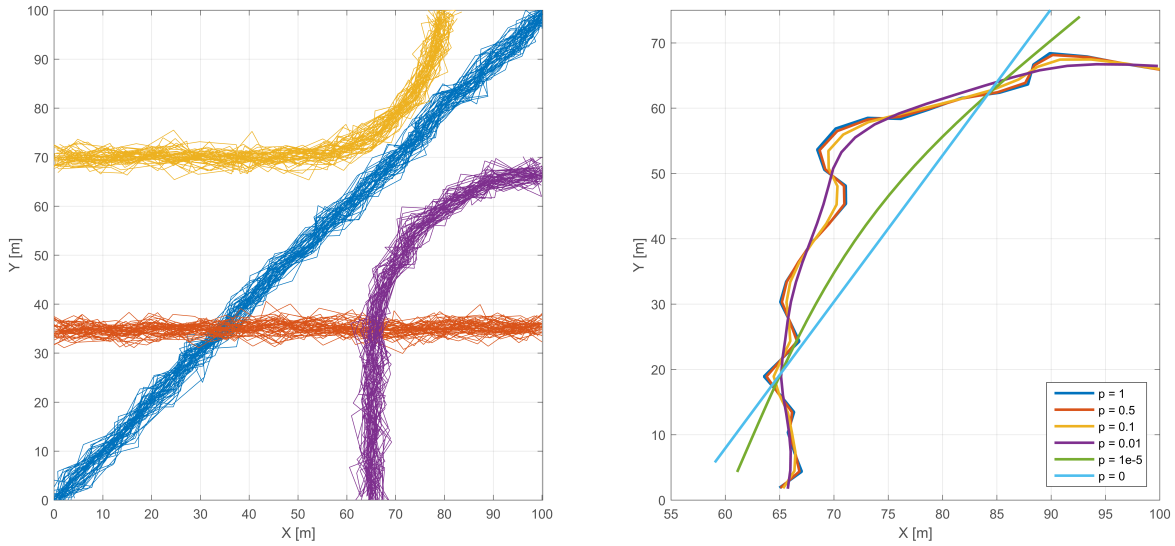
5000, we get[3]

$$\frac{1-c}{1+c} = A_1(5000) \Leftrightarrow c = -R_0 \approx 5e-5. \tag{7.1}$$

Using these choices for R_0 and c and repeating the previous test, both variants of VMM correctly cluster the dataset with the same efficiency of the other algorithms, i.e., in all trials. From this point onwards, all tests with VMM's use this prior.

7.1.2 Noisy Trajectories

For testing the effect of noise on the performance of the clustering algorithms, a new dataset was generated, the Noisy Tracks dataset. This dataset, shown in Figure 7.3(a), has 4 clusters, with 50 trajectories each, and is corrupted by zero mean Gaussian noise, with a standard deviation of 1.5 m.



(a) Noisy Tracks synthetic dataset, with 4 clusters.

(b) Effect of smoothing parameter p in a trajectory from the Noisy Tracks dataset. The best results are obtained for $p = 0.01$.

Figure 7.3: Noisy dataset and effect of the smoothing parameter.

Since the algorithm for finding the number of characteristic points needed to describe a trajectory is highly sensitive to noise, a good smoothing parameter p must be chosen. This was done by *trial-and-error*, as illustrated in Figure 7.3(b). As can be seen, $p = 0.01$ seems a good compromise, filtering most of the noise without distorting the trajectories significantly. Using this value, Algorithm 1 finds 6 characteristic points, and so we use 30 features in the following experiments.

To test the effect of noise in the algorithms, each was run with different values for p. This was repeated 1000 times and the number of times the correct clustering was found was registered. Results are summarized in Table 7.2. There are some remarks to be made:

1. k-means and both constrained and unconstrained VMM's performed similarly, and their performance increased with smoothing (decreasing p). In some cases there was a small improvement

[3] In both the synthetic noisy and the real datasets, the maximum value found for κ was less than 1000, so 5000 seems a plausible upper limit.

of the VMM's over k-means.

2. SSNMF went against the tide, with its performance decreasing drastically for $p \leq 1e-5$. For the other values, its results were fairly consistent, and it outperformed the rest of the algorithms, except for $p = 1$. This seems to suggest that the algorithm performs poorly if its input is too noisy.

3. In this particular case, the performance of VMM's and k-means was best for $p = 0$. In general, however, this will not be the case since, for $p = 0$, all trajectories are approximated by straight lines, and therefore shape information is lost.

Table 7.2: Percentage of times the optimal assignment was achieved out of 1000 trials in the noisy tracks dataset, for different p's.

Smoothing	k-means	unconstrained VMM	constrained VMM	SSNMF
$p = 1$	58.1%	58.4%	58.4%	42.6%
$p = 0.5$	58.8%	59.0%	59.0%	64.3%
$p = 0.1$	65.4%	65.4%	65.4%	83.6%
$p = 0.01$	82.4%	82.4%	82.4%	92.4%
$p = 1e-5$	99.1%	99.1%	99.1%	35.7%
$p = 0$	99.7%	99.7%	99.7%	34.6%

Overall, good results were obtained for $p = 0.01$, making it seem a reasonable value, and justifying the choice made in Figure 7.3(b).

For a final test with synthetic data, a new dataset was generated to illustrate the advantages of probabilistic modeling (VMM's) over non-parametric techniques (k-means and SSNMF). This dataset, shown in Figure 7.4, contains 100 trajectories, equally distributed between the two clusters, and was generated by the same mean trajectory, but with different noises. In both clusters, trajectories were corrupted by zero mean Gaussian noise, but with different standard deviations: 0.5 m for the red one and 2 m for the other.

As we are interested in separating the clusters by *concentration*, i.e., by noise, no smoothing was used ($p = 1$). Due to the noise, Algorithm 1 found 20 characteristic points. Since this is clearly too many, the number of features used in the following tests was $d = 30$. As always, the tests were repeated 1000 times, and the results are shown in Table 7.3.

