UNIVERSITY OF CAPE TOWN

Department of Statistical Sciences

Masters Thesis
2014

PRINCIPAL POINTS, PRINCIPAL CURVES AND PRINCIPAL SURFACES

Supervisor: Professor Sugnet Lubbe

Raeesa Ganey

October, 2014

PUBLICATION

I hereby grant the University free license to publish this dissertation in whole or part in any format the University deems fit.

PLAGIARISM DECLARATION

1. I know that plagiarism is wrong. Plagiarism is to use anothers work and pretend that it is my own.
2. I have used the APA referencing guide for citation and referencing. Each contribution to, and quotation in this dissertation from the work(s) of other people has been contributed, and has been cited and referenced.
3. I know the meaning of plagiarism and declare that all of the work in the dissertation, save for that which is properly acknowledged, is my own.

Signature: signature removed

Date: 15-04-2015

Abstract

The idea of approximating a distribution is a prominent problem in statistics. This dissertation explores the theory of principal points and principal curves as approximation methods to a distribution. Principal points of a distribution have been initially introduced by Flury (1990) who tackled the problem of optimal grouping in multivariate data. In essence, principal points are the theoretical counterparts of cluster means obtained by the k-means algorithm. Principal curves defined by Hastie (1984), are smooth one-dimensional curves that pass through the middle of a p-dimensional data set, providing a nonlinear summary of the data. In this dissertation, details on the usefulness of principal points and principal curves are reviewed. The application of principal points and principal curves are then extended beyond its original purpose to well-known computational methods like *Support Vector Machines* in machine learning.

Keywords: Principal points, k-means algorithm, principal component analysis, principal curves, principal surfaces, computational methods, machine learning

Contents

Chapter 1

Introduction

One of the classic statistical problems is the idea of approximating a distribution. Techniques related to this problem, such as descriptive or exploratory data analysis, estimation, statistical modelling and statistical learning all attempt to approximate a particular quantity of interest.

The purpose of this work is to use the main terminology and framework of statistical learning to further explain the important concept of approximation in statistics. Statistical learning falls under two main categories, namely *Supervised* and *Unsupervised* Learning.

Consider an object of interest having p measurable characteristics, either discrete or continuous. The set of all objects is called the population. Whenever an object in a population is analysed, it is assumed that uncertainty exists in terms of what the values of the p characteristics will be. Let $\mathbf{z}$ be the corresponding p variate random vector with elements representing the p characteristics of an observation in the population. In the statistical learning paradigm, it is assumed that a sample of size n, $\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_n}$, also referred to as a training sample, is drawn from the population governed by a probability mechanism represented by the probability density function $f_{\mathbf{z}}(\cdot)$.

In supervised learning, the vector $\mathbf{z}$ consists of two components $\mathbf{x}$ and $\mathbf{y}$. The $p-1$ variate vector $\mathbf{x}$ contains $p-1$ predictor variables and the random variable $\mathbf{y}$ is called the response variable. The procedure of supervised learning requires the set up of a prediction model where the future response $\mathbf{y}^*$ is predicted using the predictor variables. The training sample $(\mathbf{x_1}, \mathbf{y_1}), \ldots (\mathbf{x_n}, \mathbf{y_n})$ is used as an aid in the supervised learning process. The term supervised is used to highlight that the response variable is known and "supervises" the learning process.

In unsupervised learning, the response variable $\mathbf{y}$ is unknown in the learning process and the training sample only consists of $\mathbf{x_1}, \ldots, \mathbf{x_n}$ with probability density function $f_{\mathbf{x}}(\cdot)$. Unsupervised learning is more formally defined as the approximation of the density function $f_{\mathbf{x}}(\cdot)$. In low-dimensional cases ($p \leq 3$), non-parametric density estimation methods are used to estimate $f_{\mathbf{x}}(\cdot)$ directly. However in high-dimensional cases, $f_{\mathbf{x}}(\cdot)$ is not approximated directly, but rather descriptive statistics are found that characterise and summarise $f_{\mathbf{x}}(\cdot)$.

Unsupervised learning methods like cluster analysis and finite mixture analysis firstly try to find the main density peaks of the distribution and then partition the input space accordingly to cluster regions. Multidimensional scaling, self-organising maps, principal components and principal curves attempt to identify low-dimension manifolds of high density in the input space. The theory of principal points first introduced by Flury (1990) tackle the situation by discretising the distribution.

For this dissertation, the fundamental notion of approximation in statistics will be studied within the context of principal points and principal curves. A structured overview on the mathematical theory of principal points and principal curves will be introduced in a literature review, Chapter 2 and 3, respectively. Theorems and results pertaining to univariate and multivariate distributions for principal points will be given. The application on the estimation of principal points for two sets of data will be implemented in Chapter 4, including relevant results and figures. The concept of principal points and principal curves are then extended beyond their purpose to computational methods in machine learning practice illustrated in Chapter 5 and 6, respectively.

General Definitions

The following definitions given are extracted from Langner (2004).

- The reader is assumed to be familiar with the following terminology of linear algebra: a *vector space* and *subspace*; *dimension* of a vector space; vectors *spanning* a vector space; *normality*; *linear independence*; *orthogonality* and *orthonormality* of vectors and a *basis* for a vector space.

- The term *manifold* is used to refer to lower dimensional surfaces in $\mathbb{R}^p$. This could be linear, non-linear and include the origin or not. Whenever the manifold is linear and includes the origin the term *subspace* will be used.

- The complex number λ and the nonzero vector $\mathbf{x}$ is called an *eigenvalue-eigenvector pair* of the square matrix $\mathbf{A} : pxp$ if they satisfy the following:

$$\mathbf{Ax} = \lambda\mathbf{x}$$

- Let $\mathbf{A}$ be a square matrix. If there exists a matrix $\mathbf{A}^{-1} : pxp$ such that $\mathbf{A^{-1}A} = \mathbf{I}_p$ then $\mathbf{A}$ is called *nonsingular* or *invertible*. $\mathbf{A}$ is an *orthogonal matrix* if $\mathbf{A^{-1}} \equiv \mathbf{A^T}$, i.e. $\mathbf{A^TA} = \mathbf{I}_p$. If all the eigenvalues of $\mathbf{A}$ are positive (non negative), it is called *positive definite (positive semi-definite)*.

- Let $\Sigma : pxp$ be a positive semi-definite and symmetric matrix with $\lambda_1 \geq, \ldots, \lambda_p \geq 0$ its non negative eigenvalues. Distinct eigenvalues correspond with orthogonal eigenvectors which can be normalised. For equal eigenvalues an orthonormal basis for the space spanned by the corresponding eigenvectors can always be found.

- If $\Sigma : pxp$ is a positive semi-definite and symmetric matrix with eigenvalues $\lambda_1 \geq , \ldots, \lambda_p \geq 0$, the *spectral decomposition* indicated by $\text{SVD}(\Sigma) = (\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_p}, \lambda_1, \lambda_2, \ldots, \lambda_p)$ is written as

$$\mathbf{\Sigma} = \mathbf{ADA^T}$$

  with $\mathbf{A} : pxp = [\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_p}]$ is an orthonormal set of eigenvectors for Σ, such that a_i corresponds with the eigenvalue λ_i, for $i = 1, \ldots, p$ and $\mathbf{D} : pxp = \text{diag}(\lambda_1, \ldots, \lambda_p)$.

- Let $\{\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_q}\}$ be an orthonormal basis for ς, a q-dimensional subspace ς of $\mathbb{R}^p$, $q \leq p$. Projecting the vector $\mathbf{x} : px1$ onto ς is a linear operation with the projection matrix $\mathbf{P} : pxp = \mathbf{AA^T}$, with $\mathbf{A} : pxq = [\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_q}]$.

Chapter 2

Principal Points

2.1 Introduction

Large data sets are often difficult to assimilate, and methods of summarising and extracting relevant information are necessary. A common approach is to divide a partition of n observations into a number a groups, in such a way that observations in the same group are as alike as possible, but as distant as possible from observations in other groups.

The k points that optimally group a data set into partitions are called k principal points. Work on principal points started with a project of the Swiss army to replace existing with newly designed protection masks. The heads of 900 Swiss soldiers were measured on 25 different variables. Since human heads differ considerably in size and shape, it was clear that one single type of mask would not be sufficient to fit all users. Anthropologists were asked to provide "optimal" solutions for the number k of types ranging from 2 to 5. The study started without a clear concept of the "optimal types", and it was only after all data had been collected that professional advice from a statistician was sought.

Although principal points do not necessarily reduce the dimension of the distribution, the terminology was inspired from the idea of principal components (Pearson, 1901). Principal points are not the first attempt to represent a distribution. It's origin lies in the theory of optimal stratification (Dalenius and Gurney, 1951), optimal grouping (Cox, 1957) and maximising the correlation of grouped observations (Bofinger, 1957).

Cluster analysis is primarily concerned with optimally partitioning finite data sets without making any distributional assumptions, whereas principal points can be regarded as parameters of a theoretical distribution, that are estimated accordingly. There is a strong relationship between cluster analysis and principal points. The main ideas are partitioning, grouping or segmentation. The k-means algorithm associated with cluster analysis in (Hartigan, 1975) can be used to estimate principal points. Cluster analysis can be approached by assuming that the data set comes from a mixture of distributions with the aim to find homogenous subgroups in the data. However the main concern of principal points is finding the optimal "cluster means" of a distribution, whether it has group structure or not.

There exist separate interests of principal points for univariate and multivariate distributions. The theoretical results regarding the principal points of univariate distributions are much less complex than those of multivariate distributions. A substantial amount of research has been done since Flury (1990) had defined principal points. The following will be the focus of this chapter:

- Theoretical overview of principal points,

- The computation of principal points of a distribution (univariate and multivariate),
- The estimation of principal points from a data set (univariate and multivariate),
- The comparison between different estimators of principal points,
- A criterion for determining the optimal value of k.

2.2 Theoretical overview

Consider a p-variate random vector X with distribution function $F_{\mathrm{X}}(\cdot)$. The process of discretising the distribution according to Flury (1990) is as follows:

Let $\mathbf{y}_1, .., \mathbf{y}_k$ denote k points in a p-dimensional Euclidean space. Then the minimal distance of $\mathbf{x}$ is defined as:

$$d(\mathbf{x}|\mathbf{y}_1, ..., \mathbf{y}_k) = \min_{1 \leq j \leq k} [\{(\mathbf{x} - \mathbf{y}_j)'(\mathbf{x} - \mathbf{y}_j)\}^{\frac{1}{2}}]$$

The vectors $\xi_1, .., \xi_k$ are called k principal points of a random vector X if they minimise

$$P_{\mathrm{X}}(k) := E\{d^2(\mathrm{X}|\xi_1, ..., \xi_k)\} \tag{2.1}$$

for all choices of $\xi_1, .., \xi_k$.

Principal points may also be defined in terms of self-consistency, following the terminology of (Hastie and Stuetzle, 1989). Suppose that k points $\mathbf{y}_1, .., \mathbf{y}_k$ are given and the p-dimensional space is partitioned into k regions ("domains of attraction") $A_1, ..., A_k$, where A_j contains all points that are nearer to y_j than to any other of the given points.

$$A_j = \{\mathbf{x} : (\mathbf{x} - \mathbf{y}_j)'(\mathbf{x} - \mathbf{y}_j) < (\mathbf{x} - \mathbf{y}_h)'(\mathbf{x} - \mathbf{y}_h) \quad \forall \ h \neq j\}, \ \ j = 1, \ldots, k$$

The set of k points $\mathbf{y}_1, ..., \mathbf{y}_k$ is called self-consistent points (with respect to the p-variate random vector X) if

$$E[\mathrm{X}|\mathrm{X} \epsilon A_j] = \mathbf{y}_j$$

A sample version of self-consistent points results from the above equation, by putting probability mass $\frac{1}{n}$ on each of the observations $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^p$. With n_j equal to the number of sample points $\mathbf{x}_i$ in A_j, the set $\mathbf{y}_1, ..., \mathbf{y}_k$ are self-consistent if

$$\frac{1}{n_j} \sum_{\mathbf{x}_i \epsilon A_j} \mathbf{x}_i = \mathbf{y}_j \quad \forall_j = 1, ..., k \tag{2.2}$$

The connection between principal points and self-consistent points can now be made shown in Lemma 2.2.1 .

Lemma 2.2.1. *Principal Points are self-consistent*
Let $\xi_1, ..., \xi_k$ denote the k principal points of the random vector X with density f. Let A_j denote the domain of attraction of ξ_j, $j = 1, \ldots, k$ and $\pi_j = P(X \epsilon A_j)$, then

$$\begin{aligned} P_x(k) &= E(d^2(X|\xi_1, ..., \xi_k)] \\ &= \sum_{j=1}^{k} \pi_j E\{(X - \xi_j)'(X - \xi_j)|X \epsilon A_j\} \\ &= \min_{y_1,...,y_k} [\sum_{j=1}^{k} \pi_j E\{(X - y_j)'(X - y_j)|X \epsilon A_j\}] \\ &= \sum_{j=1}^{k} \pi_j \min_{y_j} [E\{(X - y_j)'(X - y_j)|X \epsilon A_j\}] \end{aligned}$$

But the j-th term is minimised by choosing $y_j = E(X|X \epsilon A_j)$, i.e. principal points are conditional means, and hence self-consistent. It follows from Lemma 2.2.1 that self-consistent points exist whenever principal points exists. An alternate definition of principal points can therefore be given as follows.

The principal points of a random vector X are defined as the set of self-consistent points $y_1, ..., y_k$ with domains of attraction A_j, $j = 1, ..., k$ which minimises

$$\sum_{j=1}^{k} \pi_j \text{trace}\{\text{cov}(X|X \epsilon A_j)\}$$

where $\pi_j = P(X \epsilon A_j)$ and $cov(X|X \epsilon A_j)$ is the covariance matrix of the conditional distribution of X, given $X \epsilon A_j$.

It is important to distinguish carefully between self-consistent points and principal points. For discrete distributions as in equation 2.2, sets of self-consistent points may exist in abundance. The k-means algorithm (Hartigan, 1975) converges, by construction, to a set of self-consistent points, but not necessarily to a set of principal points.

2.3 Computation of Principal Points

Principal points can be considered as parameters of a distribution which can be estimated from a random sample. For a known distribution, usually finding its corresponding parameters amounts to a simple calculation such as computing the mean or variance of the distribution. The term, computing principal points, is used when the distribution of the data set is known. This section illustrates the computation for both univariate and multivariate distributions.

2.3.1 Univariate

Consider an absolutely continuous random variable X with density function $f_x(\cdot)$ and distribution function $F_x(\cdot)$. Assume first and second moments exist. The following general notation will be useful in the computation of principal points.

Let $W_k = \{y_1, ..., y_k\}$ be a set of k points in $\mathbb{R}$, with $y_1 < y_2 < \ldots < y_k$.

- $\mu = E(X)$;
- The domains of attraction of each of the points y_j are the intervals $A_j = [c_{j-1}, c_j]$, $j = 1, ..., k$ with $c_0 \equiv -\infty$, $c_k \equiv +\infty$ and $c_j \equiv (y_j + y_{j+1})/2$ for $j = 1, ..., k$;
- These regions define the following increments $h_1 \equiv \frac{y_1}{2}, \ldots, h_j \equiv \frac{y_j - y_{j-1}}{2}$ so that $y_j = \sum_{i=1}^{j} h_i$ for $j = 1, \ldots, k$;
- By construction, the distribution function is $F_x(x) \equiv \int_{c_0}^{x} f_x(y)\, dy$ and $\bar{F}_x(x) \equiv 1 - F_x(x) = \int_{x}^{c_0} f_x(y)\, dy$
- The following functions can also be defined $Q_x(x) \equiv \int_{c_0}^{x} y f_x(y)\, dy$ and $\bar{Q}_x(x) \equiv \mu - Q_x(x) = \int_{x}^{c_0} y f_x(y)\, dy$;
- $MSE(X, W_k) = \int_{-\infty}^{+\infty} \min_j (x - y_j)^2 f_x(x)\, dx$.

Lemma 2.3.1. *Let $f_x(x) > 0$ almost everywhere on (c_0, c_k). Then given that $y_1 < \mu$ there exists a unique c_1 with $c_0 < c_1 < c_k$ such that $y_1 = \frac{Q_x(c_1)}{F_x(c_1)}$. The value y_1 is the conditional mean given $X \epsilon [c_0, c_1]$. (Zoppé, 1997)*

An analytical procedure using Lemma 2.3.1, for a given value of k, is used to find all possible sets of k self-consistent points of a random variable. It is given as follows:

- Suppose $y_1 < \mu = E[X|X > c_0]$; there exists a unique c_1 such that $c_0 < c_1 < c_k$ and
$$y_1 = \frac{Q_x(c_1) - Q_x(c_0)}{F_x(c_1) - F_x(c_0)}$$
- Because y_1 determines c_1 uniquely, c_1 can be written as a function of y_1, say $c_1 \equiv m_1(y_1)$.
- For c_1 to be the cut point between y_1 and the next point y_2, the following must hold: $y_2 = 2c_1 - y_1 \equiv 2m_1(y_1) - y_1$.Therefore y_2 can be written as a function of $y_1, y_2 = h_1(y_1)$.
- Given y_2 and c_1 and provided $y_2 < E[X|X > c_1]$, c_2 is uniquely determined as the unique solution of
$$y_2 = \frac{Q_x(c_2) - Q_x(c_1)}{F_x(c_2) - F_x(c_1)}$$
so that $c_2 = m_2(y_2) = m_2(h_1(y_1))$ and $y_3 = 2c_2 - y_2 = 2m_2(y_2) - y_2$ thus making it a function of $y_2 : y_3 = h_2(y_2)$
- This procedure is continued until $y_k = h_{k-1}(y_{k-1}) = \ldots = h^*{}_k(y_1)$ $c_{k-1} = m_{k-1}(y_{k-1}) = m_{k-1}(h_{k-2}(y_{k-2})) = \ldots = m_{k-1}(h^*{}_k(y_1))$ and where $h^*{}_k(\cdot) = h_1(\cdot) \circ h_2(\cdot) \circ \ldots \circ h_{k-1}(\cdot)$ represents a compound function.

The set $W_k = \{y_1, \ldots, y_k\}$ uniquely determined in this way by y_1, is self-consistent if and only if $y_k = E[X|X > c_{k-1}]$. For every step k of the procedure there is an upper limit b_k for y_1 such that if $y_1 < b_k$ then it is possible to find $y_k = {h^*}_k(y_1) < c_k$.

For the continuous function ${h^*}_k(y_1)$: $[c_0, b_k] \to [c_0, c_k]$, the following holds

$$\lim_{y_1 \to c_0} {h^*}_k(y_1) = c_0 \qquad \lim_{y_1 \to b_k} {h^*}_k(y_1) = c_k$$

The same procedure can be reversed, it could start at some last point y_k, with $\mu < y_k \leq c_k$, which determines c_{k-1} such that

$$y_k = \frac{\bar{Q}(c_{k-1})}{\bar{F}(c_{k-1})}$$

The cut-point c_{k-1} is uniquely determined by y_k, $c_{k-1} = n_k(y_k)$. Proceeding analogously to the "forward" way, given y_k and c_{k-1}, y_{k-1} can be determined as a function g_k of y_k.

$$y_{k-1} = 2c_{k-1} - y_k = 2n_k(y_k) := g_k(y_k)$$

Then c_1 can be obtained as a function of y_k and y_1 as a function g_k^* of y_k : $y_1 = g_k^*(y_k)$.

The set $\{y_1, \ldots y_k\}$ thus uniquely determined from y_k is self-consistent if and only if $y_1 = E[X|X < c_1]$ and y_k is greater than a lower bound l_k. The following also holds for $g_k^*(y_k)$: $[l_k, c_k] \to [c_0, c_k]$

$$\lim_{y_k \to l_k} {g^*}_k(y_k) = c_0 \qquad \lim_{y_k \to c_k} {g^*}_k(y_k) = c_k$$

Theorem 2.3.1 summarises the above discussion, which suggests a way of finding all possible sets of self-consistent points.

Theorem 2.3.1. *Let X be a univariate random variable with density $f(x) > 0$ almost everywhere on (c_0, c_k). For a fixed $k \geq 2$, the sets of k self-consistent points are in one-to-one correspondence to the interactions between the functions $y_k = h_k^*(y_1)$ and $y_1 = g_k^*(y_k)$ in the (y_1, y_k) space, i.e. they are determined by solutions y_1 or respectively y_k of one of the equations:*

$$y_1 = g_k^*(h_k^*(y_1)) \qquad y_k = h_k^*(g_k^*(y_k))$$

(Zoppé, 1997)

Theorem 2.3.1 suggests a general procedure for finding all sets of k self-consistent points of any absolutely continuous univariate distribution, using the following steps:

1. Find the function $h_k^*(\cdot)$, mapping y_1 to y_k, and the function $g_k^*(\cdot)$, mapping y_k to y_1 and plot them together in the (y_1, y_k)-plane, shown in Figure 2.1. This is known as a Zoppé curve.

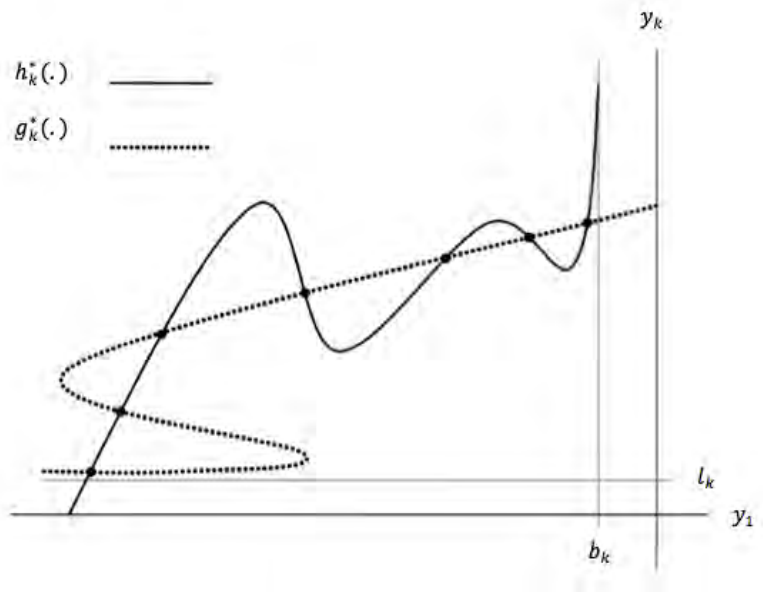


Figure 2.1: Graphical display of Zoppé Curves for an arbitrary distribution

2. Find all possible cut points of $h_k^*(\cdot)$ and $g_k^*(\cdot)$. The intersections corresponds to all possible sets of self-consistent points.

3. For each intersection point find the corresponding set of k self-consistent points $y_1, \ldots, y_k$ using the self-consistency algorithm (Tarpey, 1999) for self-consistent points.

   Self-consistency algorithm for self-consistent points. *Let* $W_{k,0} = \{y_1^{(0)}, \ldots y_k^{(0)}\}$ *be an initial set of* k *points in* $\mathbb{R}$. *Set* $m = 0$

   (a) *For each point* $y_j^{(m)}$ *in the set* $W_{k,m}$, *there exists a corresponding domain of attraction* $A_j^{(m)}$, *for* $j = 1, \ldots, k$. *Compute the conditional mean* $\tilde{y}_j^{(m)}$ *of each* $A_j^{(m)}$ *as follows:*

   $$\tilde{y}_j^{(m)} = \frac{\int_{A_j^{(m)}} x dF(x)}{\int_{A_j^{(m)}} dF(x)} \quad j = 1, \ldots, k$$

   (b) *Let* $W_{k,m+1} = \{\tilde{y}_1^{(m)}, \ldots, \tilde{y}_k^{(m)}\}$ *if and only if* $X \epsilon A_j^{(m)}$

   (c) *Update* $m \leftarrow m + 1$ *and repeat* **1** *and* **2** *until convergence.*

4. For each of the sets of self-consistent points, compute the MSE. The set corresponding to the global minimum MSE is the set of principal points.

In theory, the Zoppé-curve procedure gives a complete and absolute method for finding all possible sets of principal points of a continuous variable. In practice, this results in a graphical display of two curves and finding the intersection points. The graph will only reveal the smallest and largest values of each set of self-consistent points and not the other $(k-2)$ points. The main idea of the graphical display of the curves is to explore the behaviour of the sets of self-consistent points. The self-consistency algorithm for self-consistent points can be run with different initial points y_1 and y_k to interactively find all the sets.

Figure 2.2 is an example of a Zoppé curve for the standard normal distribution illustrated in Zoppé (1997).

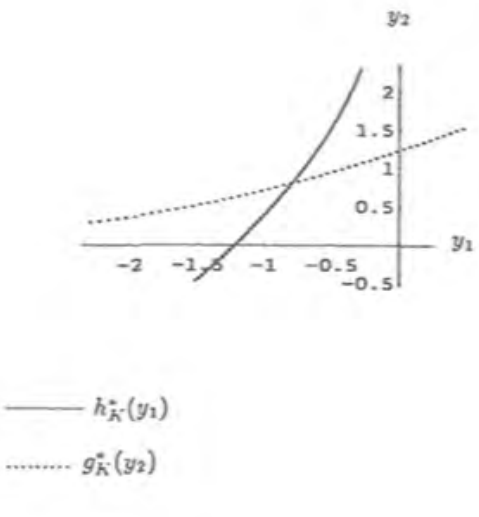


Figure 2.2: Graphs of the functions $h_K^*(y_1)$ and $g_K^*(y_K)$ for K=2 consistent points of the standard normal distribution.

2.3.2 Multivariate

Now let X be a p-variate random vector with mean vector $\mu_{\mathbf{x}} = (\mu_1, \dots \mu_{\mathbf{p}})^{\mathrm{T}}$ and covariance matrix $\sum_{\mathrm{X}} : pxp$ with $var(X_i) = \sigma_i^2, i = 1, \dots, p$.

Consider the set $W_k = \{\mathbf{y_1}, \dots \mathbf{y_k}\}$ of self consistent points, associated with domains of attraction A_j and $\pi_j = P(\mathrm{X} \epsilon A_j)$. The following results hold:

1. The mean vector μ_x is contained in the convex hull of W_k; i.e. $\mu_x = \sum \pi_j y_j$
2. $MSE(W_k, \mathrm{X}) = \sum_{i=1}^{p} (\mu_i + \sigma_i^2) - \sum_{j=1}^{k} \parallel \mathbf{y_j} \parallel^2$
3. Let $\mathrm{X_1}$ denote a p-variate random vector, and let $\mathrm{X_2} = \delta + \rho H \mathrm{X_1}$ for some $\delta \epsilon \mathbb{R}, \rho \epsilon \mathbb{R}$ and some orthogonal matrix H of dimension pxp.
   (a) If $\{\mathbf{y_1}, \dots \mathbf{y_k}\}$ is a set of self-consistent points of $\mathrm{X_1}$, then $\delta + \rho H \mathbf{y_j}$, $j = 1, \dots, k$ form a set of k self consistent points of $\mathrm{X_2}$
   (b) If $\{\xi_1, \dots \xi_k\}$ is a set of principal points of $\mathrm{X_1}$ then $\delta + \rho H \xi_j$, $j = 1, \dots, k$ form a set of principal points of $\mathrm{X_2}$, and $P_{\mathrm{X_2}}(k) = \rho^2 P_{\mathrm{X_1}}(k)$
   Proof: (Tarpey, 1992)
4. In probability and statistics, an elliptical distribution is any member of a broad family of probability distributions that generalise the multivariate normal distribution. Intuitively, in the simplified two and three dimensional case, the joint distribution forms an ellipse and an ellipsoid, respectively, in iso-density plots. (Cambanis et al., 1984) If X is elliptically distributed with mean $\mu_{\mathbf{x}} = 0$ and the set W_k of self-consistent points span a subspace of dimension $q < p$, then this subspace is spanned by a set of q eigenvectors of $\sum_{\mathrm{X}}$. This suggests that for elliptical distributions only principal component subspaces have to be searched to look for self-consistent points. This allows to assume, without loss of generality that $E[\mathrm{X}] = 0$ and that the covariance matrix of X is diagonal. Therefore it

will always be assumed for elliptical distributions that $E[\mathrm{X}] = 0$, which means that the linear manifold spanned by k self-consistent points is a subspace of dimension at most $k - 1$.

The following theorem gives a useful extension of point **4** above: whenever the set of self-consistent points is a set of principal points.

Principal Subspace Theorem. *The principal subspace theorem deals with the problem of finding subspaces supporting optimal approximations of multivariate distributions. Suppose* X *is a* p*-variate elliptical distribution with mean,* $E[\mathrm{X}] = 0$ *and covariance,* $\mathrm{cov}(\mathrm{X}) = \sum_{\mathrm{X}}$*. If* $\subset$ *is the subspace spanned by a self-consistent set of points* $\{\mathbf{y_1}, \ldots \mathbf{y_k}\}$ *of* X*, the* $\subset$ *is spanned by a set of eigenvectors of* $\sum_{\mathrm{X}}$*.*

Let the covariance matrix $\sum_{\mathrm{X}} = B\Lambda B'$*, where* $B = [B_1, \ldots B_{\mathrm{p}}]$ *is orthogonal and* Λ *is diagonal. Suppose the* k *vectors* $\{\mathbf{y_1}, \ldots \mathbf{y_k}\}$ *span the same subspace as the* q *eigenvectors* $[\beta_1, \ldots, \beta_q]$ *and* $\mathrm{X}^* = \mathbf{B_1'}\mathrm{X}$*. If the* $z_j = \beta_1' y_j$*,* $j = 1, \ldots, k$ *are self-consistent points of* X^**, then* $\{\mathbf{y_1}, \ldots \mathbf{y_k}\}$ *are self-consistent points of* X*.*

If a set of k *principal points of* X *spans a subspace* $\subset$ *of dimension* q *the* $\sum_{\mathrm{X}}$ *has a set of eigenvectors* $[\beta_1, \ldots, \beta_q]$*, with associated ordered eigenvalues* $\lambda_1 \geq, \ldots, \geq \lambda_p$ *such that* $\subset$ *is spanned by* $[\beta_1, \ldots, \beta_q]$*.*
(Tarpey and Li, 1995)

Rather than considering the entire $\mathbb{R}^p$ in the search for self-consistent points these results help by limiting the search to subspaces of $\mathbb{R}^p$. For elliptical distributions, the search for self-consistent points is narrowed down to only spaces spanned by principal component axes. If the search is for a set of principal points of an elliptical distribution then subspaces of dimension $q = 1, 2, \ldots, \min\{k - 1, p\}$ that are searched are spanned by the first q principal component axes.

2.4 Estimation of Principal Points

It has been noted that principal points can be estimated from a random sample when the distribution is unknown. Let X be a p-variate random vector with unknown distribution function $F_{\mathrm{X}}(\cdot)$. Assume that the density function $f_{\mathbf{x}}(\cdot)$ exists and is also unknown. For a given value $k > 1$ assume the existence of a set of k points $W_k^{(pp)} = \{\xi_{\mathbf{1}}, \ldots, \xi_{\mathbf{k}}\}$ for X. Let $W_k^{(pp)}$ be unique. The aim is to use a random sample $\mathrm{X_1}, \ldots, \mathrm{X_n}$ from $F_{\mathrm{X}}(\cdot)$ to find a set of k points $\hat{W}_k^{(pp)} = \{\hat{\xi}_{\mathbf{1}}, \ldots, \hat{\xi}_{\mathbf{k}}\}$ to estimate the set $W_k^{(pp)}$. The set $\hat{W}_k^{(pp)}$ can be denoted as an estimate calculated from the observed sample.

The following are two basic methods for estimating principal points that can be used both for univariate and multivariate data. The sections that follow demonstrate estimators for univariate and multivariate data separately.

Unconstrained k-means Estimator

This estimator does not make any distributional assumptions. For the observed random sample $\mathbf{x_1}, \ldots \mathbf{x_n}$, define the empirical distribution function $\hat{F}(\cdot)$ as follows:

$$\hat{F}(\mathbf{x}) = \sum_{\mathbf{j=1}}^{\mathbf{N}} \mathbf{I}(\mathbf{x_j}, \mathbf{x}), \quad \forall \ \mathbf{x} \epsilon \mathbb{R}^{\mathbf{p}} \tag{2.3}$$

$$I(\mathbf{x_j}, \mathbf{x}) = \prod_{i=1}^{p} I(x_{ji} \leq x_i)$$

where $I(\cdot)$ is the indicator function.

The empirical distribution function puts an equal mass of $\frac{1}{n}$ on each of the sample points x_j. The function $\hat{F}(\cdot)$ is used to estimate the unknown distribution function $F_X(\cdot)$. The self-consistency algorithm for self-consistent points can be applied to the empirical distribution $\hat{F}(\cdot)$ to find a set of self-consistent points for the data. The self-consistency algorithm for $\hat{F}(\cdot)$ simplifies to the well-known k-means algorithm. The k-means algorithm is outlined as follows:

***k*-Means algorithm.** *An arbitrary initial set of k points is chosen* $\hat{W}_k^{(0)} = \{\hat{\xi}_{\mathbf{1}}^{(0)}, \ldots, \hat{\xi}_{\mathbf{k}}^{(0)}\}$. *The following steps are iterated until convergence. Start with* $m = 0$

1. *Let the domains of attraction for the points in* $\hat{W}_k^{(m)}$ *be* $A_j^{(m)}$, *for* $j = 1, \ldots, k$.
   For each domain of attraction compute the conditional cluster mean $\hat{\xi}_j^{(m+1)} \equiv \sum_{x_d \epsilon A_j^{(m)}} \mathbf{x_d}$.
   This defines a new set of points $\hat{W}_k^{(m+1)} = \{\hat{\xi}_{\mathbf{1}}^{(\mathbf{m+1})}, \ldots, \hat{\xi}_{\mathbf{k}}^{(\mathbf{(m+1)})}\}$.
2. *Let* $m \leftarrow m + 1$ *and return to* **1**.

Define $\hat{W}_k^{(KM)}$ *as the final set of points to which the algorithm converges.*
(Hartigan, 1975)

Parametric k-means Estimator

The goal is to estimate k principal points from a sample $\mathbf{x_1}, \ldots, \mathbf{x_n}$ based on its distribution. The idea of the parametric k-means algorithm introduced by (Tarpey, 2007) is to run the k-means algorithm not on the raw data, but on a simulated data set with a huge sample size. The key is to simulate data from a distribution that is parametrically estimated.

The algorithm can be described as follows:
Let $\mathbf{x_1}, \ldots, \mathbf{x_n}$ denote a sample from a population with distribution $F(\cdot, \theta)$, where the parameter θ can be one-dimensional or a vector.

1. Estimate θ through maximum likelihood estimation obtaining $\hat{\theta}$.
2. Simulate a very large sample of size say, 100 000, from $F(\cdot, \hat{\theta})$.
3. Run the k-means algorithm on the simulated data set.

To implement this algorithm, the distribution $F(\cdot, \theta)$ must be specified. Exploratory data analysis and goodness-of-fit tests can be used to determine a reasonable distribution to use for the parametric k-means algorithm.

2.4.1 Univariate

Consider the random variable X with mean μ, distribution function $F_X(\cdot)$ and density function $f_x(\cdot)$. Based on an observed random sample $x_1, x_2, \ldots, x_n$ of X, different types of estimators for the set of principal points $W_k^{(pp)} = \{\xi_1 \leq \ldots \leq \xi_k\}$ will now be defined.

Quantile Estimator

The quantile estimator introduced by Tarpey (1997) is described as follows: For a set of principal points $W_k^{(pp)} = \{\xi_1, \ldots, \xi_k\}$ with $\xi_1 \leq \ldots \leq \xi_k$, there corresponds k values $\alpha_1 \leq \ldots \leq \alpha_k$ such that $\alpha_j = P(X \leq \xi_j)$. If $x_{[1]}, \leq \ldots \leq x_{[n]}$ represents the observed sampling order, the ξ_j is estimated by its sample quantile $x_{(\alpha_j,n)}$. The sample quantile estimator (QN) for the set of principal points $W_k^{(pp)}$ is then defined as the set corresponding to $W_k^{(QN)} = \{x_{(\alpha_1,n)}, \ldots, x_{(\alpha_k,n)}\}$.

Quick Estimator

The quick estimator, suggested by Stampfer and Stadlober (2002) is described as follows: Assume that the density function $g(\cdot)$ is defined by

$$g(x) = \frac{f_x^{\frac{1}{3}}(x)}{\int_{-\infty}^{+\infty} f_x^{\frac{1}{3}}(x)dx}$$

with $f_x(\cdot)$ the density function of the random variable X. The quick estimates rest upon an asymptotic result of Potzelberger and Felsenstein (1994). Let $G(\cdot)$ be the cumulative distribution function of $g(\cdot)$ and let $y_i = G^{-1}(i/k+1)$, $i = 1, \ldots, k$ be the corresponding $(i/(k+1))$-th quantiles of $G(\cdot)$.

The set $P_k = \{y_1, \ldots y_k\}$ has the following asymptotic property:

$$\lim_{k\to\infty} \frac{MSE(X, P_k)}{MSE(X, W_k^{(pp)})} = 1.$$

This illustrates that as the value of k gets larger, the set P_k is a good approximation to $W_k^{(pp)}$. In general $f_x(\cdot)$ is unknown. It needs to be estimated nonparametrically by using density estimation. The estimated $\hat{g}(\cdot)$ is found by substituting the density estimate of $f_x(\cdot)$ and $\hat{G}(\cdot)$ is found correspondingly. The quantiles of $\hat{G}^{-1}(i/(k+1))$ are used as estimators of the principal points in $W_k^{(pp)}$.

The following estimators are variants of the k-means algorithm established by Stampfer and Stadlober (2002).

Smooth k-means Estimator

Instead of estimating the unknown distribution function $F_x(\cdot)$ by the empirical distribution function $\hat{F}(\cdot)$, a smoothed non-parametric estimate $\tilde{F}(\cdot)$ can be found using a density estimate of $f_x(\cdot)$. The self-consistency algorithm is then applied to the estimated distribution function $\tilde{F}(\cdot)$ leading to the smooth k-means (SKM) estimator $\hat{W}_k^{(SKM)}$. The other two variants of the k -means estimator, the KMS1 and KMS2 estimator are based on symmetry assumptions.

KMS1 Estimator

The assumption is that the distribution of the random variable is symmetric about its mean μ, i.e.

$$F_x(x) = 1 - F_x(2\mu - x)$$

The alternate empirical distribution $\hat{F}_{sym}(\cdot)$ can be defined as

$$\hat{F}_{sym}(x) = \begin{cases} \frac{1}{2N}\sum_{j=1}^{N}(\mathrm{I}(x_j \leq x) + \mathrm{I}(x_j \geq 2\mu - x)) & x \leq \mu \\ 1 - \hat{F}_{sym}(2\mu - x) & x > \mu \end{cases} \tag{2.4}$$

Under the assumption of symmetry, Stampfer and Stadlober (2002) show that $\hat{F}_{sym}(\cdot)$ is an unbiased estimator for $F_x(\cdot)$ and that its variance is smaller than the variance of the empirical distribution function $\hat{F}(\cdot)$ defined in equation 2.3. Usually μ is unknown and can be estimated by $\hat{\mu}$, which can be the sample mean or median. When $\hat{\mu}$ is substituted for μ in 2.4, $\hat{F}_{sym}(\cdot)$ corresponds to the empirical distribution of the augmented sample $x_1, x_2, \ldots, x_n, 2\hat{\mu} - x_1, 2\hat{\mu} - x_2, \ldots, 2\hat{\mu} - x_n$.

The KMS1-estimator $\hat{W}_k^{KMS1}$ is found by using $\hat{F}_{sym}(\cdot)$ in the self-consistency algorithm. This corresponds to performing the k-means algorithm on the augmented sample.

KMS2 Estimator

Under the assumption that the principal points $W_k^{(pp)}$ consists of points that are symmetrical about the mean μ of the random variable X.
Define the transformed sample:

$$x_j^* = \begin{cases} x_j & x_j > \hat{\mu} \\ 2\hat{\mu} - x_j & x_j \leq \hat{\mu} \end{cases}$$

with $\hat{F}^*(\cdot)$ being its empirical distribution function. The idea behind the KMS2 is to use $\hat{\mu}$ as a symmetry point and find one half of the estimated principal points using all the $\{x_j^*\}$ and then reflecting them about $\hat{\mu}$ to give the other half of the symmetric principal points. A distinction must be made between k being even and odd.

If k is even, the unconstrained k-means algorithm on all $\{x_j^*\}$ is performed to find a set of $\frac{k}{2}$ of estimated principal points $\{\hat{\xi}_1, \ldots, \hat{\xi}_k\}$. They are reflected about $\hat{\mu}$ to find the KMS2 estimator of $W_k^{(pp)}$.

$$\hat{W}_k^{KMS2} = \{2\hat{\mu} - \hat{\xi}_{\frac{k}{2}} < \ldots < 2\hat{\mu} - \hat{\xi}_1 < \hat{\xi}_1 < \ldots < \hat{\xi}_{\frac{k}{2}}\}$$

If k is odd then $\xi_{[\frac{k}{2}]} = \mu$ and let $\hat{\xi}_{[\frac{k}{2}]} = \hat{\mu}$. The unconstrained k-means algorithm is performed on all the $\{x_j^*\}$ to find a set of $(k-1)/2$ estimated principal points $\{\hat{\xi}_1 < \ldots < \hat{\xi}_{(k-1)/2}\}$. They are reflected about $\hat{\mu}$ to find the KMS2 estimator of $W_k^{(pp)}$.

$$\hat{W}_k^{KMS2} = \{2\hat{\mu} - \hat{\xi}_{(k-1)/2} < \ldots < 2\hat{\mu} - \hat{\xi}_1 < \hat{\xi}_{[\frac{k}{2}]} = \hat{\mu} < \ldots < \hat{\xi}_{(k-1)/2}\} \tag{2.5}$$

2.4.2 Multivariate

The following methods of estimation for multivariate distributions are introduced in (Flury, 1993).

Subspace Constrained Estimator

If the assumption of ellipticity is made, then the principal subspace theorem can be used to find semi-parametric estimators constraining the search for principal points to the principal component subspaces of the empirical distribution. Therefore minimizing $\sum_{j=1}^{n} d^2(\mathbf{x_j}|\mathbf{y_1}, \ldots, \mathbf{y_k})$ subject to the constraint that $\{\mathbf{y_1}, \ldots \mathbf{y_k}\}$ lie in the subspace spanned by the first q principal components, with q specified. Computationally, this amounts to running a k-means algorithm on principal component scores in dimensions $1, \ldots, q$.