Table 7.3: Percentage of times the optimal assignment was achieved out of 1000 trials in the concentration dataset.

Dataset	k-means	unconstrained VMM	constrained VMM	SSNMF
Concentration	0.0%	68.9%	79.4%	0.0%

As expected, only the VMM algorithms can distinguish these clusters. It is also interesting to note that the constrained version outperformed the unconstrained one. This is probably due to the unconstrained version having too many degrees of freedom for the available dataset.

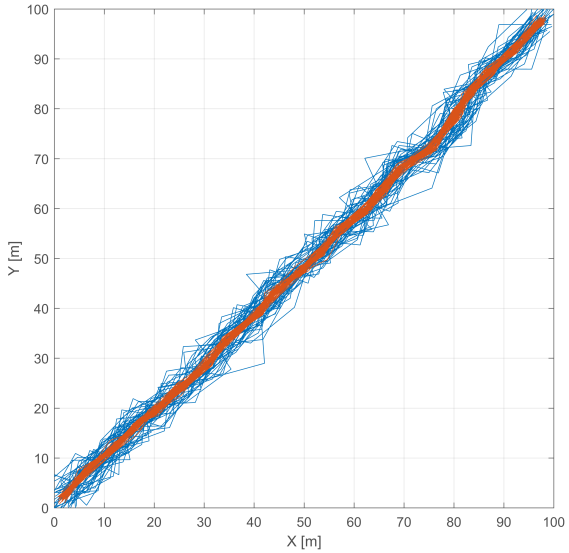


Figure 7.4: Concentration dataset and its 2 clusters. The two clusters only differ in their *concentration*.

7.1.3 Influence of β on SSNMF results

In the previous tests, the β parameter of SSNMF was chosen as 0.1. To determine its effect on the results, a parametric study was conducted. Using the noisy tracks dataset (Figure 7.3(a)) with $p = 0.01$, β was varied. For each β, the tests were repeated 100 times. Results are summarized in Table 7.4.

Table 7.4: Percentage of times the optimal assignment was achieved out of 100 trials for various values for β.

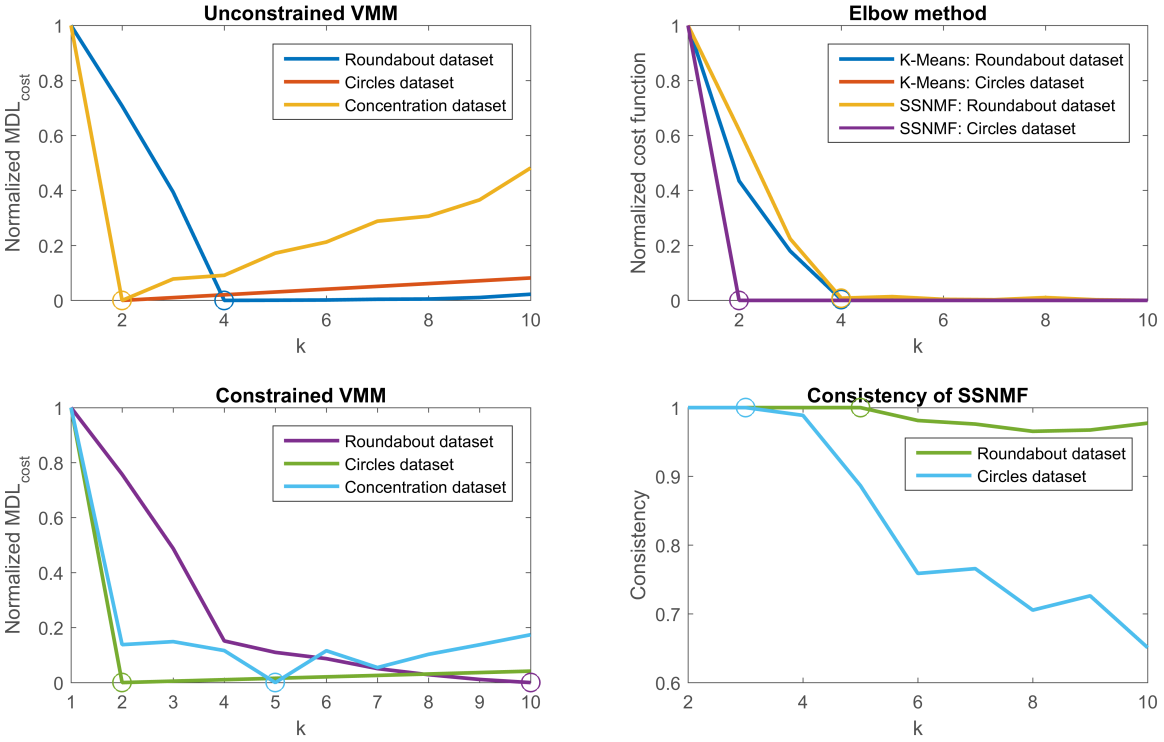
β	0	0.001	0.01	0.1	0.2	0.5	1	2	5	10
Results	1%	32%	92%	94%	92%	91%	82%	63%	61%	63%

As can be seen, for $\beta \in [0.1, 0.5]$, the interval of values suggested by [50], performance is approximately constant, thus justifying the choice of β used. For $\beta = 0$, SSNMF reduces to semi-NMF, and gives very bad results. This happens because without the sparsity constraint, all columns of the W matrix (centroid matrix) will be used. Since they can have negative components (features), they can be used to cancel out other terms. While this may lead to a better approximation, it gives poorer clusters. High β's, on the other hand, weigh the sparsity too much, and the quality of the results declines for $\beta > 0.5$.

7.1.4 Model Selection

In all the previous tests, the algorithms were given the correct number of clusters. In real-life situations, this is seldom the case, so we now turn to our model selection methods. In previous chapters, three methods were introduced: the elbow method, the MDL principle, and the consistency method, that are now tested in the synthetic datasets.

For each algorithm and dataset, the number of clusters, k, was varied in the range[4] $1, \ldots, 10$ and the tests were repeated 20 times for each k, to avoid local optima. Figures 7.5(a) and 7.5(b) show the plots of the MDL cost, cost function and consistency[5] for the roundabout, circles and concentration datasets. To test the influence of smoothing on the results, similar tests were conducted with the noisy tracks dataset while varying p, as shown in Figures 7.6 and 7.7.



(a) Normalized MDL costs for the synthetic datasets using unconstrained (top) and constrained (bottom) VMM's.

(b) Normalized cost functions for the synthetic datasets using k-means and SSNMF (top) and consistency of SSNMF (bottom).

Figure 7.5: Results with the MDL principle, on the left, and the elbow and consistency methods, on the right. The circles indicate the chosen number of clusters.

The results are summarized in tables 7.5 and 7.6. Some remarks can be made:

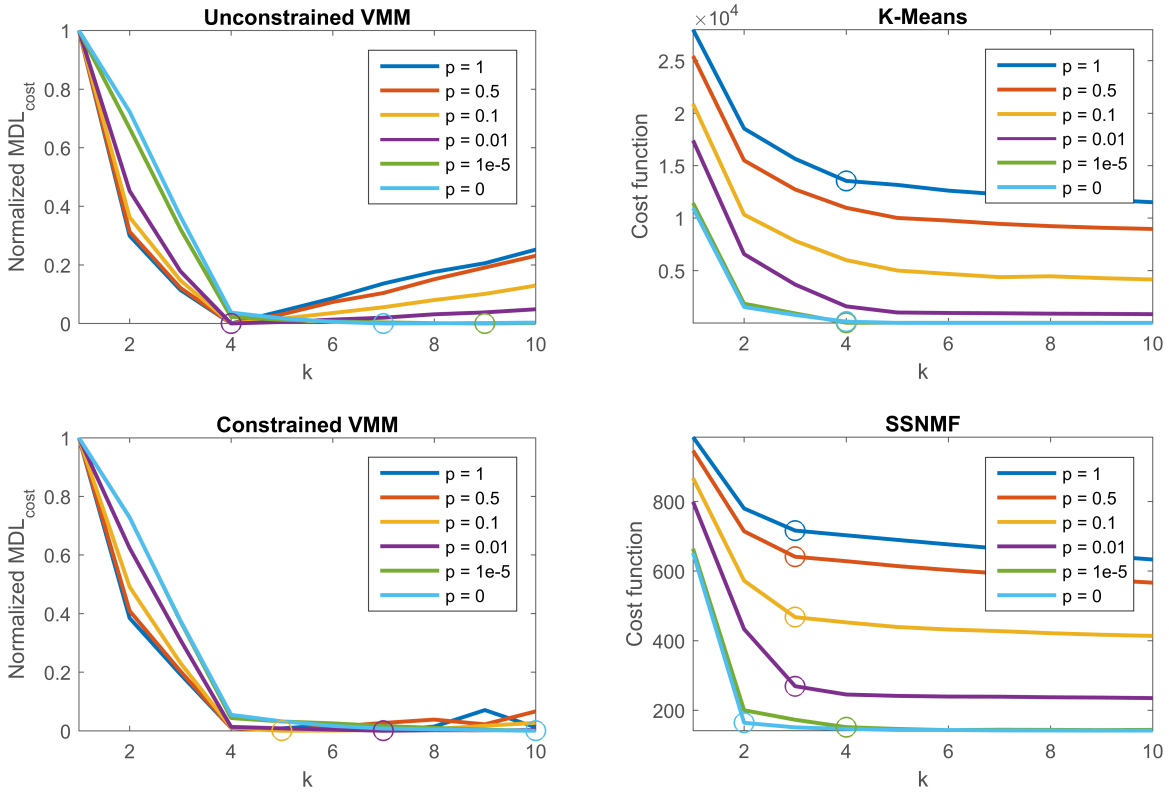
- In the absense of noise, both the elbow method (used with k-means and SSNMF) and MDL minimization (with the unconstrained VMM) work well, always finding the true number of clusters[6]. However, MDL has the advantage of being better defined, as it just requires finding k minimizing the MDL cost. Finding an elbow, on the other hand, is trickier and it may not be clear if one exists, as is the case for p between 0.01 and 0.5 in the noisy dataset.

- MDL minimization does not work well with constrained VMM's, and tends to choose too many clusters. This happens because the constrained model uses very few parameters, and so $L(H)$ does not grow rapidly enough to compensate for the decrease in $L(D|H)$.

[4] Except when computing consistency, since consistency is always maximum for 1 cluster.

[5] Since we are only interested in the minimum (for the MDL cost) or the elbow, for the elbow method, the plots were normalized to the range $[0, 1]$ in order to fit the same figure. This was also done in some of the following figures.

[6] In the cases where they are applicable. Neither k-means nor SSNMF can correctly cluster the concentration dataset, and so they cannot find the true number of clusters.



(a) Normalized MDL costs for the noisy tracks dataset (for different p's), using unconstrained (top) and constrained (bottom) VMM's.

(b) Cost functions for the noisy tracks dataset (for different p's) using k-means (top) and SSNMF (bottom).

Figure 7.6: Results with the MDL principle, on the left, and the elbow and consistency methods, on the right. The circles indicate the chosen number of clusters.

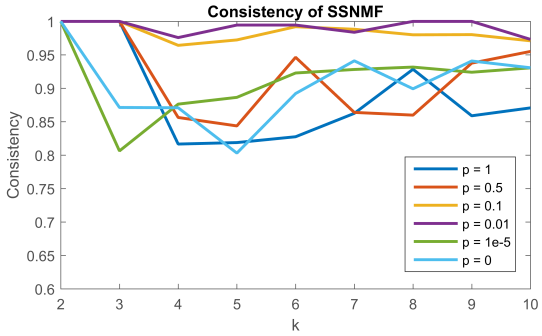


Figure 7.7: Consistency of SSNMF's results for the noisy tracks dataset, for varying p.