Pattern Constrained Estimator

If certain structural assumptions can be made about an unknown distribution, this can often result in finding good estimators for self-consistent points. Symmetric properties of patterns of distributions often result in similar symmetric patterns for the sets of self-consistent points.

There exist numerical evidence that for the multivariate normal distribution, the principal points form symmetrical patterns. If the assumption can be made that these patterns also hold for a larger class of distributions then another type of semi-parametric estimator for the principal points can be defined by finding the set of points which minimise the MSE, under a particular pattern constraint.

There are three main pattern constraints, particularly for $k = 4$. They are, namely the line pattern, the cross pattern and the rectangular pattern. This can be done by setting up a grid, and finding the coordinates of the k points constrained to a pattern which minimise the MSE. Figure 2.3 illustrate the line, cross and rectangular patterns respectively.

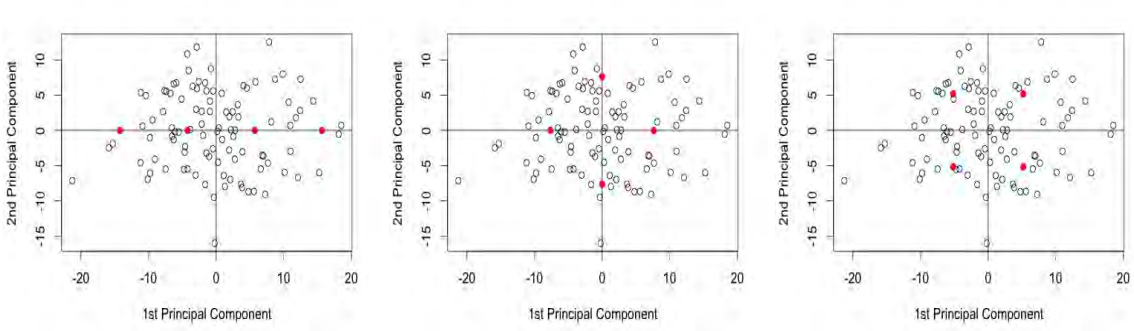


Figure 2.3: Various symmetric patterns

2.5 Evaluation of Estimators of Principal Points

The question naturally arises which estimator in Section 2.4 is used for finding the best set of principal points. There are a few measures of performance described in (Flury, 1993) which are all based on the minimising property of principal points.

For a given set of estimates $\hat{\xi}_1^{(m)}, \ldots, \hat{\xi}_k^{(m)}$, for a fixed k, and m representing the methods defined in 2.4, the *actual mean squared deviation* is defined as:

$$\text{AMSD}_x(m,k) = E_X\{d^2(X|\hat{\xi}_1^{(m)}, \ldots, \hat{\xi}_k^{(m)}\}, \tag{2.6}$$

where the expectation is with respect to the distribution of X, the $\hat{\xi}_i^{(m)}$ is fixed. Under repeated sampling, the $\text{AMSD}_x(m,k)$ are random variables, and the comparison between methods can therefore be based on the *expected actual mean squared deviation*,

$$\text{EMSD}_x(m,k) = E\{\text{AMSD}(m,k)\} \tag{2.7}$$

Here the expectation is with respect to repeated sampling of estimators $\hat{\xi}_i^{(m)}$. Clearly $\text{EMSD}(m) \geq P_x(k)$, and the method preferred, for a given k, would minimise equation 2.7.

The following two measures does not require for the distribution of X to be known and is applicable to practical situations. For a sample $\mathbf{x_1}, \ldots, \mathbf{x_n}$, the *sample mean squared deviation* is defined,

$$\text{SMSD}(m,k) = \frac{1}{n}\sum_{j=1}^{n} d^2(x_j|\hat{\xi}_1^{(m)}, \ldots, \hat{\xi}_k^{(m)}\} \tag{2.8}$$

The SMSD suffers from the same limitation as the coefficient of determination in regression and the re-substitution method of estimating errors in discriminant analysis: it reflects performance of an estimate as measured on the same data used in fitting the parameters. A better criterion should be based on prediction. The following measure has been inspired by what is knows as the *predicted sum of squares* statistic in regression, which is based on 'leave-one-out' residuals.

The idea is to omit the j-th observation from the process of estimating principal points, and then to measure the distance between the omitted observation and the nearest principal point. The process is repeated for all n observations in the sample, and then averaging the squared distances, the *predictive mean squared deviation* is obtained:

$$\text{PMSD}(m,k) = \frac{1}{n}\sum_{j=1}^{n} d^2(x_j|\hat{\xi}_{1,-j}^{(m)}, \ldots, \hat{\xi}_{k,-j}^{(m)}\} \tag{2.9}$$

where $\hat{\xi}_{i,-j}^{(m)}$ is the estimate of the i-th principal point, using method m, and omitting observation j. For a given set of data, the method that minimises the PMSD is preferred.

2.6 A criterion for selecting k

Typically, the value of k is chosen before the principal points of a distribution are estimated. The question arises what is the best value of k. The usual approach is to repeat the computations described in the previous sections successively for different values of k. The optimal value of k is selected at which the function $P_x(k)$ in 2.1 is a minimum. However using 2.1 as a criterion, only creates a monotonically decreasing function of k. So as the value of k increases, the objective function decreases, thus making it hard to choose k.

Krzanowski and Lai (1988) discusses an alternative method to optimally select k. The following need to be computed:

- $\text{Diff}(k) = (k-1)^{\frac{2}{p}} P_x(k-1) - k^{\frac{2}{p}} P_x(k)$, where p is the number of variables considered.
- $C_x(k) = \left| \frac{\text{Diff}(k)}{\text{Diff}(k+1)} \right|$

The function $C_x(k)$ is calculated for different values of k and the optimal value of k is the value that maximises $C_x(k)$. This criterion provides a useful stopping rule for the $P_x(k)$ function. However, care should be taken when choosing large values of k. For large values of k, the value of $C_x(k)$ will tend to become very large. This maximum is only a result of division of a very small number by an even smaller one, and would usually be rejected as being artificial.

2.7 Summary

In this chapter, principal points were defined as a method of discretising a distribution. They were also defined as those sets of self-consistent points which globally minimise the MSE. Theoretical properties of principal points and ways of computing them when the data comes from a known distribution, were studied. The problem of estimating principal points from a finite sample, whose distribution is unknown, was also discussed. Different estimators of principal points and ways of comparing them were examined. Lastly, a criterion was introduced to determine the optimal value of k.

In the next chapter, the focus will be shifted from discrete self-consistent approximations, principal points, to smooth self-consistent approximations, called principal curves. The subsequent chapter will be on the methods of estimation of principal points. The results and procedures will be applied to a univariate and multivariate data set.

Chapter 3

Principal Curves

3.1 Introduction

The theory of principal points was introduced as a method of approximating a distribution by discretisation. Principal points are self-consistent points which give an optimal representation of a distribution in terms of the mean-squared error (MSE). Alternative techniques of approximation involve the reduction of the dimension of a multivariate distribution. Often the probability distribution of a p-variate random vector X is concentrated around a lower dimensional space. Dimension reduction is referred to finding an optimal lower dimensional manifold to approximate X.

Pearson (1901) introduces a one-dimensional line that approximates a p-variate random vector by minimising the sum of squared orthogonal distances to the line. This line, also called the first linear principal component approximation goes through the mean of the distribution and is parallel to the eigenvector of the sample covariance matrix, Σ corresponding to the largest eigenvalue. The first principal component line treats the variables as symmetrical as opposed to having a response and predictor variables as in linear regression. The aim in linear regression analysis is to seek a rule for predicting the response using a predictor variable. Figure 3.1 illustrates the differences between linear regression lines and the first principal component line.

The first principal component line is the foundation of principal curves. Principal curves are non-linear generalisations of the first principal component line. Instead of summarising the data with a straight line, a smooth curve is used. These curves, like linear principal components, focus on the orthogonal or shortest distance to the points. More formally principal curves are smooth curves that are self-consistent for a distribution or data set. This means that any point on the curve coincides with the average of the observations projected onto this point.

This chapter will begin with focus on linear principal components and the link made with self-consistent approximations. The theoretical properties of the non-linear extension of principal components, namely principal curves will then be discussed. The computation and estimation of principal curves will be illustrated using the principal-curve algorithm - a version of the self-consistency algorithm. The subsequent section studies the extension of principal curves to higher dimensional principal surfaces.

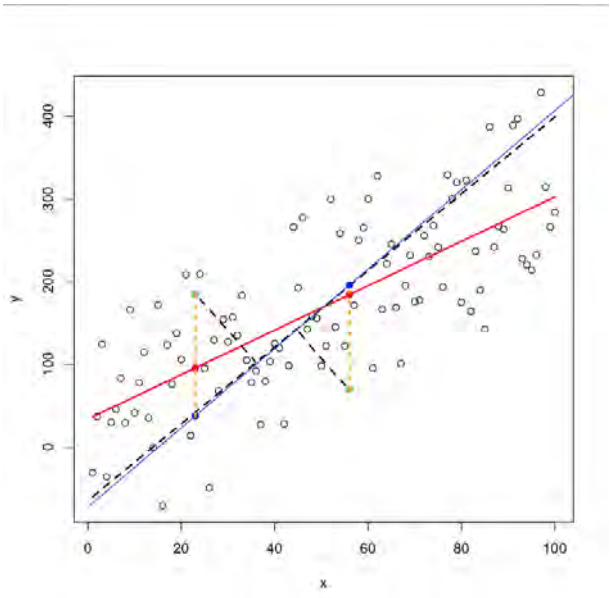


Figure 3.1: Red line represents regression line when y is used as the response, blue line represents regression line when x is used as the response and dashed line represents the first principal component line. Two points (green) are projected orthogonally onto the principal component line (black dashed line).

3.2 Linear Principal Components

Pearson (1901) approaches principal component analysis in the following way: If X is an nxp matrix with $n > p$ and rank$(X) = p$, what is the best rank in a q dimension, $q < p$ approximation to X? Consider approximating X with a random vector $\mathbf{Y}$ with a q-dimensional hyperplane ς.

The random vector can be defined according to the minimal distance as:

$$\mathbf{Y} = \mathbf{y} \quad \text{iff} \quad \mathrm{X} \epsilon A_y \;\; \forall\, \mathbf{y} \epsilon \varsigma$$

where A_y is the domain of attraction as defined in Chapter 2. This idea is translated into the framework of self-consistency. The random vector $\mathbf{Y}$ is the orthogonal projection of X onto the hyperplane ς. For any point $\mathbf{y}$, its domain of attraction is exactly all the points in $\mathbb{R}^{\mathrm{p}}$ that project onto $\mathbf{y}$. The conventional way of measuring the optimal hyperplane ς is by the minimisation of the mean squared error (MSE). The following theorem gives a characterisation of the optimal q-dimensional orthogonal approximation $\mathbf{Y}$ of X.

Optimal q-dimensional orthogonal approximation. *Let* X *be a* p*-variate random vector with mean vector* μ_{X} *and covariance matrix* $\mathbf{\Sigma}_{\mathrm{X}}$ *and let* $\mathbf{Y}$ *denote the orthogonal projection of* X *onto a hyperplane of dimension* q*,* $1 \leq q \leq p$*. The spectral decomposition of the covariance matrix consists of eigenvectors* $[\mathbf{a}_1, \ldots, \mathbf{a}_p]$ *with associated ordered eigenvalues* $[\lambda_1, \ldots, \lambda_p]$*. Let the* q *columns of the matrix* A_q *:* p*x*q *be the* first q *orthonormal eigenvectors* $[\mathbf{a}_1, \ldots, \mathbf{a}_q]$*. The MSE*$(X, Y)$ *is minimal if* $\mathbf{Y}$ *is defined as:*

$$\mathbf{Y} = \mu_{\mathrm{X}} + A_q A_q^T (\mathrm{X} - \mu_{\mathrm{X}})$$

Proof: (Flury, 1997)

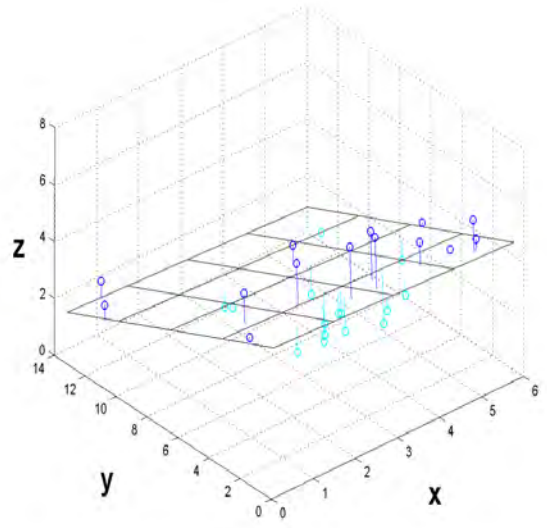


Figure 3.2: Graphical sketch of linear principal components in a 3 dimensional space with variables x, y and z.

Up until now, no assumptions have been made about the distribution of X. For a given distribution, the question arises of how well the MSE-optimal q-dimensional hyperplane approximates X. The following theorem gives a necessary condition for a random vector $\mathbf{Y}$ to be self-consistent for a q-dimensional hyperplane ς.

Necessary condition for Y to be self-consistent. *Let* X *be a* p*-variate random vector with mean vector* μ_{X} *and covariance matrix* $\mathbf{\Sigma}_{\mathrm{X}}$ *and let* Y *denote the orthogonal projection of* X *onto a hyperplane of dimension* q, $1 \leq q \leq p$. *The spectral decomposition of the covariance matrix consists of eigenvectors* $[\mathbf{a}_1, \ldots, \mathbf{a}_p]$ *with associated eigenvalues* $[\lambda_1, \ldots, \lambda_p]$. *Let the* q *columns of the matrix* A_q : pxq *be any* q *distinct orthonormal eigenvectors* $[\mathbf{a}_1, \ldots, \mathbf{a}_q]$. *If* Y *is a self-consistent approximation for* X, *then it follows that*

$$\mathbf{Y} = \mu_{\mathrm{X}} + AA^T(\mathrm{X} - \mu_{\mathrm{X}})$$

Proof: (Flury, 1997)

It follows from the above theorem that when projecting a p-variate random vector X orthogonally onto a q-dimensional hyperplane, the only prospectives for self-consistent approximations are the hyperplanes going through the mean vector X and is parallel to subspaces spanned by any q distinct eigenvectors of the covariance matrix $\mathbf{\Sigma}_{\mathrm{X}}$. Now that the link between linear principal components and self-consistent approximations have been made, the next section on principal curves is introduced.

3.3 Principal Curves

3.3.1 Theoretical Overview

Linear approximations are the simplest way to reduced the dimensionality of multivariate data to one dimension. When variables are treated as symmetrical, the first principal component axis can be used. Pearson (1901) approaches principal component analysis by approximating a p-variate data set with a straight line by minimising the orthogonal sum of squared distances to the lines, as described in section 3.1.

In general, distributions can have more complex structure in which a straight line will not be a sufficient approximation of the distribution. The curvature of a nonlinear smooth curve is needed for the approximation to go through the centre of the distribution. The following example illustrate the notion of a curve going through the centre of a distribution or data set.

A sample of size 200 drawn from a bivariate distribution [$X_1 \sim$ Uniform$(0, 2\pi)$ and $X_2 = sin(X_1) + X_1 + U$, where $U \sim$ Uniform$(-0.4, 0.4)$] is plotted in Figures 3.3 and 3.4. This data set will be referred to as the *S-shaped* data. The first figure illustrates the first principal component approximation to the data set. However this straight line does not summarise the general trend in the data well enough. Parts of the data lie entirely on either side of the line. This demonstrates that the straightness of the principal component line is too rigid to follow the nonlinearity in the data set. The second figure presents a smooth curve passing through the centre of the data, which shows a better approximation. The concept of principal curves can now be studied.

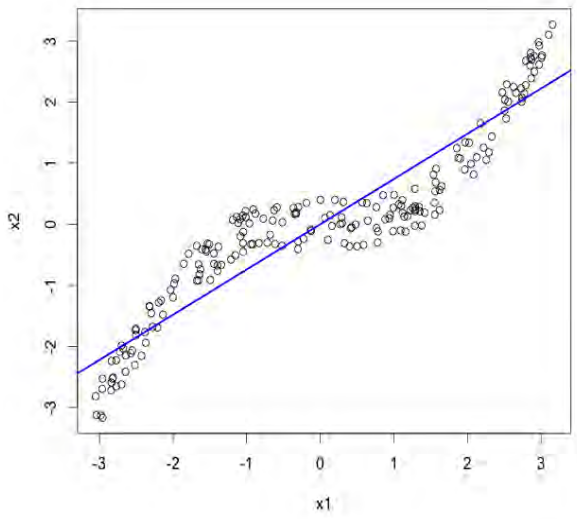


Figure 3.3: The first principal component approximation to the S-shaped data set

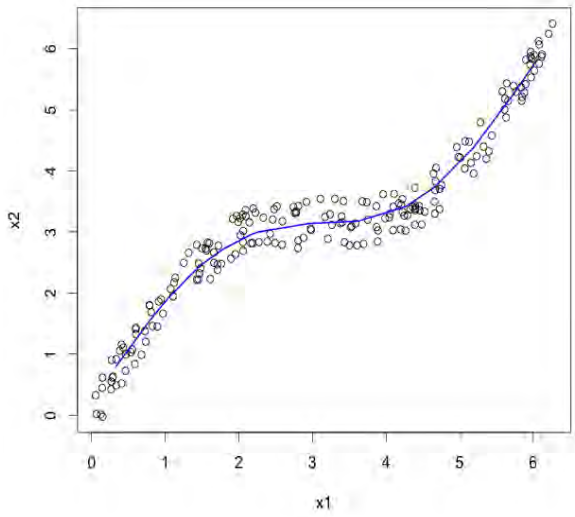


Figure 3.4

Principal curves are smooth one-dimensional curves that pass through the middle of the data providing a nonlinear summary of the data. Principal curves do not make any distributional assumptions and their shape is suggested by the data (Hastie and Stuetzle, 1989). A self-consistent or principal curve serves as a nonlinear generalisation of principal component axes. It is a smooth curve that has the self-consistency property, which means that each point on the curve is the conditional mean of all the points projecting onto it.

Before formally introducing the theory of principal curves, the definitions relating to one-dimensional curves are given as follows:

- A one-dimension curve in p-dimensional space is a vector $\mathbf{f}(\lambda)$ of p functions of a single variable λ. The curve $\mathbf{f}(\lambda)$ in $\mathbb{R}^{\mathbf{p}}$ is a functional association between the parameter value λ and a point in $\mathbb{R}^{\mathbf{p}}$.
- The p functions $\mathbf{f}(\lambda) = [f_1(\lambda), \ldots, f_p(\lambda)]$ are called coordinate functions and λ provides an ordering along the curve.
- If the first derivatives of p coordinate functions exist and are continuous, then $\mathbf{f}(\lambda)$ is a smooth curve.
- If $\mathbf{f}(\lambda)$ is smooth, the velocity vector $\mathbf{f}'(\lambda)$ is defined as the vector tangent to the curve at λ.
- If the velocity vector $\mathbf{f}'(\lambda)$ is smooth then the acceleration vector $\mathbf{f}''(\lambda)$ is defined as the vector tangent to $\mathbf{f}'(\lambda)$.
- The arc-length of a curve $\mathbf{f}(\lambda)$ from λ_0 to λ_1 is given by $\int_{\lambda_0}^{\lambda_1} ||\mathbf{f}'(\lambda)|| d\lambda$.
- A curve with $||\mathbf{f}'(\lambda)|| = 1$, for all values of λ is called a unit-speed parameterised curve.
- Any smooth curve with $||\mathbf{f}'(\lambda)|| > 0$, for all values of λ can be re-parameterised to make it unit-speed.

Consider X to be a continuous p-variate random vector and let $\mathbf{f}(\lambda)$ to be a parameterised smooth curve in z. The parameter λ can be chosen to be the arc-length along the curve from some fixed origin.

The *Projection index* can be defined as:

$$\lambda_{\mathbf{f}}(x) = \text{argmin}_{\lambda'}||x - \hat{\mathbf{f}}(\lambda')||^2. \tag{3.1}$$

The projection index $\lambda_{\mathbf{f}(x)}$ is the value of λ for which $\mathbf{f}(\lambda)$ is closest to $\mathbf{x}$. If there are a variety of values, the largest one is picked.

The curve $\mathbf{f}(\lambda)$ is called self-consistent or a principal curve of the random vector X if

$$E[\mathbf{X}|\lambda_{\mathbf{f}(x)} = \lambda] = \mathbf{f}(\lambda), \tag{3.2}$$

for almost every λ. For any particular parameter value λ, all observations that have $\mathbf{f}(\lambda)$ as their closest point on the curve are collected. If $\mathbf{f}(\lambda)$ is the average of those observations, and it holds for all λ, then $\mathbf{f}(\lambda)$ is a principal curve.