- The consistency measure used with SSNMF performed poorly, even in the noiseless datasets. This is not because of SSNMF being inconsistent, but rather, for it also being consistent for k's greater than the true value, rendering it useless for model selection.

- In the presence of noise, only MDL minimization (using unconstrained VMM's) found the right number of clusters, except for very small p, i.e., too much smoothing. This suggests it is better to use less smoothing, and risk a worse likelihood of finding a good clustering than to use too much and find useless ones. There can be two explanations for the effect of small p's:

1. With too much smoothing, local patterns in the dataset may emerge, which can lead to a wrong choice.
2. The more smoothing is used, the more similar trajectories in the same cluster will be, which will cause the concentration parameter to grow. Even though a prior was used, and its value is limited, large κ's cause higher likelihoods, making the $L(D|H)$ term decrease faster than the increase in $L(H)$, leading to wrong choices.

Table 7.5: Results of automatic model selection techniques and the true answer.

Method	Roundabout	Circles	Concentration
True	4	2	4
Unconstrained VMM	4	2	4
Constrained VMM	10	2	5
k-means	4	2	–
SSNMF (elbow method)	4	2	–
SSNMF (consistency)	5	3	–

Table 7.6: Influence of p on model selection in the noisy tracks dataset. The true number of clusters is 4 and the '*' indicates no clear k could be chosen.

Algorithm	$p=1$	$p=0.5$	$p=0.1$	$p=0.01$	$p=1e-5$	$p=0$
Unconstrained VMM	4	4	4	4	9	7
Constrained VMM	7	5	5	7	10	10
k-means	4	$4/5^{\star}$	$4/5^{\star}$	$4/5^{\star}$	4	4
SSNMF (elbow method)	3	3	3	$3/4^{\star}$	4	2
SSNMF (consistency)	3	3	3	3	2	2

7.2 Real Datasets

We now consider two different real surveillance scenarios: one from the IST Alameda campus, and another from a staircase, also at IST. Homographic transformations were applied on both datasets to compensate for perspective induced distortions.

7.2.1 Campus Dataset

The campus dataset is shown in Figure 7.8. It is composed of 134 trajectories, and we can recognize different types of activities:

- People using the zebra crossing (in the middle) and going in and out of the main building (on the right).
- People coming from the left and going going diagonally up and down, and people doing the opposite movement (entering on the top or bottom of the image and leaving from the left).

- People leaving the main building and going up and down, and people coming from up or down and entering the building.
- People walking up and down in front of the main building.

This way, it is expected that any clustering algorithm finds around 12 clusters, so these activities can be recognized.

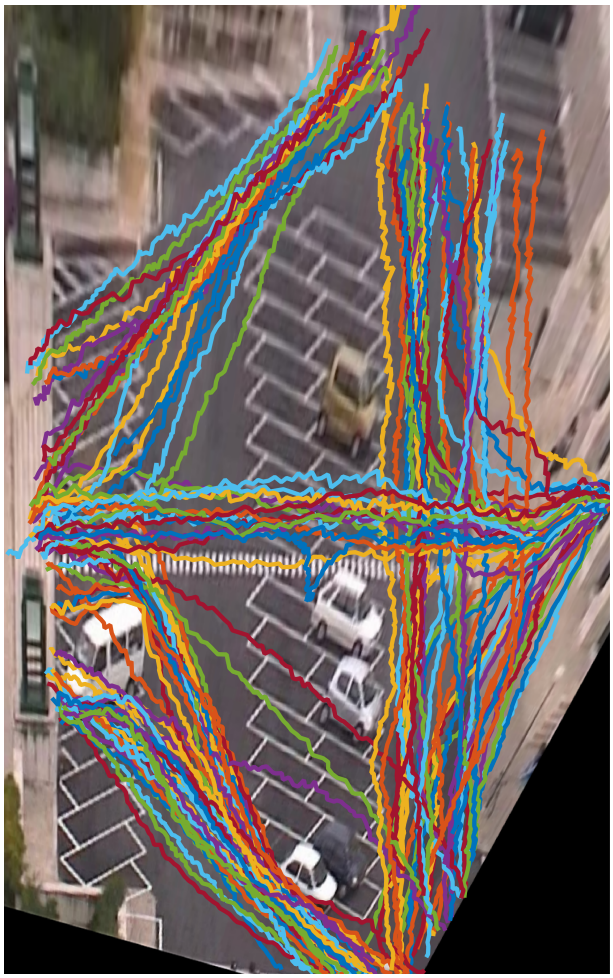


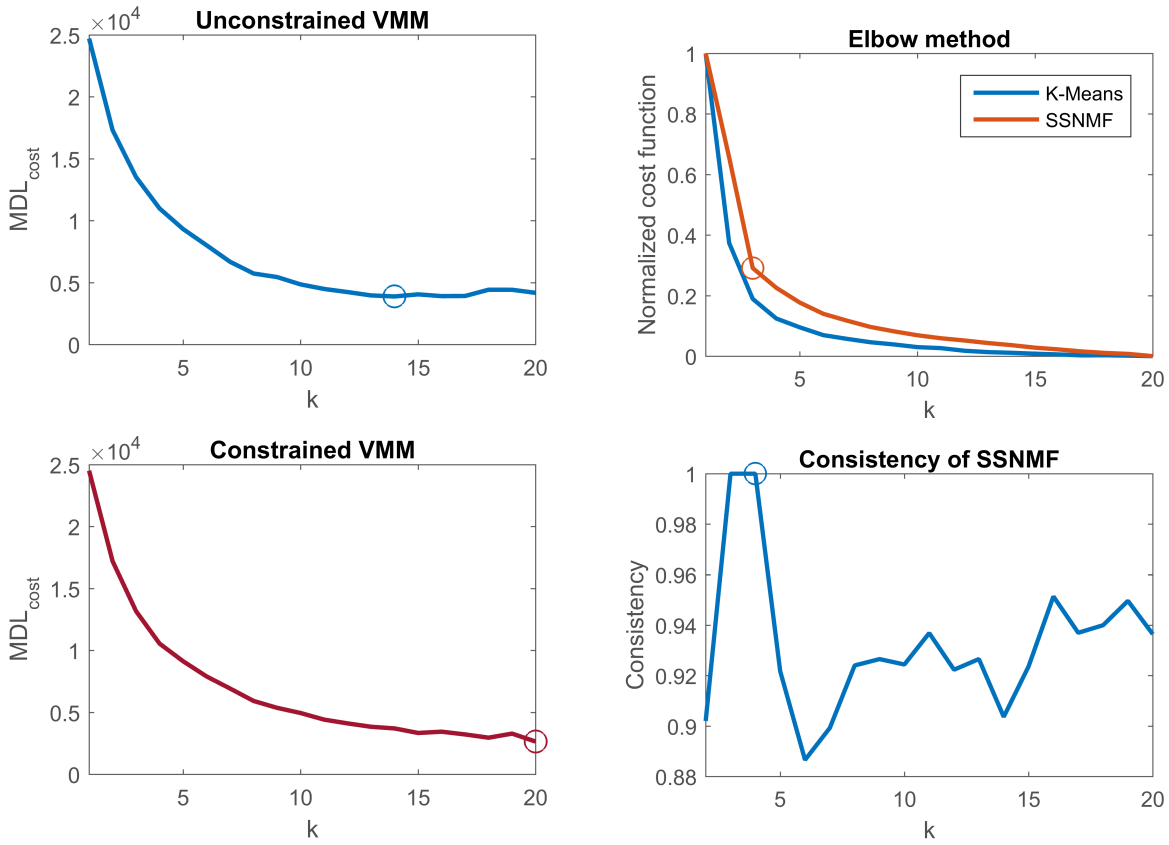
Figure 7.8: Homographic transformation of the IST Campus dataset.

Similarly to what was done with the synthetic datasets, the methods for model selection were executed 20 times for each k between[7] 1 and 20. By *trial-and-error*, $p = 0.001$ was chosen, which gave 21 characteristic points, and so 100 features were used. The results of model selection methods are shown in Figures 7.9(a) and 7.9(b).

With the exception of MDL minimization with the unconstrained VMM, none of the other model selection criteria worked:

- Similarly to what happened on the synthetic datasets, when using a constrained VMM, MDL cost keeps decreasing with the number of clusters, as the $L(H)$ term does not grow fast enough to compensate for the decrease $L(D|H)$.

[7] Once again, consistency was not computed for $k = 1$, since it is necessarily 1.



(a) MDL cost for unconstrained (top) and constrained (bottom) VMM's.

(b) Normalized cost function of k-means and SSNMF (top) and consistency of SSNMF (bottom).

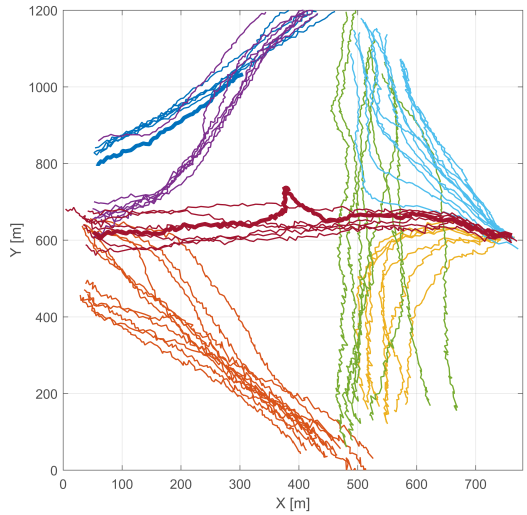
Figure 7.9: Model selection methods applied to the IST Campus dataset.

- In the case of k-means, there is no elbow in the plot of the cost function, so k cannot be estimated. With SSNMF, there is a weak elbow at $k = 3$, but this gives poor results, as there clearly are more clusters.
- Although SSNMF continued to be a consistent algorithm, with consistencies almost always above 0.9, this property cannot be used to estimate the number of clusters, since the only sharp decrease occurs at $k = 4$, which gives a poor clustering. For larger values of k, the consistency changes irregularly, rendering this approach useless.

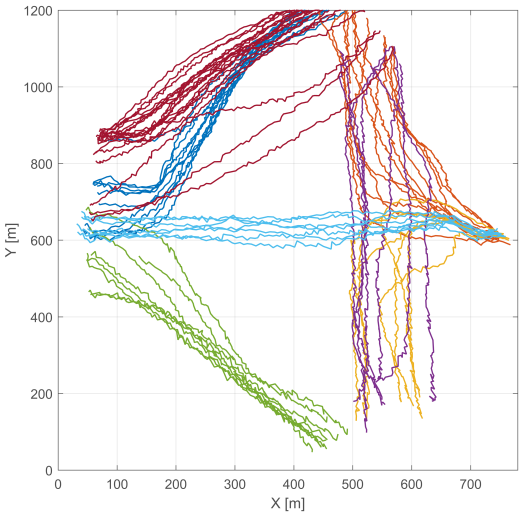
The only method that gave plausible (and good) results was MDL minimization with the unconstrained VMM. In this case, the MDL cost has a clear minimum ($3879\ nats$) at $k = 14$ and gives the clusters shown in Figures 7.10(a) and 7.10(b). These clusters are divided over two figures for easier visualization, with opposing directions plotted in different figures.

As it can be seen, the algorithm correctly found the clusters we were expecting:

- People entering and leaving the main building and going up and down.
- People using the zebra crossing in both directions.
- People walking up and down in front of the main building.



(a) 7 of the 14 clusters found in the IST Campus dataset. The 2 trajectories in bold illustrate two typical problems with real data: the different lengths (blue) and the outliers (red).



(b) The other 7 clusters.

Figure 7.10: The 14 clusters found in IST Campus dataset. They are separated over 2 plots for facility of visualization. Overlapping trajectories in opposing directions are plotted in separate figures.