3.3.2 Computation of Principal Curves

In Chapter 2, it is noted that the term *computation* is used when the distribution of X is known. As principal curves are defined as self-consistent approximations to distributions, they can be computed by applying the general self-consistency algorithm. Hastie and Stuetzle (1989) proposes this as the principal curve algorithm. Consider the p-variate random vector X with known distribution $F_{\mathbf{x}}(\cdot)$ that has mean zero and assume that there exist at least one principal curve for X. The principal curve algorithm is described as follows:

Principal-Curve algorithm. *The following is an iterative algorithm*

1. *Initialisation*
   Set a smooth curve $\mathbf{f}^{(0)}(\lambda) = \lambda\mathbf{a}_1$ *to initialise the procedure, where* $\mathbf{a}_1$ *is the eigenvector corresponding to the largest eigenvalue of the covariance matrix of the mean zero vector* X.
   Set $\lambda^{(0)}(\mathbf{x}) = \lambda_{\mathbf{f}^{(0)}}(\mathbf{x})$ *and* $j = 0$.

2. *Expectation*
   Update $j \leftarrow j + 1$
   Set $\mathbf{f}^{(j)}(\lambda) = E[\mathbf{X}|\lambda_{\mathbf{f}^{(j-1)}}(\mathbf{X}) = \lambda]$

3. *Projection*
   Define $\lambda^{(j)}(x) = \lambda_{\mathbf{f}^{(j)}}(\mathbf{x})$ *for all* $\mathbf{x}$
   Transform $\lambda^{(j)}(x)$ *so that* $\mathbf{f}^{(j)}(\lambda)$ *is unit speed.*

Steps 2 and 3 are repeated until the reduction of the MSE reaches a certain threshold.
(Hastie and Stuetzle, 1989)

3.3.3 Estimation of Principal Curves

An observed sample $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is used to estimate principal curves for X as the distribution, $F_{\mathbf{x}}(\cdot)$, is unknown. Hastie and Stuetzle (1989) proposes an algorithm to estimate a principal curve from finite samples, modelled on the principal curve algorithm. The algorithm itself is an iterative procedure between an expectation and projection step until convergence.

The first principal component line translated to go through the sample mean is used to initialise the algorithm. The curve is represented by n pairs of values $(\lambda_i^{(j)}, \hat{\mathbf{f}}^{(j)}(\lambda_i^{(j)}))$ for the j-th stage of the algorithm. All $\lambda_i^{(j)}$ are assumed to be ordered but not necessarily corresponding to the observed sample. Their particular values are also not important. The smooth curve is approximated by a polygon by consecutively connecting the n points $\hat{\mathbf{f}}^{(j)}(\lambda_i^{(j)})$, called nodes. The curve is parameterised by setting $\lambda_1^{(j)} = 0$ and by letting $\lambda_i^{(j)}$ to be the arc-length along the polygon from $\hat{\mathbf{f}}^{(j)}(\lambda_1^{(j)})$ to $\hat{\mathbf{f}}^{(j)}(\lambda_i^{(j)})$.

The steps are described in detail as follows:

1. The *Expectation step*
   The conditional expectation $\mathbf{f}^{(j)}(\lambda) = E[\mathrm{X}|\lambda_{\mathbf{f}^{(j-1)}}(\mathrm{X}) = \lambda]$ as in the principal curve algorithm must be estimated. As the approximated curve $\hat{\mathbf{f}}^{(j)}(\lambda)$ only consists of n nodes the estimation of the conditional expectations can only take place at the n points $\lambda_1^{(j-1)} \leq , \ldots, \leq \lambda_n^{(j-1)}$.

   Typically the way of estimating the conditional expectation would be to gather all observations that project onto $\hat{\mathbf{f}}^{(j-1)}$ at $\lambda_i^{(j-1)}$ and find their mean. However, there is generally only one such observation, $\mathbf{x}_i$. Instead of using all points projecting onto $\hat{\mathbf{f}}^{(j-1)}(\lambda_i^{(j-1)})$ to estimate the conditional expectation, all points in the neighbourhood of $\lambda_i^{(j-1)}$ are used. This means that all of the observations $\mathbf{x}_k$ in the sample for which $\lambda_k^{(j-1)}$ is close to $\lambda_i^{(j-1)}$, is averaged. The way in which the conditional expectation is estimated for finite data sets is referred to as a *Scatterplot Smoother*.

   It should be noted that a bias is introduced when using the Scatterplot Smoother as the variance of the curve decreases. There will always be a variance-bias trade-off. However as long as the observations are close enough and the underlying conditional expectation is smooth, the bias introduced will be small.

2. The *Projection step*
   The n points are projected onto the current curve estimated in the Expectation step and the corresponding projection indices $\lambda_i^{(j)} = \lambda_{\mathbf{f}^{(j)}}(\mathbf{x}_i)$ for $i = 1, \ldots, n$ are found. The curve-length parameterisation of all the $\lambda_i^{(j)}$ should be considered at this step.

Iteration between these steps continues until the relative change in the sum-of-squared distances from the sample points to the curve is below a certain threshold. Hastie and Stuetzle (1989) suggest a threshold of 0.001.

3.3.4 Features of Principal Curves

- If the search for principal curves is restricted to a class of linear one-dimensional curves, then the only possibilities are the principal component axis of the distribution.

- The first principal component axis will correspond to a global minimum of the MSE.
- Generally it is not true that if a principal curve is searched for within the entire class of smooth curves it corresponds to a local minimum of the MSE. For this class of functions, it is proven in (Hastie and Stuetzle, 1989) that a curve is a principal curve if and only if it is a critical point of the MSE, i.e. the value of the derivative of the MSE at the curve is zero.
- Flury (1997) proves that if X is multivariate elliptical, then the principal components are self-consistent and are therefore principal curves.

3.4 Principal Surfaces

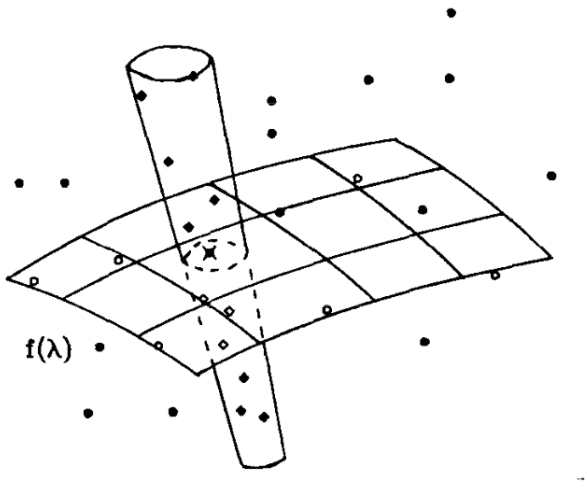
Principal surfaces have exactly the same form as principal curves, but are of higher dimension. The most commonly used is the two-dimensional principal surfaces, as dimension greater than two have a less attractive visualisation aspect. A principal surface consists of p- coordinate functions $\mathbf{f}(\lambda_1, \lambda_2) = [f_1(\lambda_1, \lambda_2)], \ldots, f(\lambda_p, \lambda_2)]$. The concept of smoothness as in the one-dimensional case can be generalised analogously. The projection index $\lambda_{\mathbf{f}}(\mathbf{x})$ of a point $\mathbf{x}$ in $\mathbb{R}^{\mathrm{p}}$ is also defined for the surface $\mathbf{f}(\lambda)$ as the parameter vector corresponding to the point on the surface closest to $\mathbf{x}$.

The two-dimensional surface $\mathbf{f}(\lambda)$ is a principal surface of a random vector X if

$$E[\mathrm{X}|\lambda_{\mathbf{f}(x)} = \lambda] = \mathbf{f}(\lambda), \tag{3.3}$$

for almost every λ.

Figure 3.5 demonstrates the situation. The plane spanned by the first and second principal components minimises the distance from the points to their projections onto any place.



surface.png

Figure 3.5: Each point on a principal surface is the average of the points that project there

All the results for principal curves also hold for principal surfaces. Estimating principal surfaces is similar to the principal-curve algorithm, but instead of using one-dimensional scatterplot smoothers, two-dimensional surface smoothers are used.

3.5 Summary

The study between principal points and principal curves and surfaces present strong similarities in terms of self-consistency. The idea of approximation has shown great depth in both principal points and principal curves and surfaces. Principal points give a discrete approximation to a distribution or data set, whereas principal curves are one-dimensional continuous approximations. Principal curves, just like principal points, have shown a connection with cluster analysis which can be used to find group structure in data by classifying to the nearest principal curve.

Chapter 4

Estimation of Principal Points

4.1 Introduction

The k principal points of a distribution are the k points that optimally represent a distribution in terms of the MSE. The theoretical properties of principal points for both univariate and multivariate distributions were studied in Chapter 2. When the distribution of the data is known, principal points of the distribution are said to be computed. However, when the distribution is unknown, principal points need to be estimated from a finite sample.

The main focus for this chapter is the estimation of principal points. The estimation is divided into applications for both univariate and multivariate distributions. For the univariate case, ecological data on the mass of bird species will be used. The familiar head dimension data will be used to illustrate the estimation of principal points for multivariate data. The criterion for selecting the best of value of k in each section will also be demonstrated.

4.2 Univariate distributions

4.2.1 Data

Understanding the relationships between body size of organisms and the environments in which they live has been a central concern for ecologists for decades. Body size is an interest in ecology because it is thought to offer an iterative, readily quantifiable measure of an organism's interaction with the environment (Roy, 2008). Considerable amount of support exists that claim that body masses in ecological communities occur in distinct groups, however the mechanisms leading to these clusters remain unclear.

The data consisted of Southern African birds from five different habitats : fynbos, forest, savannah, karoo and grassland. The range of habitats were deliberately selected as they have clear structural differences, fall into different rainfall regions and have relatively little overlap of generalist species. The idea is to aggregate or group the different bird species according to their body mass for each habitat. Body mass relationships provide a way of comparing different organisms using a common currency and of both extracting general biological principals and identifying exceptions or outliers. Table 4.1 gives the frequency of bird species in each habitat.

Table 4.1: Frequency of birds in each habitat

Fynbos	Forest	Savannah	Karoo	Grassland	**Total**
109	43	103	68	209	**532**

The importance of this data is to identify clusters that optimally group the bird species according to their body mass under each habitat. The concept of principal points can therefore be well applied to this data. Before estimating the principal points, Figure 4.1 demonstrate how the body mass is distributed amongst the different habitats. The densities are plotted using the log of the mass variable, log(mass), with a smoothing bandwidth of approximately 0.25 . The peaks of each density should give some indication as to where the principal points should be located.

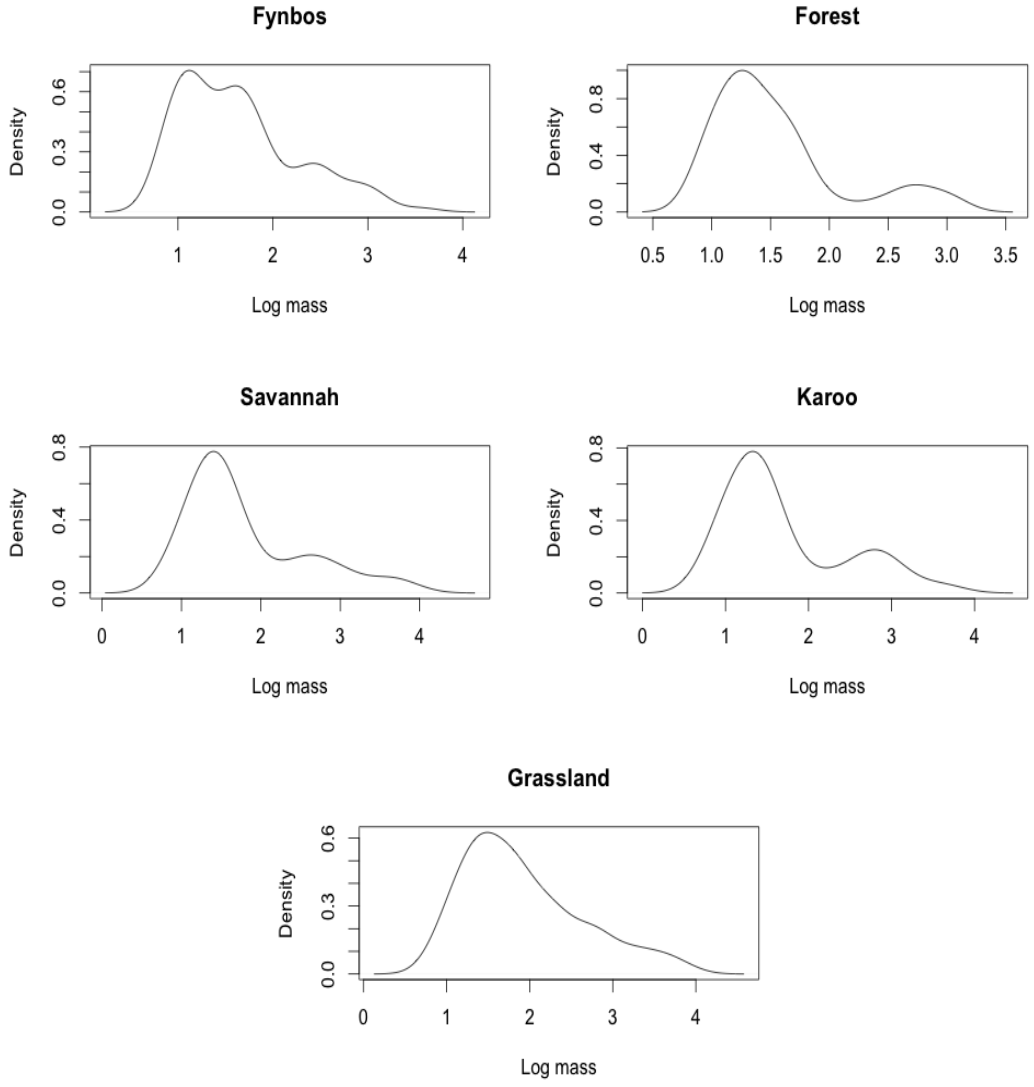


Figure 4.1: Density plots of different habitats

4.2.2 Estimation of Principal points

Typically before estimating the principal points, the number of principal points, k is selected. In this case, it would be good to know what value of k is best selected, to optimally discretise the distribution. The criterion $C_x(k)$ described in section 2.6 will help determine this value.

Table 4.2 shows the values of $C_x(k)$ for the different habitats. The value of k ranges from 1 to 5. The values highlighted in blue indicate the maximum value $C_x(k)$ for which k is the best value to choose for each habitat. Therefore the number of principal points chosen for the five habitats are 4, 3, 3, 3 and 3, respectively.

Table 4.2: Table showing values of C_k for $k = 1, \ldots, 5$ for different habitats.

	Habitat				
k	Fynbos	Forest	Savannah	Karoo	Grassland
1	-	-	-	-	-
2	0.917	0.360	0.359	1.012	2.273
3	0.378	1.480	1.951	4.638	7.192
4	2.703	0.512	1.531	0.266	0.092
5	0.569	0.852	1.340	2.934	0.573

Now that the value of k is determined for each habitat, the principal points can now be estimated. There are 7 different estimators mentioned in detail in Chapter 2. For brevity, only the results for the Fynbos habitat will be presented. For each estimator, a density plot for the Fynbos habitat will be provided indicating the 4 principal points represented by a coloured bullet point. They are given as follows:

1. **Unconstrained k-means Estimator**
   This estimator makes no distributional assumptions. The first 3 principal points are located closely, corresponding to the high peaks of the density plot.

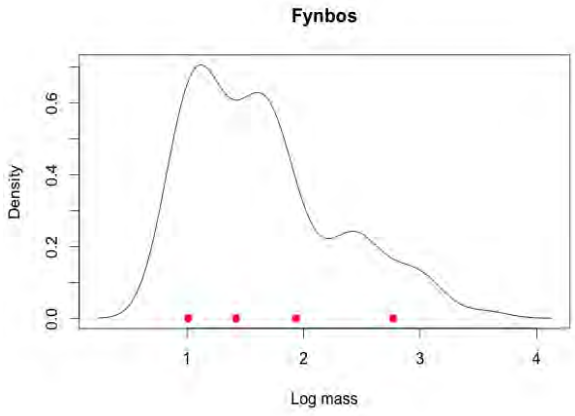


Figure 4.2: Density plot of Fynbos habitat with 4 principal points for the Unconstrained Estimator

2. **Parametric k-means Estimator**
   This estimator assumes the data comes from a known distribution, and its parameters are estimated via maximum likelihood estimation. The normal distribution is assumed for the body mass data, with MLE: $\hat{\mu} = 1.642$ and $\hat{\sigma} = 0.630$. The principal points for this estimator seem to be more spaced out and the first principal point is shifted more to the left of the data. The blue line represents the density of the simulated dataset of the Normal distribution.

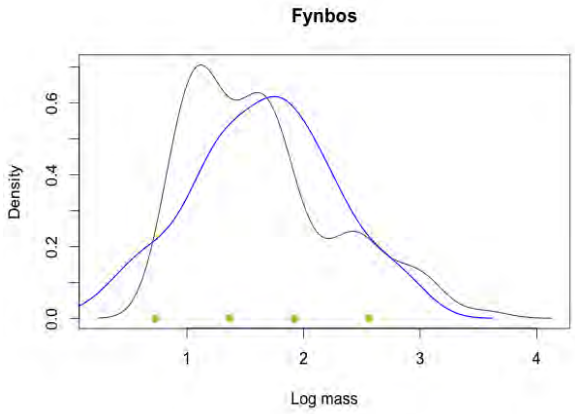


Figure 4.3: Density plot of Fynbos habitat with 4 principal points for the Parametric Estimator

3. **Quantile Estimator**
   This assumes a distribution and for this case the Normal$(0, 1)$ is assumed. The principal points are then estimated according to the quantiles. The first 2 principal points are close together near the first peak, and the 3rd near the second peak, while the 4th principal point is towards the end of the data.

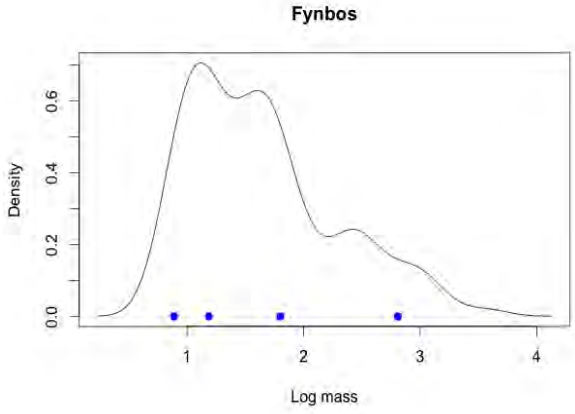


Figure 4.4: Density plot of Fynbos habitat with 4 principal points for the Quantile Estimator

4. **Quick Estimator**
   The Normal$(0, 1)$ distribution is assumed for this estimator. The density estimate of $f_x(\cdot)$ is estimated non-parametrically and used to find $g_x(\cdot)$. The principal points represent the corresponding quantiles of the cumulative function $G_x(\cdot)$. Like the *Parametric k-means* estimates, the points are evenly spaced out across the data.

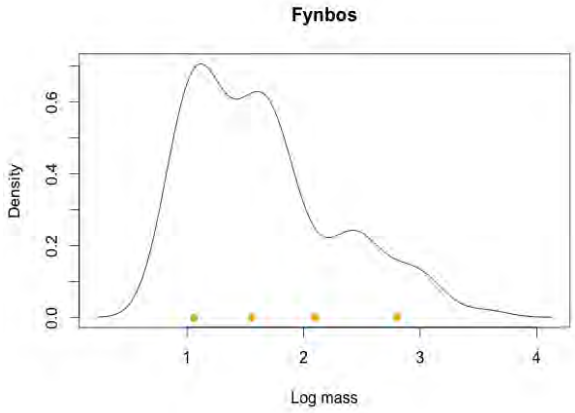


Figure 4.5: Density plot of Fynbos habitat with 4 principal points for the Quick Estimator

5. **Smooth k-means Estimator**
   This estimator uses the self-consistency algorithm on a smoothed non-parametric estimate $\tilde{F}(\cdot)$ found from a density estimate of $f_x(\cdot)$.

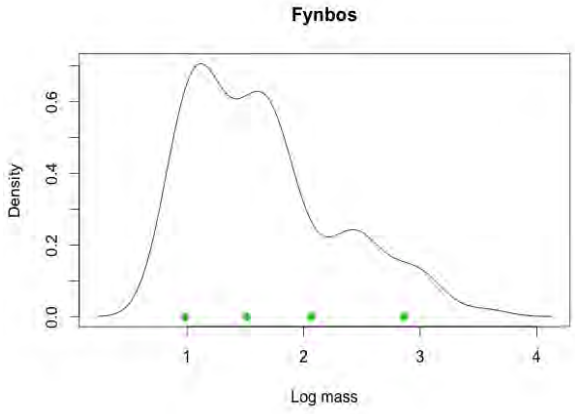


Figure 4.6: Density plot of Fynbos habitat with 4 principal points for the Smooth k-means Estimator

6. **KMS1 Estimator**
   The assumption is that the distribution of the data is symmetric about its mean. It is assumed for this estimator that the body mass data is symmetric about its median $= 1.560$ instead of its mean.