- People coming from the left and going diagonally down and people doing the opposite.
- For the people coming from the left and going diagonally up (and people doing the opposite), the algorithm distinguished 2 clusters (for each direction). This is indeed plausible: in Figure 7.10(a) the dark blue cluster contains straighter trajectories than those in purple; similarly in Figure 7.10(b), the trajectories in burgundy are different from those in dark blue.

Another interesting point is that this real dataset is "plagued" with two common problems in real trajectory datasets (besides noise), shown as the bold lines in Figure 7.10(a): a shorter than average trajectory in blue, and a trajectory with outliers, in red. Despite this, the algorithm worked well and found a good clustering. Although none of the other methods found useful clusterings, when given $k = 14$, all found similar results.

7.2.2 Staircase Dataset

The staircase dataset is shown in Figure 7.11. It shows a series of people going up and down the stairs, tracked from two different views, totaling 90 different trajectories.

By trial an error, $p = 0.01$ was found, which gave 58 characteristic points needed to describe the trajectories. This is a very large number, larger than is in fact necessary, and so, instead of $5 \times 58 \approx 300$ features, we limited ourselves to 50. Similarly to what was previously done, all numbers of clusters from 1 to 12 were tried and the tests were repeated 20 times to avoid bad local optima. The results are show in Figures 7.12(a) and 7.12(b).

Contrary to what happened in the previous dataset, in this case all methods gave good results, with the exception of MDL minimization using a constrained VMM:

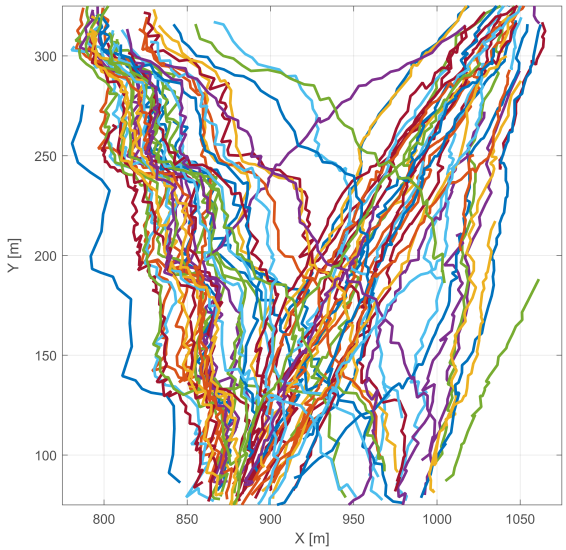
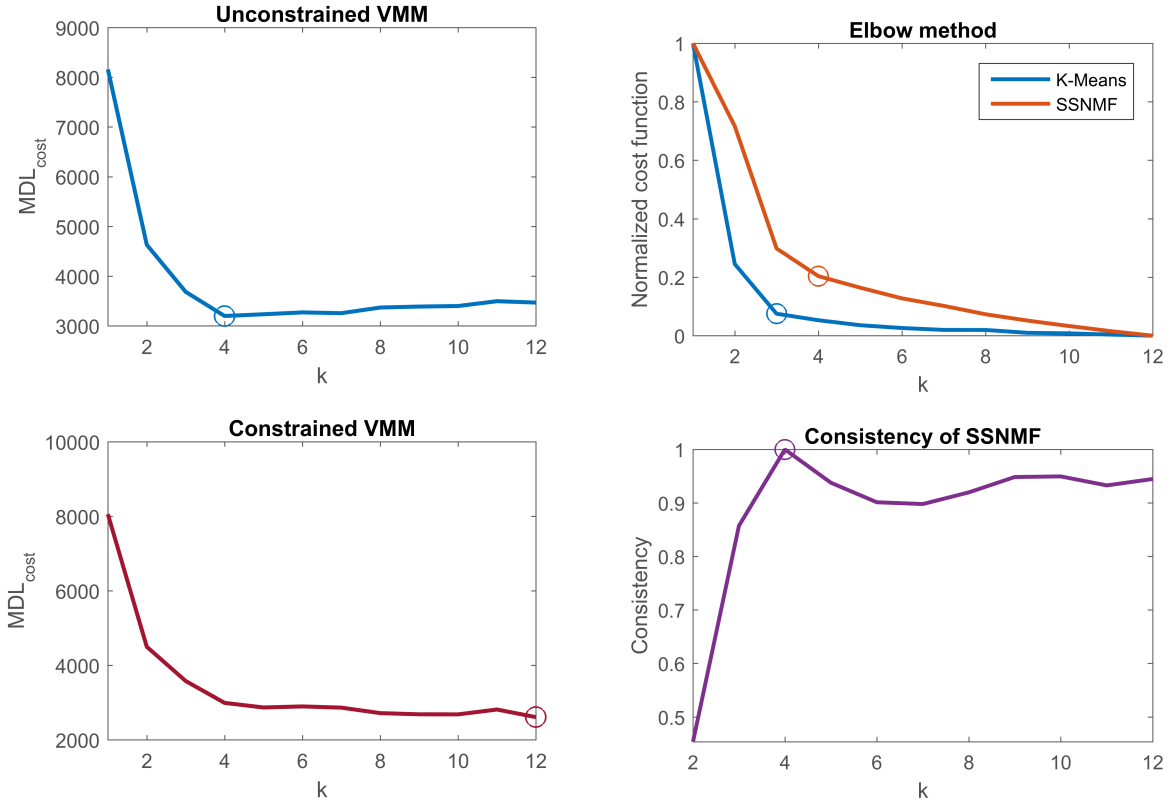


Figure 7.11: Homographic transformation of the IST staircase dataset.



(a) MDL cost for unconstrained (top) and constrained (bottom) VMM's.