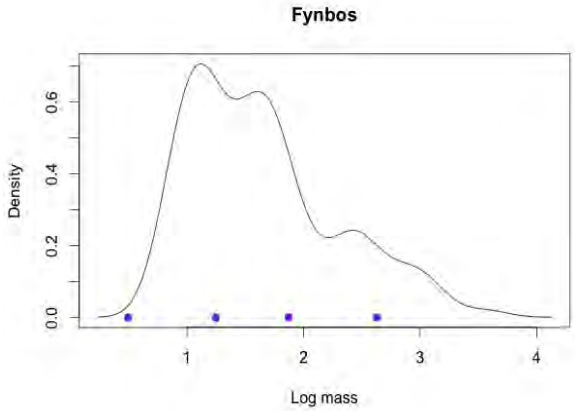


Figure 4.7: Density plot of Fynbos habitat with 4 principal points for the KMS1 Estimator

7. **KMS2 Estimator**
   Here the assumption is that the principal points are symmetric about the mean of the data. From the figure below, one can see how the principal points are centred about the mean of 1.643

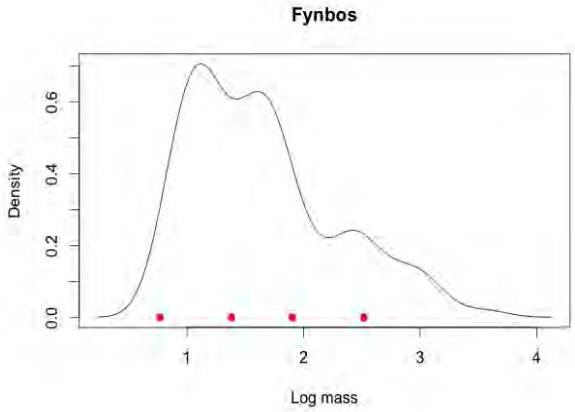


Figure 4.8: Density plot of Fynbos habitat with 4 principal points for the KMS2 Estimator

Table 4.3 gives the values of the four principal points under each estimator. There is a slight difference when comparing the 4 principal points respectively under each estimator, however a measure of performance studied in Chapter 2 can be used to evaluate which estimator is best to use. The two measures, namely the *sample mean squared deviation* SMSD(m, k) and the *predictive mean squared deviation* PMSD(m, k) are calculated.

Table 4.3: Estimates of principal points for the Fynbos habitat with an evaluation of each estimator

Estimator	Principal Points				SMSD	PMSD
Unconstrained	1.015	1.423	1.921	2.756	0.0354	0.0377
Parametric	0.724	1.366	1.919	2.561	0.0541	0.0568
Quantile	0.890	1.188	1.801	2.809	0.0398	0.0425
Quick	1.060	1.551	2.098	2.801	0.0341	0.0381
Smooth k-means	0.985	1.510	2.066	2.863	0.0373	0.0439
KMS1	0.484	1.249	1.864	2.621	0.0571	0.0609
KMS2	0.769	1.382	1.903	2.516	0.0538	0.0603

The estimator with the minimum value of SMSD is the *Quick Estimator*. However, the value for the *Unconstrained Estimator* is very close to the minimum and can be considered. The PMSD measure also votes for these estimators to be the most powerful.

To view the density plots of the other habitats with their respective principal points, refer to Appendix A.

4.3 Multivariate distributions

4.3.1 Data

Flury (1990) defined the term principal points in the problem of determining optimal sizes and shapes of protection masks for men in the Swiss army. The head dimension data also known as the anthropometric data, consisted of 6 variables that were measured on a sample of 200 men: *minimal frontal breadth* (MFB), *breadth of angulus madibulae* (BAM), *true facial height* (TFH), *length from glabella to apex nasi* (LGAN), *length from tragion to nation* (LTN), and *length from tragion to agnation* (LTG) (Flury, 1997).

Figure 4.9 represents the 200 men in the coordinate system of the first two principal components. Principal component analysis (PCA) allows one to view the data in 2 dimensions. The first and second principal components accounts for the most variability in the data and therefore not much information is lost in the transformation into a 2-dimensional space. For each estimator, the PCA plot will give perspective as to how the data is optimally assigned to different principal points under the criterion of each estimator.

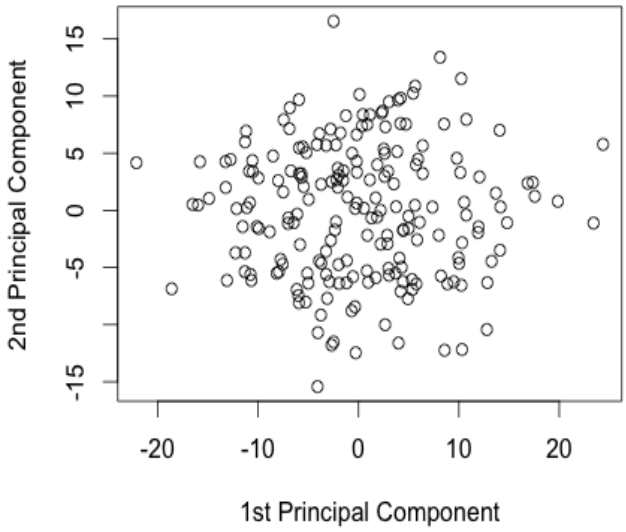


Figure 4.9: Plot of Head dimension data in 2 dimensions

4.3.2 Estimation of Principal Points

The number of principal points, k, selected for this problem is 4. The 6 estimators, explained in detail of Chapter 2 are used to estimate the principal points for the anthropometric data.

The principal points in Figures 4.10 to 4.15 are represented by the *yellow* points. The 200 men that are partitioned into 4 groups are indicated by the *blue, black, red* and *green* points respectively. From these figures, one can see how differently and similar each estimator groups the

data. More information on the list of assignments for the first 40 observations to the estimated principal points is provided in Table 4.4.

1. **Unconstrained k-means estimator**

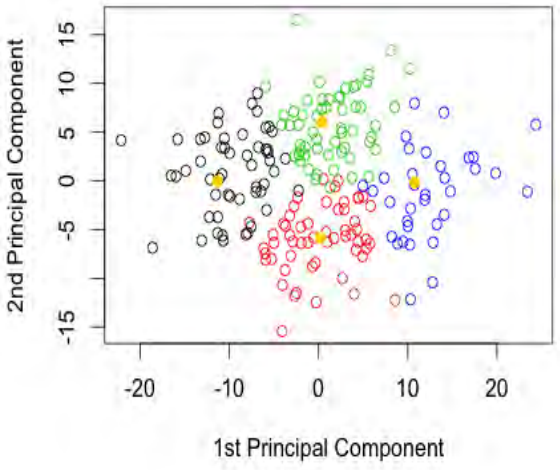


Figure 4.10: PCA plot of head measurement data with 4 principal points for the Unconstrained Estimator

2. **Parametric k-means Estimators**
   The data appears consistent with a multivariate normal distribution, shown in (Flury, 1990), and thus for this estimator it assumes the distribution of the multivariate normal.

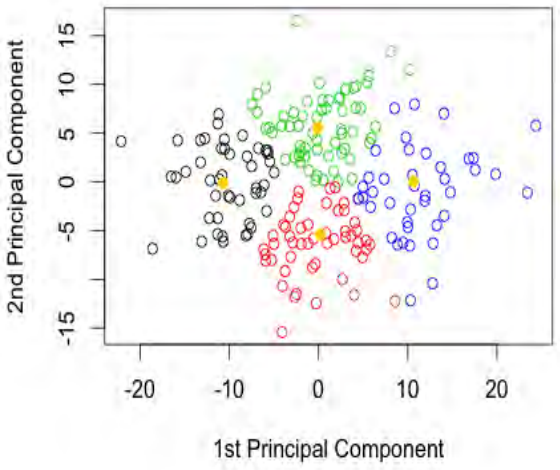


Figure 4.11: PCA plot of head measurement data with 4 principal points for the Parametric Estimator

3. **Subspace Constraint Estimator**
   This method required running the k-means algorithm on the principal component scores of the data in dimension $1, \ldots, q$. For this problem, the value chosen for q is 3, as a simulation conducted on all values of q, illustrated that $q = 3$ to be the optimal value with respect to the performance measures in Table 4.7.

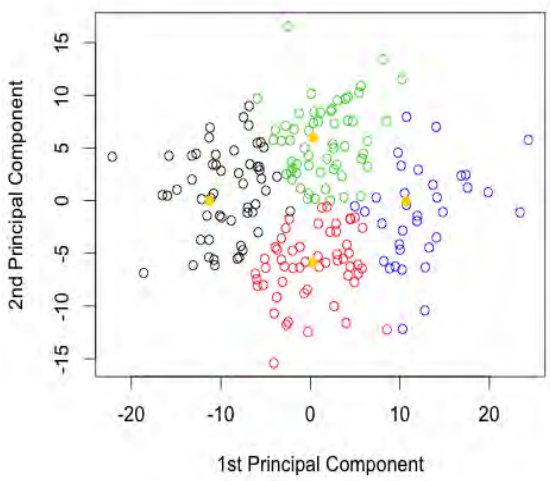


Figure 4.12: PCA plot of head measurement data with 4 principal points for the Subspace Constraint Estimator

4. **Pattern Constraint Estimator**

   (a) **Line Pattern**

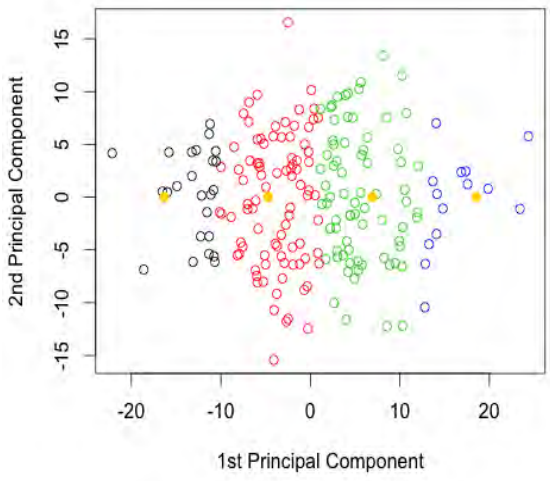


Figure 4.13: PCA plot of head measurement data with 4 principal points under the line pattern constraint

(b) **Cross Pattern**

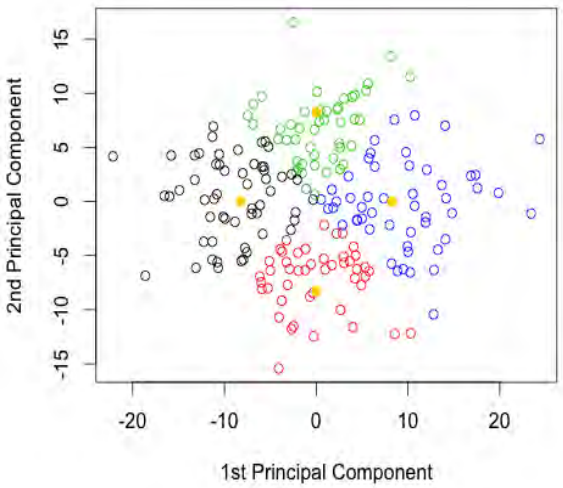


Figure 4.14: PCA plot of head measurement data with 4 principal points under the cross pattern constraint

(c) **Rectangular Pattern**

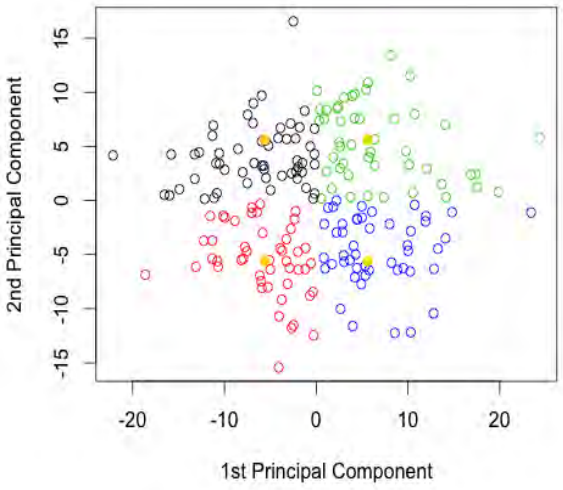


Figure 4.15: PCA plot of head measurement data with 4 principal points under the rectangular pattern constraint

Table 4.4: Assignments of first 40 observations to 1 of 4 principal points

Observation	Estimators					
	Unconstrained	Parametric	Subspace	Line Pattern	Cross Pattern	Rectangle Pattern
1	3	3	3	2	3	2
2	3	3	3	3	3	4
3	4	4	4	3	4	3
4	3	3	3	2	3	2
5	1	1	2	1	1	2
6	1	1	2	1	1	2
7	2	2	1	2	2	1
8	1	1	2	1	1	2
9	3	3	3	3	4	4
10	1	1	2	1	1	2
11	3	3	3	2	3	2
12	2	3	1	2	4	4
13	1	1	1	2	1	1
14	2	2	1	2	2	1
15	1	1	2	1	1	2
16	3	3	3	2	3	4
17	2	1	1	2	1	1
18	2	2	4	3	2	3
19	2	2	1	2	2	1
20	1	1	1	2	1	1
21	3	3	3	2	3	2
22	2	3	1	3	4	3
23	2	2	1	2	2	1
24	3	3	3	3	3	4
25	3	3	3	3	3	4
26	4	4	4	3	4	4
27	4	4	4	3	4	3
28	1	1	2	2	1	1
29	2	2	1	2	2	1
30	3	3	3	3	4	4
31	1	1	2	1	1	1
32	2	2	1	2	1	1
33	1	1	2	1	1	2
34	1	1	2	1	1	2
35	3	3	3	3	4	4
36	2	2	1	2	2	1
37	1	3	2	2	1	2
38	3	3	3	3	4	4
39	4	4	4	3	4	4
40	3	3	3	2	3	2

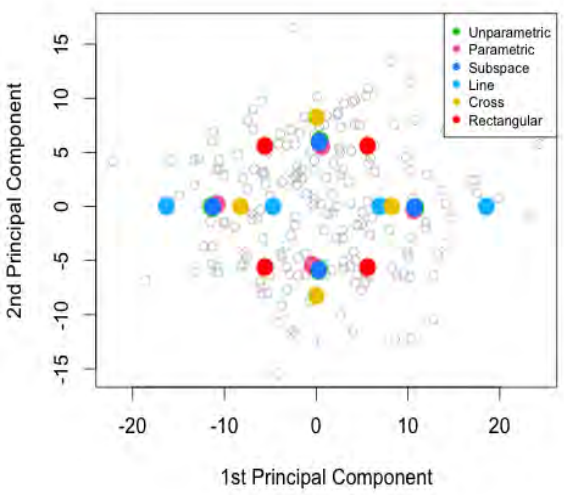


Figure 4.16

Observations in Table 4.4 are assigned to different principal points under each method of estimation. The line and rectangular pattern constraint seem to assign observations differently than the other estimators. Figure 4.16 demonstrates all principal point estimates under each method. To determine which estimator performs better, the SMSD and PMSD measures are calculated.

Table 4.5: Principal point estimates for first 3 methods

	Method of Estimation					
Points	*Unconstrained k-means*					
	MFB	BAM	TFH	LGAN	LTN	LTG
1	111.02	113.15	118.80	55.68	118.51	132.78
2	112.65	113.01	127.76	59.73	122.05	139.02
3	117.17	118.08	118.83	56.50	123.25	140.76
4	119.81	121.37	129.04	61.22	126.64	144.73
Points	*Parametric k-means*					
	MFB	BAM	TFH	LGAN	LTN	LTG
1	109.99	111.46	118.56	55.54	118.48	132.83
2	112.89	113.43	127.30	59.81	121.43	137.28
3	116.29	117.85	118.99	56.26	122.69	139.85
4	119.20	120.02	127.51	60.51	125.96	144.65
Points	*Subspace Constraint*					
	MFB	BAM	TFH	LGAN	LTN	LTG
1	111.47	112.75	118.83	55.94	117.72	133.38
2	112.85	113.31	127.89	60.11	122.20	138.49
3	116.48	118.21	118.93	56.29	123.43	141.05
4	119.92	121.15	129.13	60.79	127.31	144.33

Table 4.6: Principal point estimates for Pattern Constraint estimators

	Method of Estimation					
Points	*Line Pattern Constraint*					
	MFB	BAM	TFH	LGAN	LTN	LTG
1	107.46	109.20	116.60	54.60	113.09	129.69
2	112.63	113.98	121.19	57.01	119.60	136.19
3	117.80	118.75	125.79	59.42	126.10	142.70
4	122.97	123.53	130.38	61.83	132.61	149.21
Points	*Cross Pattern Constraint*					
1	111.33	112.65	119.79	56.28	119.37	134.21
2	111.63	122.51	129.65	60.84	121.64	137.49
3	117.82	109.32	116.46	55.14	122.83	140.18
4	118.12	119.17	126.32	59.70	125.10	143.45
Points	*Rectangular Pattern Constraint*					
	MFB	BAM	TFH	LGAN	LTN	LTG
1	110.72	111.47	125.34	58.77	119.86	134.74
2	113.70	115.71	116.30	54.86	120.68	136.59
3	115.75	116.12	129.81	61.12	123.79	141.08
4	118.73	120.36	120.77	57.21	124.60	142.93

Tables 4.5 and 4.6 provide the estimates of the principal points under each method together with their performance measures given in Table 4.7. The numerical results for all principal points estimates are not given in the coordinate system of principal points, but in their actual value according to the data values.

Table 4.7: Performance measures for the 6 methods of estimation

Estimator	SMSD	PMSD
Unconstrained	83.46	87.59
Parametric	81.12	86.39
Subspace Constraint	81.82	85.44
Line Pattern	99.60	100.84
Cross Pattern	86.48	87.91
Rectangular Pattern	89.60	90.76

The estimator which minimises the SMSD and PMSD measures are the *Parametric* and *Subspace Constraint* estimators respectively. These estimators are both satisfactory in estimating the principal points for the head dimension data, as their SMSD results as well as their PMSD values are fairly close. The *Line pattern constraint* proved to be the worst estimator under each performance measure.

4.4 Summary

This chapter provided a useful application of principal points to ecological data as well as to the head dimension data which has been previously researched by Flury (1993), Tarpey (1998) and Tarpey (2007). The head dimension data was the original example that motivated the term principal points. All methods of estimation were implemented for both sets of data, with a demonstration of how each estimator performed. With the help of two performance measures, the best estimator was determined.

In Chapter 3.1, the strong ties between cluster analysis and principal points were pointed out as well as their differences. (Flury, 1990) points out that "it would be interesting to study principal points of theoretical distributions that reflect group structure, such as finite mixtures, for which cluster analysis is meant to work". "The concept of principal points has therefore opened a new view of some concepts in cluster analysis. In the next chapter, the theory principal points is extended beyond its purpose and applied to computational methods in statistics.

Chapter 5

Principal Points in Computational Methods

5.1 Introduction

Computational methods can be described as constructing mathematical models and quantitative analysis techniques with the use of computers to analyse and solve scientific problems. It is typically the application of computer simulation and other forms of computation from theoretical computer science to problems in various scientific disciplines. For the context of this dissertation, computational methods can be defined as those statistical methods that are computationally intensive.

The common reason for methods to be computationally intensive is the size of a data set to be analysed. As the size of the data increases, a computational method can become more intensive. This chapter develops the idea of using a set of principal points in a computational method. Two computational methods will be described. The first being a exploratory data analysis tool called *bagplots* and the second a machine learning technique called *Support Vector Machines*. These methods are known to give accurate results, but are generally computer intensive. Section 5.2 reviews the times taken to estimate principal points from simulated datasets of different sizes, as estimating principal points from a very large data set can also be computer intensive.

5.2 Time Results

A study of the time taken to run the methods of estimation of principal points on simulated data sets of sizes $n = 50$, 500, 5000, 50 000 and 500 000, are evaluated in this section. Three types of data are simulated, a bivariate Normal distribution, a multivariate Normal distribution with 5 variables and a mixture of a Normal and two skew t-distributions with 2 variables. Ten samples of each data set are simulated and the time taken for the algorithms to run are then averaged.

This study is conducted in order to view how big sample sizes each method of estimation can handle. The number of principal points k can be chosen to be anything from 1 to $n - 1$. For each simulation in this case, k is set to half of the sample size, i.e. $k = \frac{n}{2}$. The reason for this choice is that the time taken to run any of the algorithms, takes the longest when $k = \frac{n}{2}$. The results therefore, shown for each type of dataset in Tables 5.1, 5.2 and 5.3, give the maximum time (in seconds) each algorithm takes to run.

Table 5.1: Time (in seconds) results showing for bivariate Normal Distribution

n	*Unconstrained*	*Parametric*	*Subspace*
50	0.0011	1.4190	0.0018
500	0.0074	7.1605	0.0073
5000	0.3106	19.065	0.2978
50000	26.7568	84.7334	26.9027
500000	2903.3020	3496.8060	3014.4210

Table 5.2: Time results showing for multivariate Normal Distribution

n	*Unconstrained*	*Parametric*	*Subspace*
50	0.0015	3.916	0.0012
500	0.0102	9.9543	0.0077
5000	0.4696	35.6576	0.2313
50000	36.01	109.9114	20.149
500000	3480.395	4026.9100	1830.5370

Table 5.3: Time results showing for mixture Distribution

n	*Unconstrained*	*Parametric*	*Subspace*
50	0.0689	15.4316	0.0018
500	0.0754	52.3792	0.0086
5000	0.376	393.6084	0.3038
50000	26.8101	795.8987	27.1381
500000	2915.142	3481.672	3016.746

The greater the sample size of each dataset, the time taken to run each method took to estimate the principal points increased linear, as expected. This can be seen graphically for the bivariate Normal example in Figure 5.1. For the first set of data - the bivariate Normal, each method estimated the principal points fairly quickly for $n = 50$, 500, 5000 and 50 000. For $n =$ 500 000, the methods took at least 50 minutes to run. For the data distributed Normal with 5 variables, the time results look fairly the same. The *Parametric k-means* method took the longest, as this method computes the MLE of the data, and simulates from a certain distribution with a huge sample size (for this study, $ns = 550\,000$).

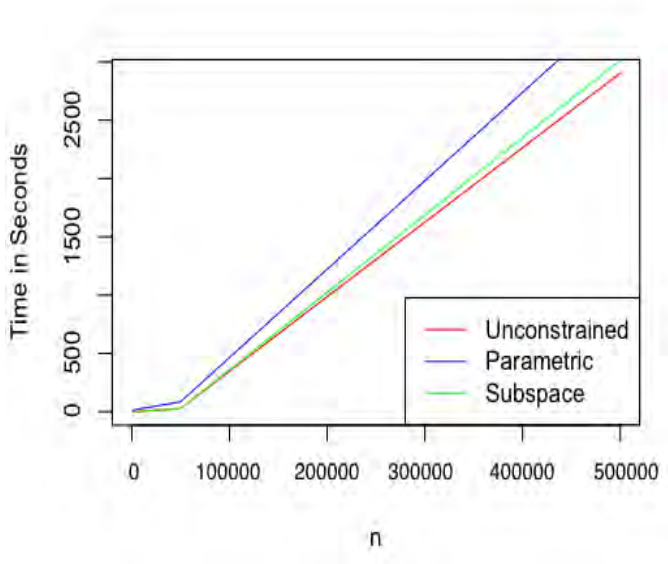


Figure 5.1: Graphical representation demonstrating that as the sample size increases, the time increases linearly for the bivariate Normal data set

The mixture distribution dataset is used to inspect whether the algorithms will influence the time taken to run for a more a complicated dataset than a symmetric normal distributed dataset. For $n = 50\,000$, the *Parametric k-means* method took almost 7 times longer than the previous datasets. However the times for $n = 500\,000$ looked similar. Above all, the *Unconstrained k-means* and *Subspace Constraint* estimators are quicker methods to use for estimating principal points, especially for big datasets.

5.3 The Bagplot: A bivariate boxplot

A bagplot is a bivariate generalisation of the univariate box plot. The univariate box plot, also known as the box-and-whiskers plot, was proposed by Tukey (1977) as a tool for exploratory data analysis. To illustrate the idea of a bagplot, the LGAN and LTN variables from the head-dimension data mentioned and used in Chapter 2 will be used as an example. The boxplots for the LGAN and LTN variables are shown in Figure 5.2.

A boxplot consists of a box from the lower quartile of observations to their upper quartile, with a crossbar at the median of the observations. Outside the box, the upper fence is given by

$Q_2 + 4(Q_3 - Q_2)$ and the lower fence by $Q_2 + 4(Q_1 - Q_2)$, where Q_j is the j-th quartile. The points lying outside the fences are flagged as outliers.

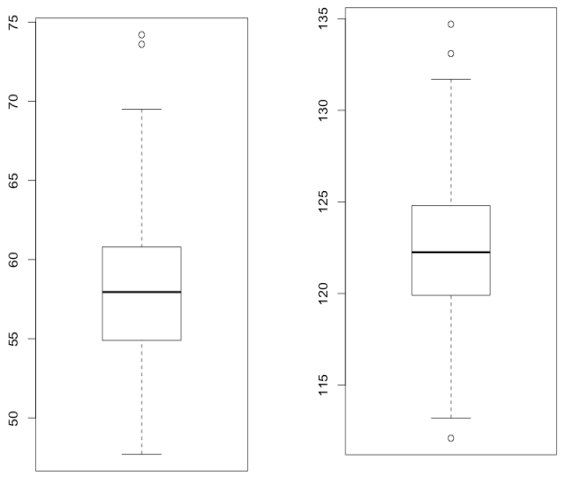


Figure 5.2: Boxplots of LGAN and LTN variables respectively.

Using the concept of *half space depth* (Tukey, 1975) which will not be discussed in this dissertation, a bivariate version of the box plot is introduced. Its main components are a bag that contains 50% of the data points, a fence that separates the inliers from outliers, and a loop indicating the points outside the bag but inside the fence. The resulting graph is called a bagplot. (Rousseeuw et al., 1999).

Consider the scatterplot in Figure 5.3. The depth median, the point with the highest half space depth, lies in the centre and is indicated by a red cross. The bag is the polygon drawn as a full line, with dark blue interior. The observations that lie outside of the bag but inside of the fence are indicated by a light blue interior . Any observations outside the fence, indicated by red points are outliers. There are two in this case.
Like a boxplot, the bagplot also visualises several characteristics of the data: its location shown by the depth median, its spread shown by the size of the bag, its correlation shown by the orientation of the bag, its skewness shown by the shape of the bag and the loop, and its tails shown by the points near the boundary of the loop and the outliers. By showing all the points in the bagplot, the advantages of the scatterplot is preserved in terms of local structure. (Rousseeuw et al., 1999).

In Rousseeuw et al. (1999), the construction of the bagplot using the half space location depth theory is studied. Two examples with an algorithm and implementation of the bagplot are also illustrated. Each function in the S-plus code for constructing bag plots are discussed. It is mentioned that as the number of observations n increase, the computational time for constructing a bagplot also increases. When n is very large, the function *bagplot()* in the *aplpack* package (Wolf and Bielefeld, 2013) in `R 2.15.0` (R Core Team, 2014), takes a random sample of 300 data points from the dataset to model the bagplot, as $n > 300$ takes very long to compute. Because a random sample is used, the bagplot produced will not be an accurate representation of the entire dataset.

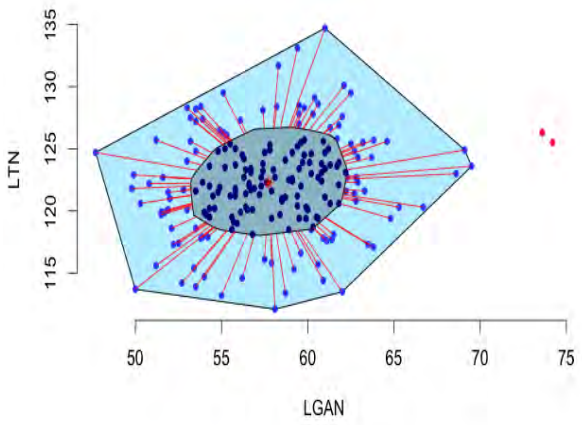


Figure 5.3: Bagplot of LGAN and LTN variables

The theory of principal points can be extended beyond its purpose to the problem of constructing a well represented bagplot of the data. Instead of using a random sample, that will construct a different bagplot each time it is run, the estimated principal points of the large dataset can be used. The quickest method of estimating principal points would be the Unparametric k-means algorithm as shown in Section 5.2.

The following example is used to illustrate the idea of principal points versus random sampling in the construction of bagplots. The data, extracted from the Centers for Medicare and Medicaid Services (www.cms.gov), consisted of charges and payments of 163065 hospitals made in the U.S for the financial year 2012. This large set of data, which will be referred to as the *Impatient* data, is used only for illustration purposes and no specific analysis relating to the contents of the data will be made.

Figure 5.4 demonstrates three bagplots of three random samples, each of size 300 drawn from the large dataset. The two variables used are *Average Covered Charges* and *Average Medicare Payments*. As expected, all three bagplots are not the same - in terms of their depth median, bag orientation and number of outliers. The bag plot with 300 estimated principal points in Figure 5.5 demonstrates a better representation of the data, as it reduces the randomness of the data chosen to construct a bag plot and all the data is considered.

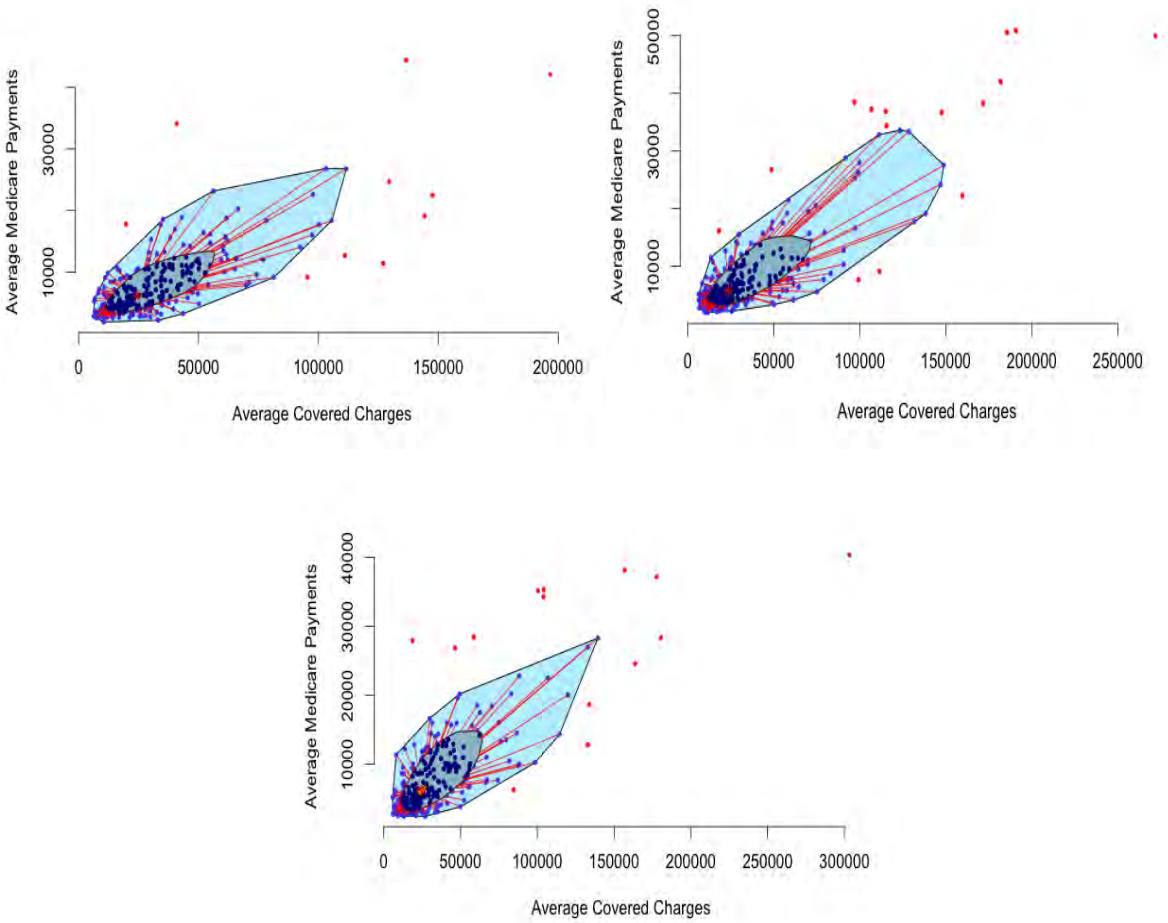


Figure 5.4: Bagplots of 3 random samples from the Inpatient data

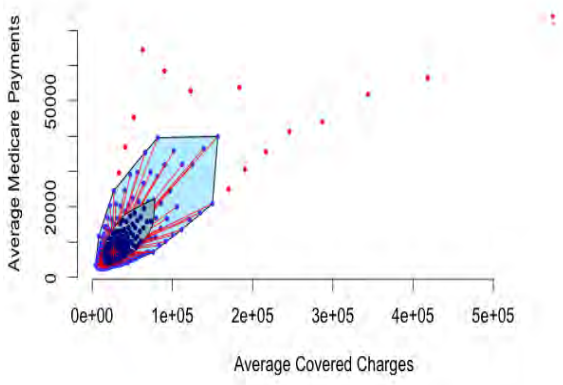


Figure 5.5: Bag plot of 300 estimated principal points from the Inpatient data

5.4 Support Vector Machines

In Chapter 1, two paradigms, namely *Supervised* and *Unsupervised Learning* were described. In Supervised Learning, a training sample $(\mathbf{x_1}, \mathbf{y_1}), \ldots (\mathbf{x_n}, \mathbf{y_n})$ is said to aid in the learning process in order to predict a future response $\mathbf{y}^*$. In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyse data and recognise patterns, used either for classification or regression analysis.

A support vector machine constructs a hyperplane or a set of hyperplanes in a high dimensional space which separates observations into different classes. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class, called a functional margin, since in general the larger the margin the lower the generalisation error of the classifier. Classification of new data points is done by a winner-takes-all strategy, in which the classifier with the highest output function assigns the class.

SVMs tend to have an unacceptable building time as the training data set becomes larger. SVMs, though accurate, are not preferred in applications requiring great speed, due to the number of support vectors being large. Principal points estimated from the data, can be used to model the SVM classifier in the attempt of improving the time aspect without reducing the accuracy of the classifier substantially. The accuracy is defined as the percentage of correctly classifying observations predicted on a test data set.

To illustrate this idea, an Activity Recognition data set built from recordings of 30 subjects doing different activities while carrying a waist-mounted smartphone with embedded inertial sensors, will be used. Human Activity Recognition (HAR) aims to identify the actions carried out by a person given a set of observations of him/herself and the surrounding environment. The HAR data set, available in the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.html) has been created using inertial data from smartphone accelerometers and gyroscopes, targeting the recognition of six different human activities. The six activities were *standing, sitting, laying down, walking, walking downstairs and upstairs*. A total of 561 features were extracted to describe each activity.

The data is split into two independent sets, where 70% of the data were selected for a training set and the remainder for a test set. An experiment is conducted on the 7352 observations in the training set using a multiclass SVM classifier for the six activities.

The results from the function *svm()* of package *e1071* (Meyer et al., 2014) in `R 2.15.0` (R Core Team, 2014) is used to classify new digits in the test set. The following function call to *svm()* can be made: `svm (train.Activity ~ ., data=HAR.train)`
where the variable *train.Activity* contain 7352 activity classes.

The classification results on the test data are shown in Table 5.4, where the rows represent the actual class and columns the predicted class. The overall accuracy of 95.3% for the test data composed of 2947 patterns. The time taken for the classifier to run took approximately 84 seconds.

Principal points are then estimated from the training data using the Unconstrained k-means estimator. For $k = 100, 200, \ldots, 7000$, the time taken in seconds to collectively estimate the principal points and to run the SVM classifier on those points, are noted. This time is denoted by T_k.

Table 5.4: Classification results for SVM classifier on full training data set

	W	WU	WD	S	SD	LD	Precision
W	482	6	8	0	0	0	97.2%
WU	14	456	1	0	0	0	96.8%
WD	7	28	385	0	0	0	91.7%
S	0	1	0	441	47	2	90.2%
SD	0	0	0	28	504	0	94.7%
LD	0	0	0	0	0	537	100%
Precision	95.8%	93.1%	97.7%	94.0%	91.5%	99.6%	**95.3%**

Figure 5.6 demonstrates values of k plotted against a time ratio. The time ratio at each value of k is $\frac{T_k}{T}$, where $T = 84$s represents the time taken to perform an SVM classifier on the full training data set. The figure only represents values of k, for which the time ratio is less than 1. Any value of the time ratio greater than 1, only means that T_k took longer than 84s which becomes of no interest. Thus any value of k chosen between 100 to 2000 will only mean that T_k is less than 84s. The time ratio is represented by a black line.

The accuracy A_k is the accuracy of the SVM classifier on each set of k principal points, for $k = 100, \ldots, 2000$. The accuracy ratio is then $\frac{A_k}{A}$, where $A = 0.952$ represents the accuracy of the SVM classifier on the full training data set. This is graphed for each value of k represented by a green line in the figure. A value of k needs to be chosen such that the time is reduced, but the accuracy is maintained. At $k = 1000$, the time is almost halved, and the accuracy does not decrease by a large amount. The values of the time and accuracy at $k = 1000$ are $T_{1000} = 45$s and $A_{1000} = 0.934$ respectively. Table 5.5 reveals the classification results of an SVM classifier on a set of 1000 principal points estimated from the training data set.

Table 5.5: Classification results for SVM classifier on a set of 1000 principal points estimated from training data set

	W	WU	WD	S	SD	LD	Precision
W	482	0	14	0	0	0	97.2%
WU	18	448	5	0	0	0	89.6%
WD	11	25	384	0	0	0	91.4%
S	0	0	0	424	63	4	87.1%
SD	0	0	0	54	478	0	89.8%
LD	0	0	0	0	0	537	100%
Precision	94.3%	94.7%	95.3%	88.7%	88.4%	99.3%	**93.4%**

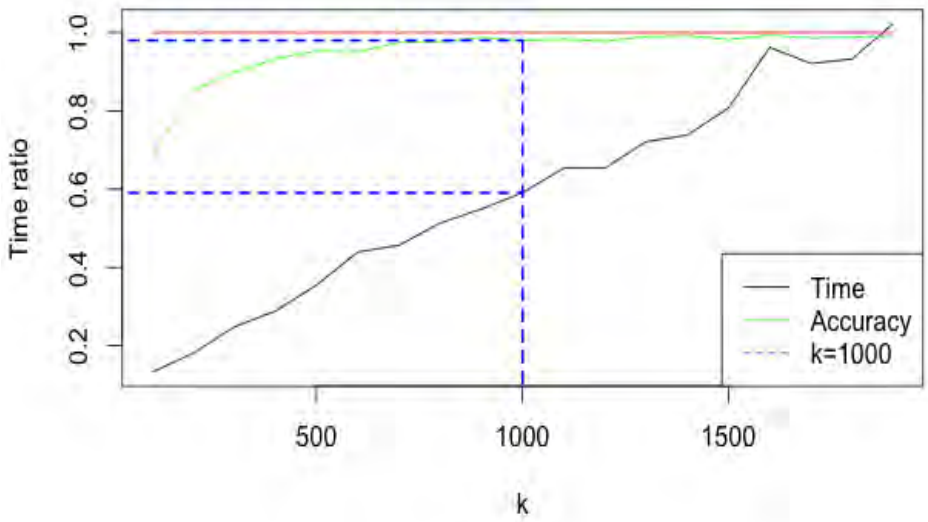


Figure 5.6: Plot of Time ratio vs number of principal points k

5.5 Summary

Principal points are those points that optimally group a data set into partitions. The technique of using principal points from a large data set is a better approximation than a random sample, as all observations are considered. There are many other statistical methods other than the bagplot idea, that become computationally expensive as the data becomes larger and random sampling has to be used. The phrase *Big Data* has become a popular term used to describe the exponential growth and availability of data, both structured and unstructured. It is known in the statistical framework that more data leads to better accurate analyses. Principal points estimated from Big Data can be used to perform an analysis, in the hope that the selection of those points will achieve a better analysis. As for machine learning techniques, the estimated principal points has shown to improve the computational time.

Chapter 6

Principal Curves in Digit Recognition

6.1 Introduction

The task of handwritten digit recognition has great importance and use such as online handwriting recognition on computer tablets, recognising zip codes on mail for postal mail sorting, processing bank cheque amounts, numeric entries in forms filled in by hand and so on. There are different challenges faced while attempting to solve this problem. The handwritten digits are not always of the same size, thickness, or orientation and position relative to the margins.

The goal, generally for this problem, is to implement a pattern classification method like multiclass SVM's to recognise the handwritten digits provided in a dataset of images of hand written digits ($0 - 9$). In this chapter, a further step is taken by introducing principal curves to the problem of digit classification. A subset of the data, available in The MNIST Database of Handwritten Digits (http://yann.lecun.com/exdb/mnist/), is used for this application and is made up of 8000 training images and 2000 testing images. Each image is a 28x28 grayscale (0-255) labeled representation of an individual digit. Figure 6.1 represent the first 100 digits within the training set.

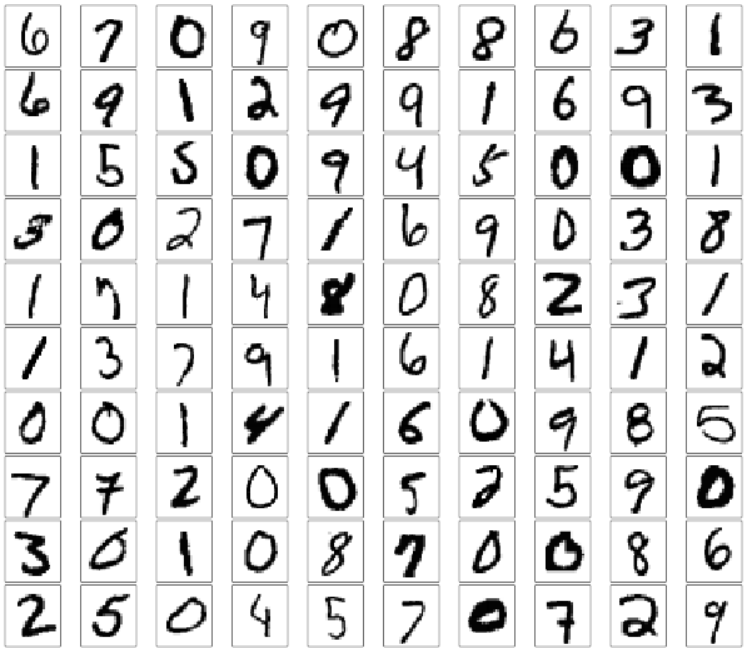


Figure 6.1: The first 100 digits within the training set

6.2 Multi-class SVM classifier

For the multi-class SVM, the function *svm()* of package *e1071* (Meyer et al., 2014) in `R 2.15.0` (R Core Team, 2014) is used to classify new digits in the test set. The following function call to *svm()* can be made:

```
svm(formula = trainDigit ~ ., data = digits, scale=F, kernel="radial", gamma=0.0001, cost= 10)
```

formula A symbolic description of the model to be fit. The variable *trainDigit* contain 8000 digit classes with 784 (= 28x28) variables.

data A data frame containing the variables in the model in the training set.

scale A logical vector indicating the variables to be scaled.

kernel The kernel used in training and predicting

gamma Parameter needed for all kernels

cost Cost of constraints violation

New digits are classified from an object of class *svm* containing the fitted model. For the test set of 2000 digit images, the SVM classifier gives an accuracy of 10.5%, and takes approximately 264s to run. Table 6.1 represents the classification results where the rows represent the actual class and the columns represent the predicted class.