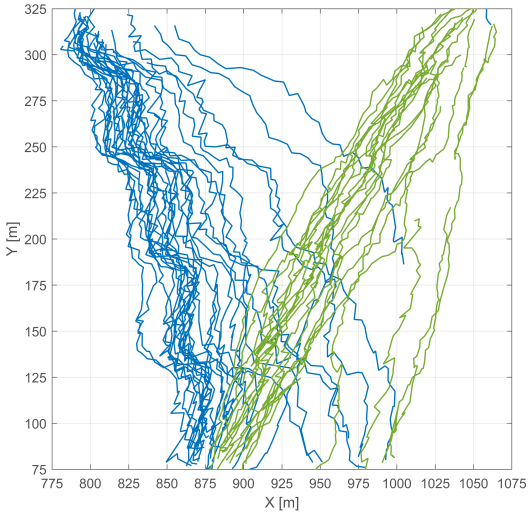
(b) Normalized cost function of k-means and SSNMF (top) and consistency of SSNMF (bottom).

Figure 7.12: Model selection methods applied to the IST staircase dataset.

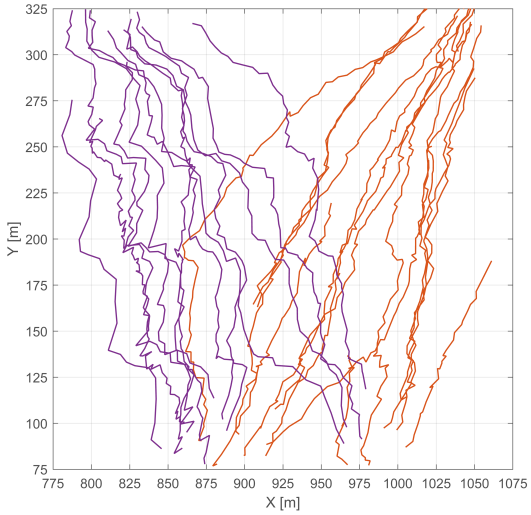
- MDL cost minimization using an unconstrained VMM finds 4 clusters, shown in Figures 7.13(a) and 7.13(b). It is a very good clustering, separating people going up and down the stairs from the

two different views in a cluster each.

- As happened in other datasets, in the case of constrained VMM, the $L(H)$ term does not grow fast enough to compensate for the decrease in $L(D|H)$, so the MDL cost is minimized at the maximum k tested, $k = 12$.
- In the case of SSNMF, both methods (elbow and consistency) gave the same result, $k = 4$, which gives the same cluster as MDL minimization (with an unconstrained VMM), shown in Figures 7.13(a) and 7.13(b).
- In the case of k-means, there is not a clear elbow, leaving some ambiguity between $k = 3$ or $k = 4$. For $k = 4$, it gives the same clustering as SSNMF, shown in Figures 7.13(a) and 7.13(b).



(a) 2 of the found clusters, corresponding to trajectories of people going down the stairs.



(b) The other 2 clusters, showing people going up the stairs.

Figure 7.13: Clusters found in the staircase dataset, using SSNMF (both the elbow and consistency methods) and MDL (with constrained VMM's). Overlapping trajectories in opposing directions are plotted in separate figures.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The problem of trajectory clustering has numerous applications, namely in the natural sciences (zoology and meteorology), in urban traffic planning and in automatic surveillance and activity recognition. This thesis proposed a novel approach to tackle it, based on the shapes of trajectories, rather than their positions.

This problem was studied with the assumption of 2-dimensional trajectories, and a pre-processing step was proposed for converting the position-based representation, in which most trajectory datasets are in, to a shape-based one. Three different methods already used in clustering were adapted for working with shape data. Finally, methods for the choice of the number of features to be used and the number of clusters were presented, so as to automate the process.

The proposed methods were tested on both synthetic and real datasets. The synthetic datasets were designed in order to illustrate noteworthy characteristics of our approach, as well as to give a greater control over experiments. The real datasets used were provided by the SPARSIS project and came from surveillance tracking at IST. In both cases, all algorithms could find good cluster assignments when given the desired number of clusters. When using model selection, the results differed much between techniques. The elbow method failed for one of the real datasets, and consistency performed poorly in almost all tests. The best method was clearly the minimization of the MDL cost, when fitting unconstrained VMM's. Overall, unconstrained VMM's proved to be the best approach, providing a consistent method for model selection, and finding good clusters most of the time.

8.2 Future Work

As a follow-up to this work, many possible directions of study could be taken. Without being exhaustive, we list a few:

- In many cases of interest, trajectories are 3-dimensional, so a a generalization of shape clustering to higher dimensions would be interesting.

- Adaptation of this methodology (shape clustering) to clustering parts of trajectories, rather than the whole trajectory. This would have the advantage of finding common sub-patterns, what can be helpful for animal tracking, for example.
- Other cluster/decomposition methods could be adapted for shape clustering and a comparative analysis could be performed.
- It would also be interesting to compare shape clustering to other approaches for trajectory classification.

Bibliography

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[2] R. A. Becker, R. Caceres, K. Hanson, J. M. Loh, S. Urbanek, A. Varshavsky, and C. Volinsky. Route Classification Using Cellular Handoff Patterns. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 123–132, New York, NY, USA, 2011. ACM.

[3] M. A. Bayir, M. Demirbas, and N. Eagle. Discovering spatiotemporal mobility profiles of cellphone users. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks & Workshops*, pages 1–9, 2009.

[4] M. Demirbas, C. Rudra, A. Rudra, and M. A. Bayir. iMAP: Indirect measurement of air pollution with cellphones. In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pages 1–6, March 2009.

[5] S. J. Camargo, A. W. Robertson, S. J. Gaffney, P. Smyth, and M. Ghil. Cluster Analysis of Typhoon Tracks. Part I: General Properties. *Journal of Climate*, 20(14):3635–3653, 2007.

[6] J. B. Elsner. Tracking Hurricanes. *Bulletin of the American Meteorological Society*, 84(3):353–356, 2003.

[7] D. Brillinger, H. K. Preisler, A. A. Ager, and J. G. Kie. An exploratory data analysis (EDA) of paths of moving animals. In *Journal of Statistical Planning and Inference*, pages 43–63, May 2004.

[8] M. Perše, M. Kristan, S. Kovačič, G. Vučkovič, and J. Perš. A Trajectory-based Analysis of Coordinated Team Activity in a Basketball Game. *Computer Vision and Image Understanding*, 113(5): 612–621, May 2009.