Table 6.1: Classification results using the multi-class SVM classifier

	0	1	2	3	4	5	6	7	8	9
0	0	167	0	0	0	0	0	0	0	0
1	0	181	0	0	0	0	0	0	0	0
2	0	234	0	0	0	0	0	0	0	0
3	0	200	0	0	0	0	0	0	0	0
4	0	198	0	0	0	0	0	0	0	0
5	0	195	0	0	0	0	0	0	0	0
6	0	213	0	0	0	0	0	0	0	0
7	0	219	0	0	0	0	0	0	0	0
8	0	177	0	0	0	0	0	0	0	0
9	0	216	0	0	0	0	0	0	0	0

6.3 Principal curves classifier

In order to improve on the accuracy and computational time of the multi-class SVM classifier, principal curves can be fitted for each digit and subsequently used to classify new digits.

The following steps are performed on the training set of images:

1. The average for all digit images $(0-9)$ are computed.
2. All grayscale values in for each averaged 28x28 digit matrix are converted to a 0 or 1, where 1 represents the image.
3. The coordinates of all the 1's in the converted 28x28 averaged matrix from point **2** above, are transformed to a two-column matrix where the first coordinate represents a column coordinate and the second a row coordinate.
4. Ten principal curves are fitted on each of the ten coordinate matrices, using the function *principal.curve()* in the *princurve* package of Hastie and Weingessel (2013) in `R 2.15.0` (R Core Team, 2014).
5. A starting curve for each principal curve fitted is specified by a curve of the digit, since it is known what each digit looks like.

The following procedure will be illustrated with an example. For all the digit matrices classified as a **3**, the average 28x28 matrix, with converted grayscale values, is shown in Table 6.2. Table 6.3 represents a two-column matrix of coordinates for each value 1 in Table 6.2. A principal curve is fitted on the coordinate matrix for digit **3**. Refer to Appendix B for the complete R code for the above procedure. Figure 6.2 represent the fitted principal curves for each digit graphed together with their average coordinates from 8000 training images.

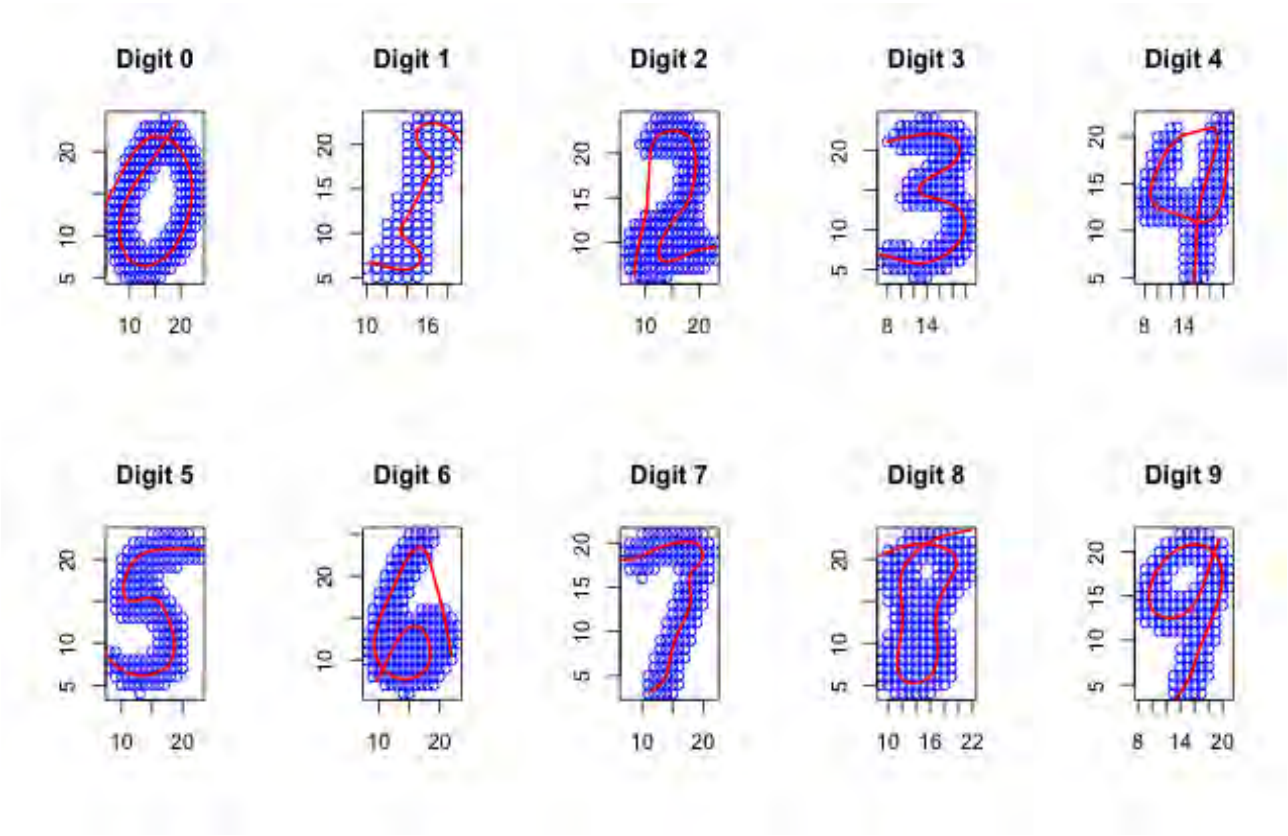


Figure 6.2: Averaged digits from 8000 training images shown by blue points together with fitted principal curves for each digit shown by the red curve.

Table 6.2: Averaged digit matrix for digit **3** with grayscale values converted to 1 highlighted in red.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
22	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.3: Two-column matrix for digit **3** with the column and row coordinates for all 1's in Table 6.2.

Column	Row
13	5
14	5
15	5
10	6
.	.
.	.
14	25

Classifying a new digit

To classify a new digit in the test set of images, the following steps are taken:

1. The grayscale value is converted to a 0 or 1, and then translated into a two column matrix of coordinates.

2. The Euclidean distance is calculated between the ten fitted principal curves and the coordinate matrix of the digit to be classified.

3. The new digit is classified to the digit represented by the principal curve, whose distance is a minimum.

An accuracy of 66% is given by classifying all digits in the test set, using the above procedure. The time taken (in seconds) to fit principal curves for each digit and classify all digits in the test set totalled to an amount of 95.67s. Table 6.4 represents the classification results where the rows represent the actual class and the columns represent the predicted class.

Digits 4 and 7 had the lowest accuracy rates, whereas digits 0 and 6 had the best. A possible explanation for digits with low accuracy rates could be because they are different ways these digits are handwritten. One could possibly improve on this by using more training images, to give a more accurate average of all digits.

Table 6.4: Classification results using a principal curve classifier

	0	1	2	3	4	5	6	7	8	9	Precision
0	148	0	1	0	0	0	18	0	4	0	86.5%
1	0	164	7	2	0	1	2	0	41	2	74.9%
2	8	0	161	11	0	0	30	0	13	0	72.2%
3	10	0	7	166	0	5	2	0	23	3	76.9%
4	5	0	11	2	70	2	26	0	35	47	35.4%
5	22	1	2	15	6	94	14	0	24	1	52.5%
6	9	0	13	0	0	2	167	0	13	0	81.9%
7	23	0	4	1	13	0	3	75	41	29	39.7%
8	4	1	6	6	2	8	15	1	154	0	78.2%
9	5	1	5	1	37	2	4	1	28	120	58.8%
Precision	63.2%	98.2%	74.2%	81.4%	54.7%	82.5%	59.4%	94.7%	41.0%	59.4%	**66%**

6.4 Other versions

Other versions of the principal curve classifier when classifying a new digit consisted of:

- Fitting a principal curve to the coordinate matrix of the digit,

- Stretching or shrinking the coordinate matrix of the digit, and then using Procrustes to rotate the matrix to optimally fit one of ten principal curves,

followed by calculating the distance between the ten principal curves. Both these versions have given lower accuracy rates of 50% and 37%, respectively.

6.5 Summary

Digit recognition has been described as a popular problem in machine learning practice. Most research papers have focused on other methods like k-Nearest Neighbours (kNN) as it outperforms the SVM classifier. Method kNN gives an accuracy of 95.4% with computational time of approximately 306s. Although what has been described as the principal curve classifier in Section 6.3 has not given the best accuracy results, it has improved on the time aspect on both SVM and kNN methods.

6.6 Recommendations

There are some other ideas that might be taken up to improve the performance of the procedures based on principal points and principal curves. The method kNN performs the best in the digit recognition problem, and therefore one could possibly combine the idea of principal curves and kNN with the hope of improvement. Because the averages of the digit images are taken for digits $0 - 9$, some variations of the handwritten images are lost, as noticed with digits **4** and **7** in the previous section. With kNN, there are at least K possible handwritten images for each digit, and therefore performs better in classifying new digits.

Chapter 7

Conclusion

In this dissertation, the importance of approximation was studied in the context of principal points. A literature review given in Chapter 2 researched the theorems related to univariate and multivariate distributions. This included the *Self-consistency algorithm*, the *k-means algorithm* and the *Principal Subspace theorem*. The chapter consisted of four main sections : the computation of principal points (when the distribution of data is known), the estimation of principal points (when the distribution of data is unknown), performance measures for choosing the best estimator and a criterion for selecting optimal k.

Principal points has proven to be useful in many applications such as clothing or equipment. A single size may be based on the mean of the distributions but multiple sizes (e.g. small, medium and large) can be based on the principal points of the distribution. In functional data analysis applications (Ramsay and Silverman 1997) when the data consist of curves, principal point methodology can be used to determine a small set of curves that represent the primary modes of variation (Flury, 1993). For instance, using principal points to estimate a set of representative longitudinal response curves from a clinical trial can be used to describe various patient types such as non-responders, drug responders, placebo responders, drug/placebo responders (Tarpey et al. 2003). In signal processing and digital communication the term quantization is used when a signal is represented by a finite set of values. The solution to finding the set of values that minimizes the loss of information due to quantization is mathematically equivalent to determining principal points.

A second approach to approximation was studied in the setting of principal curves. A short literature review given in Chapter 3 investigated the idea of approximating a distribution or data set in a lower dimension manifold through linear principal components. Principal curves are one-dimensional curves that are generalisations of the first linear principal component line. The chapter consisted of computing and estimating principal curves using the *principal-curve algorithm* by Hastie and Stuetzle (1989). The notion of principal curves extended to higher dimension namely principal surfaces was given.

Chapter 4 included an application on the estimation of principal points on two sets of data. The first set of data consisted of observations on the body mass of Southern African birds from five different habitats. Principal points for univariate distributions were calculated under each habitat. Finding the principal points in an ecological study can help better understand the interactions and relationships organisms have with the environment. The second application was to the well-known example of the Swiss Army head measurements in order to find k optimal sizes. Principal points were estimated using six different estimators for multivariate distributions and were compared using two performance measures.

Examples given in Hastie and Stuetzle (1989) of practical applications of principal curves are given. The first uses principal curves to align the approximately 950 magnets of the Standford linear collider which bend electron and positron beams and bring them into collision. The second example uses principal curves to pick up nonlinear systematic differences between two different types of assays for gold content in several samples of computer-chip waste. There are many other applications that have proven to be useful. To name a few, principal curves are used to extract skeletal structures of handwritten characters in faded documents (Singh et al., 1998). Reinhard and Niranjan (1998) apply principal curves to model the short tie spectrum of speech signals. Banfield and Raftery (1992) model the outlines of the ice floes in satellite images by closed principal curves and they develop a robust method which reduces the bias in the estimation process.

In Chapter 5, the use of principal points in computational methods in statistics was discussed. Two computational methods, bag plots and support vector machines were applied to the idea of principal points. There are many other known computational methods in statistics that can benefit from the use of principal points.

In Chapter 6, principal curves were applied to the famous problem of digit recognition with some success in the accuracy of classifying handwritten digits as well as computational time. Further improvements in the application of principal curves for digit recognition are possible within the research field of machine learning.

Bibliography

Banfield, J. and Raftery, A. (1992). Ice floe identification in satellite images using mathematical morphology and clustering about principal curves, *Journal of the American Statistical Association* **87**: 7–16.

Bofinger, E. (1957). Maximising the correlation of grouped observations, *Journal of the American Statistical Association* **65**: 1632–1638.

Cambanis, S., Huang, S. and Simons, G. (1984). On the theory of elliptically contoured distributions, *Journal of Multivariate Analysis* **11**: 368–385.

Cox, D. R. (1957). Note on grouping, *Journal of the American Statistical Association* **52**: 543–547.

Dalenius, T. and Gurney, M. (1951). The problem of optimum stratification ii, *Skandinavisk Aktuarietidskrift* **34**: 133–148.

Flury, B. D. (1990). Principal points, *Biometrika* **77**(1): 33–41.

Flury, B. D. (1993). Estimation of principal points, *Journal of the Royal Statistical Society* **42**(1): 139–151.

Flury, B. D. (1997). *A First Course in Multivariate Statistics*, New York: Springer.

Hartigan, J. A. (1975). *Clustering Algorithms*, New York: Wiley.

Hastie, T. (1984). Principal curves and surfaces, *Ph.D dissertation : Stanford University* .

Hastie, T. and Stuetzle, W. (1989). Principal curves, *Journal of the American Statistical Association* **84**(406): 502–516.

Hastie, T. and Weingessel, A. (2013). *princurve: Fits a Principal Curve in Arbitrary Dimension*. R package version 1.1-12.
URL: *http://CRAN.R-project.org/package=princurve*

Krzanowski, W. J. and Lai, Y. T. (1988). A criterion for determining the number of groups in a data set using sum-of-squares clustering, *Biometrics* **44**: 23–34.

Langner, R. E. (2004). *Summarising a Distribution with Self-consistent Approximations*, Masters Dissertation : University of Stellenbosch.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A. and Leisch, F. (2014). *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*. R package version 1.6-3.
URL: *http://CRAN.R-project.org/package=e1071*

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space, *Philosophical Magazine Series* **2**(6): 559–572.

Potzelberger, K. and Felsenstein, K. (1994). An asymptotic result on principal points for univariate distributions, *Optimisation* **28**: 397–406.

Reinhard, K. and Niranjan, M. (1998). Subspace models for speech transitions using principal curves, *Proc. Inst. of Acoustics* **20**: 53–60.

Rousseeuw, P. J., Ruts, I. and Tukey, J. W. (1999). The bagplot: A bivariate box plot, *Journal of the American Statistical Association* **53**(4): 382–387.

Roy, K. (2008). Evolution: dynamics of body size evolution, *Science* **321**: 1451–1452.

Singh, R., Wade, M. C. and Papanikolopoulos, N. P. (1998). Letter-level shape description by skeletonization in faded documents, *Proceedings of the fourth IEEE workshop on applications of computer vision* pp. 121–126.

Stampfer, E. and Stadlober, E. (2002). Methods of estimating principal points, *Communications in Statistics, Computation and Simulation* **31**: 261–277.

Tarpey, T. (1992). *Principal Points*, Ph.D. Dissertation : Indiana University.

Tarpey, T. (1997). Estimating principal points of univariate distributions, *Journal of Applied Statistics* **24**(5): 499–512.

Tarpey, T. (1998). Self-consistent patterns for symmetric multivariate distributions, *Journal of Classification* **15**: 57–79.

Tarpey, T. (1999). Self-consistency and principal component analysis, *Journal of the American Statistical Association* **94**: 456–467.

Tarpey, T. (2007). A parametric k-means algorithm, *Computational Statistics* **22**: 71–89.

Tarpey, T. and Li, L. (1995). Self-consistency: a fundamental concept in statistics, *Journal of Statistical Science* **11**: 229–243.

`R 2.15.0` (R Core Team, 2014). R: A language and environment for statistical computing. `http://www.R-project.org`.

Tukey, J. W. (1975). Mathematics and picturing of data, *Proceedings of the International Congress of Mathematicians* **2**: 523–531.

Tukey, J. W. (1977). *Exploratory Data Analysis*, Addison-Wesley, New York.

Wolf, P. and Bielefeld, U. (2013). *aplpack: Another Plot PACKage: stem.leaf, bagplot, faces, spin3R, plotsummary, plothulls, and some slider functions*. R package version 1.2.9. **URL:** *http://CRAN.R-project.org/package=aplpack*

Zoppé, A. (1997). On uniqueness and symmetry of self-consistent points of univariate continuous distributions, *Journal of Classifications* **14**: 147–158.

APPENDIX

Appendix A

Diagrams and Plots

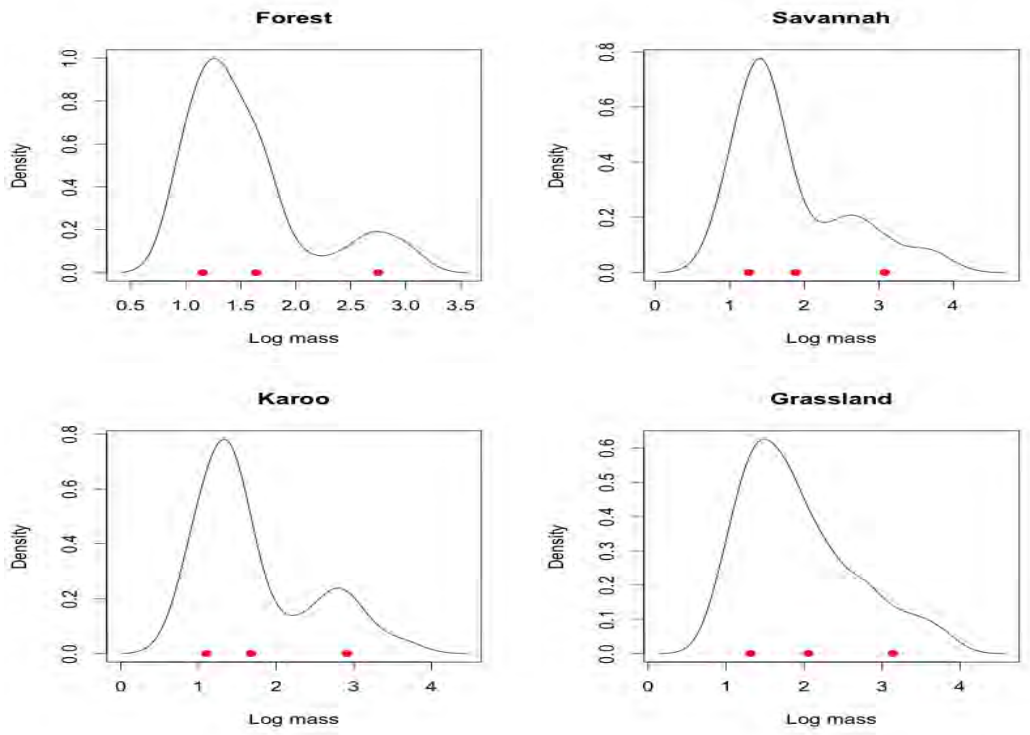


Figure A.1: Density plots of different habitats illustrating estimation of principal points under the Unconstrained k-means Estimator

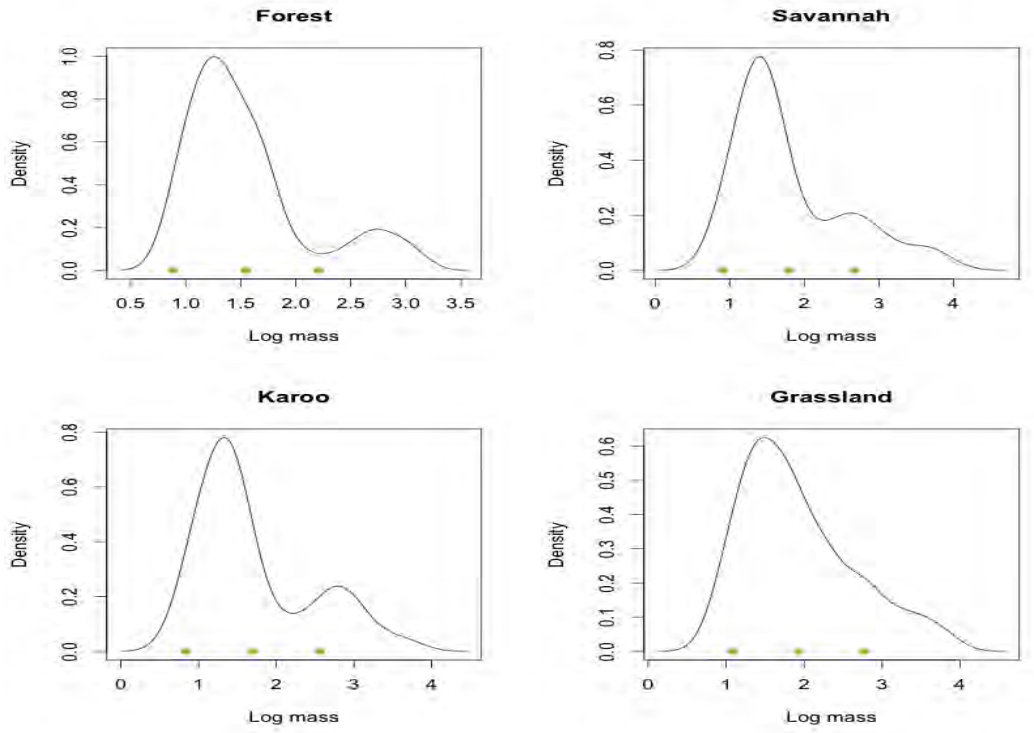


Figure A.2: Density plots of different habitats illustrating estimation of principal points under the Parametric Estimator