[9] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684, 2002.

[10] Z. Fu, W. Hu, and T. Tan. Similarity based vehicle trajectory clustering and anomaly detection. In *IEEE International Conference on Image Processing 2005*, volume 2, pages II–602–5, Sept 2005.

[11] W. Hu, D. Xie, Z. Fu, W. Zeng, and S. Maybank. Semantic-Based Surveillance Video Retrieval. *IEEE Transactions on Image Processing*, 16(4):1168–1181, April 2007.

[12] F. I. Bashir, A. A. Khokhar, and D. Schonfeld. Object Trajectory-Based Activity Classification and Recognition Using Hidden Markov Models. *IEEE Transactions on Image Processing*, 16(7):1912–1919, July 2007.

[13] M. Pierobon, M. Marcon, A. Sarti, and S. Tubaro. Clustering of human actions using invariant body shape descriptor and dynamic time warping. In *IEEE Conference on Advanced Video and Signal Based Surveillance, 2005.*, pages 22–27, Sept 2005.

[14] M. Müller. Dynamic Time Warping. In *Information Retrieval for Music and Motion*, pages 69–84, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[15] D. J. Bemdt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series, 1994.

[16] B. Morris and M. Trivedi. Learning trajectory patterns by clustering: Experimental studies and comparative evaluation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 312–319, June 2009.

[17] J. Lee, J. Han, and K. Whang. Trajectory Clustering: A Partition-and-Group Framework. In *In SIGMOD*, pages 593–604, 2007.

[18] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465 – 471, 1978.

[19] C. Jiashun. A new trajectory clustering algorithm based on TRACLUS. In *Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on*, pages 783–787, Dec 2012.

[20] S. Gaffney and P. Smyth. Trajectory Clustering with Mixtures of Regression Models. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 63–72, New York, NY, USA, 1999. ACM.

[21] D. Chudova, S. Gaffney, E. Mjolsness, and P. Smyth. Translation-invariant Mixture Models for Curve Clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 79–88, New York, NY, USA, 2003. ACM.

[22] J. Wei, H. Yu, J. H. Chen, and K. L. Ma. Parallel clustering for visualizing large scientific line data. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 47–55, Oct 2011.

[23] N. Ferreira, J. T. Klosowski, C. E. Scheidegger, and C. T. Silva. Vector Field k-Means: Clustering Trajectories by Fitting Multiple Vector Fields. *CoRR*, abs/1208.5801, 2012.

[24] J. C. Nascimento, M. A. T. Figueiredo, and J. S. Marques. Activity Recognition Using a Mixture of Vector Fields. *IEEE Transactions on Image Processing*, 22(5):1712–1725, May 2013.

[25] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31 (3):264–323, Sept 1999.

[26] S. M. Kay. *Fundamentals of statistical signal processing. [Volume I]. , Estimation theory*. Prentice Hall Signal Processing Series. Prentice Hall, Upper Saddle River (N.J.), 1993.

[27] K. V. Mardia, G. Hughes, C. C. Taylor, and H. Singh. A Multivariate Von Mises Distribution with Applications to Bioinformatics. *The Canadian Journal of Statistics / La Revue Canadienne de Statistique*, 36(1):99–109, 2008.

[28] C. De Boor. *A Practical Guide to Splines*. Applied Mathematical Sciences. Springer, Berlin, 2001.

[29] J. Robert E. Smith, J. M. Price, and L. M. Howser. A Smoothing Algorithm Using Cubic Spline Functions, February 1974.

[30] R. O. Keskin. Spatial querying for camera-based tracking platforms. 2010.

[31] A. K. Jain. Data Clustering: 50 Years Beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, Jun 2010.

[32] S. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2): 129–137, Mar 1982.

[33] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The Planar k-Means Problem is NP-Hard. In *Proceedings of the 3rd International Workshop on Algorithms and Computation*, WALCOM '09, pages 274–285, Berlin, Heidelberg, 2009. Springer-Verlag.

[34] P. Berkhin. *A Survey of Clustering Data Mining Techniques*, pages 25–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[35] A. Ng. Lecture Notes. CS229: Machine Learning. *Stanford University*, 2003.

[36] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning : Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, New York, 2009.

[37] M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009.

[38] D. Arthur and S. Vassilvitskii. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[39] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–38, 1977.

[40] K. V. Mardia and P. E. Jupp. *Directional Statistics*. John Wiley & Sons, Inc., 2008.

[41] D. Fink. A compendium of conjugate priors, 1997.

[42] J. Blömer and K. Bujna. Simple Methods for Initializing the EM Algorithm for Gaussian Mixture Models. *CoRR*, abs/1312.5946, 2013.

[43] T. C. M. Lee. An Introduction to Coding Theory and the Two-Part Minimum Description Length Principle. *International Statistical Review*, 69(2):169–183, 2001.

[44] X. Zhou, C. Yang, H. Zhao, and W. Yu. Low-Rank Modeling and Its Applications in Image Analysis. *CoRR*, abs/1401.3409, 2014.

[45] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, Oct 1999.

[46] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, Aug 2009.

[47] Wikimedia Commons. Illustration of approximate non-negative matrix factorization (nmf)., 2013. URL `https://commons.wikimedia.org/wiki/File:NMF.png`. File: NMF.png.

[48] A. Caner Türkmen. A Review of Nonnegative Matrix Factorization Methods for Clustering. *ArXiv e-prints*, July 2015.

[49] C. Bauckhage. k-Means Clustering Is Matrix Factorization. *ArXiv e-prints*, Dec 2015.

[50] J. Kim and H. Park. Sparse Nonnegative Matrix Factorization for Clustering. 2008.

[51] C. H. Q. Ding, T. Li, and M. I. Jordan. Convex and Semi-Nonnegative Matrix Factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):45–55, Jan 2010.

[52] Y. Li and A. Ngom. The non-negative matrix factorization toolbox for biological data mining. *Source Code for Biology and Medicine*, 8(1):1–15, 2013.