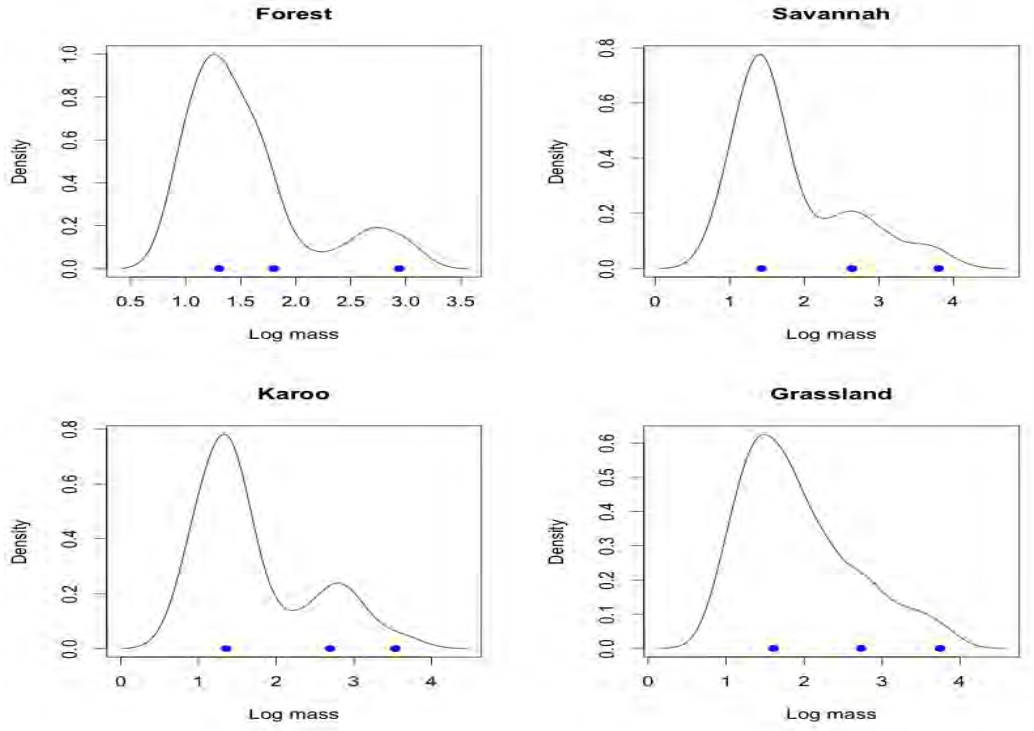


Figure A.3: Density plots of different habitats illustrating estimation of principal points under the Quantile Estimator

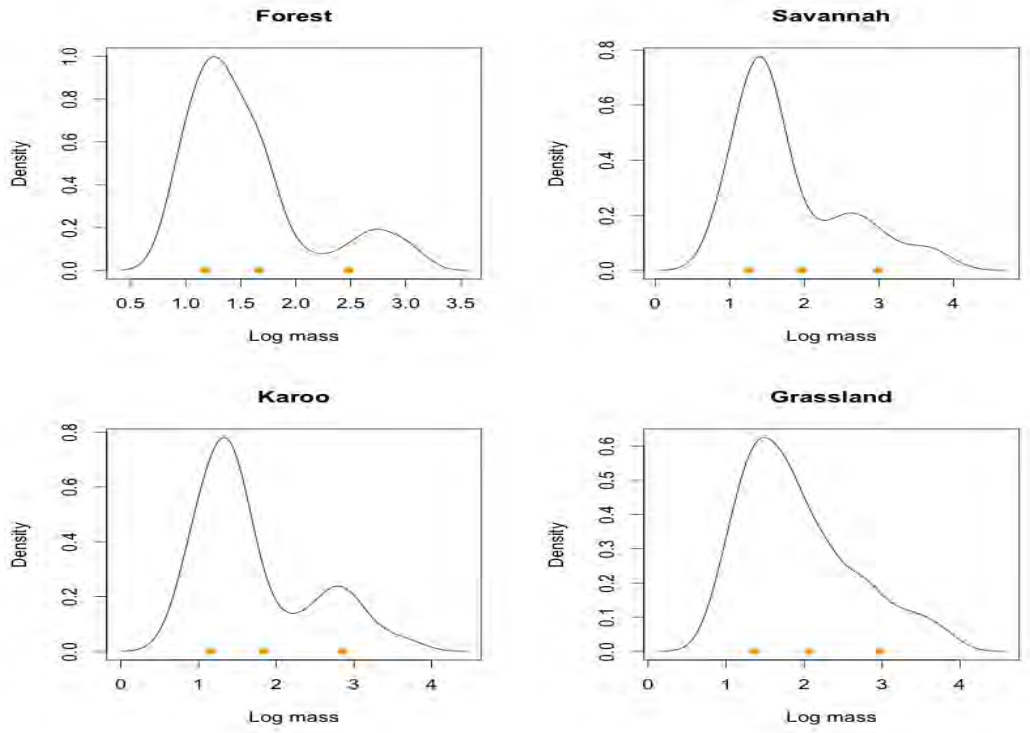


Figure A.4: Density plots of different habitats illustrating estimation of principal points under the Quick Estimator

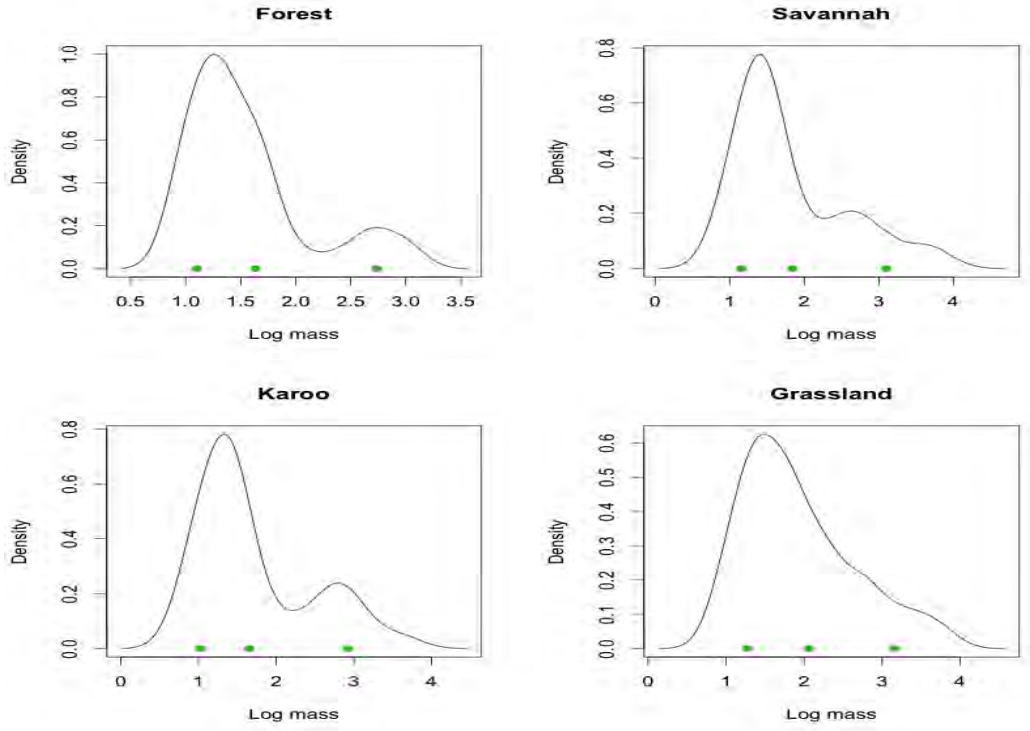


Figure A.5: Density plots of different habitats illustrating estimation of principal points under the Smooth k-means Estimator

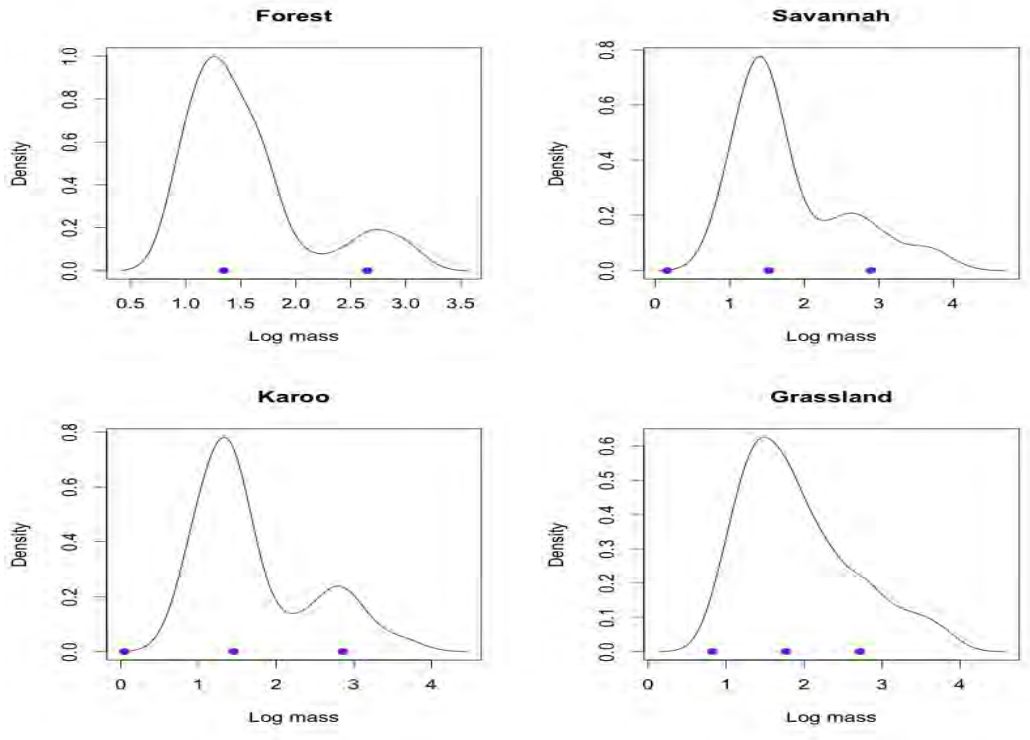


Figure A.6: Density plots of different habitats illustrating estimation of principal points under the KMS1 Estimator

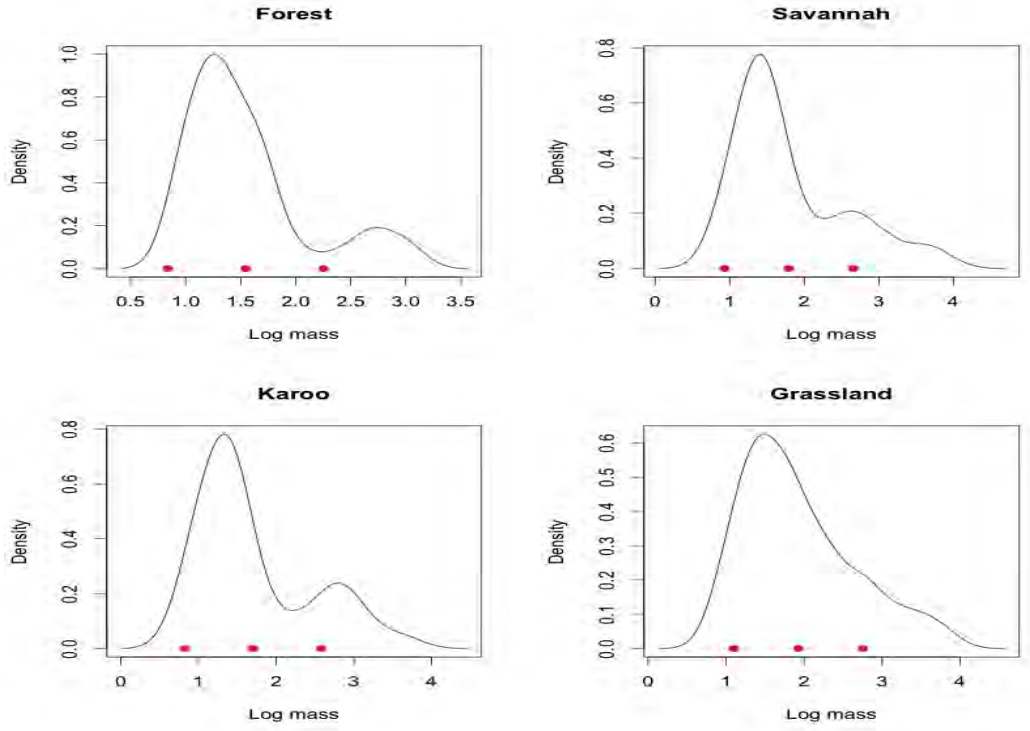


Figure A.7: Density plots of different habitats illustrating estimation of principal points under the KMS2 Estimator

Appendix B

R Codes

The following functions are written as part of the research project.

B.1 Univariate Estimators

The following code for the Univariate Estimators is gathered from Langner (2004).

```
# Necessary Functions to estimate principal points for Univariate Distributions
# Langner E.R (2004). Summarising a distrbutions with self-consistent approximations.

Q.norm = function(x,mean=0,sd=1)
  ##Computes the Q-function of the normal(mean,sd^2)-distribution:
{
  return(mean*pnorm((x-mean)/sd)-sd*dnorm((x-mean)/sd))
}

Q.expo = function(x,rate=1)
  ## Computes the Q-function of the exponential(rate)-distribution:
{
  if(any(x<0)) stop(message="only non-negative values are alowed for 'x'")
  answ = rep(0,length(x))
  if(any(x==Inf)) answ[x==Inf] = 1/rate
  answ[x!=Inf] = 1/rate-exp(-rate*x[x!=Inf])*(x[x!=Inf]+1/rate)
  return(answ)
}

Q.dbexpo = function(x,rate=1,mu=0)
  ##Computes the Q-function of the double-exponential(rate,mu)-distribution:
{
  if(rate<=0) stop(message="'rate' must be positive")
  answ <- rep(0,length(x))
  if(any(x<=mu))
  {
    if(any(x==-Inf)) answ[x==-Inf] = 0
    answ[(x<=mu)&(x!=-Inf)] = 0.5*exp(rate*(x[(x<=mu)&(x!=-Inf)]-
                                            mu))*(x[(x<=mu)&(x!=-Inf)]-1/rate)
  }
  if(any(x>mu))
    {
      if(any(x==Inf)) answ[x==Inf] = mu
      answ[(x>mu)&(x!=Inf)] = -0.5*exp(-rate*(x[(x>mu)&(x!=Inf)]-
    }
  return(answ)
}

##  Computes the conditional mean of the pdf specified by 'distr' over the interval [a,b].
##  'distr' is a character string specifying the distribution.
##  The only options to choose from are:
##  "norm" "expo" "dbexpo" "numerical". For the "numerical" case numerical integration is used
     and then
##  'pdf' must be specified - the density function.
##  For the first three cases analytical formulae are used and then 'distr' overrides 'pdf'.
##  All parameters for distributions are given in ...
##  The probability to be in [a,b] is also calculated.
cond.mean = function(a,b,distr="norm",pdf=NULL,...)
```

```
{
  if(a >= b) stop(message = "a must be less than b")
  distr.opts = c("norm","expo","dbexpo","numerical")
  ## compute the probability and conditional mean for the case specified by 'distr'
  if(distr=="norm")
  {
    prob = pnorm(b,...)-pnorm(a,...)
    cond.mean = (Q.norm(b,...)-Q.norm(a,...))/prob
  }
  else if(distr=="expo")
  {
    prob = pexp(b,...)-pexp(a,...)
    cond.mean = (Q.expo(b,...)-Q.expo(a,...))/prob
  }
  else if(distr=="dbexpo")
  {
    prob = pdbexp(b,...)-pdbexp(a,...)
    cond.mean = (Q.dbexpo(b,...)-Q.dbexpo(a,...))/prob
  }
  else if(distr=="numerical")
  {
    if(is.null(pdf)) stop(message="if numerical integration is used, 'pdf' must be specified")
    prob = integrate(pdf,lower=a,upper=b,...)$value
    intergrand = function(x,pdf,...) x*pdf(x,...)
    cond.mean = integrate(intergrand,lower=a,upper=b,pdf=pdf,...)$value/prob
  }
  else
  {
    print(distr.opts)
    stop(message="'distr' can only be one of the above options")
  }
  return(c(cond.mean=cond.mean,prob=prob))
}

find.sc.points = function(scpts.init=NULL,distr="norm",pdf=NULL,...,distr.lims=NULL)
  ## A fast way of finding a set of K self-consistent points for the
  ## normal(mu,sd^2)-// exponential(rate)-// double-exponential(rate,mu)-//
  ## 'scpts.init' must be specified to initialise the algorithm
  ## 'distr' is a character string specifying the distribution.
  ## The only options to choose from are:
  ## "norm" "expo" "dbexpo" "numerical".
  ## For the "numerical" case numerical integration is used and then
  ## 'pdf' must be specified - the density function.
  ## For the first three cases analytical formulae are used and then
  ## 'distr' overrides 'pdf'.
  ## All parameters for distributions are given in ...
  ## 'distr.lims' is a vector giving the range of the distribution.
{
  distr.opts = c("norm","expo","dbexpo","numerical") ## make a list of the parameters of the
      distribution
  ## error message if no parameters are given
  arg.list = list(...)
  if(length(arg.list)==0) stop(message="parameters must be given") ## 'scpts.init' must be
      specified and have length > 1
  if(!is.null(scpts.init))
  {
    scpts.init = sort(scpts.init)
    K = length(scpts.init)
  }
  else stop(message="scpts.init must be initialised, length indicating number of self-
      consistent points to be computed")
  if(is.null(distr.lims)) stop(message="distr.lims must be specified")
  if(K==1)
  {
    scpts = cond.mean(a=distr.lims[1],b=distr.lims[2],distr=distr, pdf=pdf,...)[1]
    probs = 1
  }
  else
  {
    old.scpts = scpts.init
    bounds = c(distr.lims[1],(old.scpts[-1]+old.scpts[-K])/2,distr.lims[2]) ## self-
        consistency algorithm iterates until convergence
    repeat
```

```
    {
      probs = NULL
      scpts = NULL
      for(k in 1:K)
      {
        answ = cond.mean(a=bounds[k],b=bounds[k+1],distr=distr,pdf=pdf,...)
        probs = c(probs,answ[2])
        scpts = c(scpts,answ[1])
      }
      if(max(abs(scpts-old.scpts))<1e-010) break
      bounds = c(distr.lims[1],(scpts[-1]+scpts[-K])/2,distr.lims[2])
      old.scpts = scpts
    }
  }
  ## compute MSE of resultant scpts
  if(distr=="norm") MSE = (arg.list$sd^2+arg.list$mean^2)-sum(probs*scpts^2)
  else if(distr=="expo") MSE = 2/arg.list$rate^2-sum(probs*scpts^2)
  else if(distr=="dbexpo") MSE = (2/arg.list$rate^2+arg.list$mu^2)-sum(probs*scpts^2)
  else if(distr=="numerical")
  {
    library(Integrat)
    MSE = gkint(function(x,pdf,...)(x^2)*pdf(x,...),
                low=distr.lims[1],hi=distr.lims[2],pdf=pdf,...)$result-sum(probs*scpts^2)
  }
  listy = list("scpts"=scpts,"probs"=probs,"MSE"=MSE)
  return(listy)
}

get.FQ.funcs = function(dens.list=NULL)
  ## 'dens.list': output from the function density().
  ## Computes the functions F(.) and Q(.) for the estimated density function
  ## using numerical integration as in Algorithm 4.1 p.202 of
  ## Numerical Analysis, Burden R.L and Faires J.D, Sixth Edition.
{
  if(is.null(dens.list)) stop(message="a density estimate is needed as input")
  else
  {
    x = dens.list$x
    y = dens.list$y
  }
  y.ster = x*y
  n = length(x)
  h = x[2]-x[1]
  T1 = c(0,0)
  T2 = c(0,0)
  T.mat = c(0,0)
  for(i in 1:(n-1))
  {
    if((i/2)==round(i/2)) T2 = T2+c(y[i],y.ster[i])
    else T1 = T1+c(y[i],y.ster[i])
    T.mat = rbind(T.mat,2*T2+4*T1)
  }
  T0.mat = cbind(c(0,y[2:n]+y[1]),c(0,y.ster[2:n]+y.ster[1]))
  FQ.mat = h*(T0.mat+T.mat)/3
  FQ.mat = rbind(c(0,FQ.mat[2,2]),FQ.mat[-1,])
  dimnames(FQ.mat) = list(NULL,c("F.x","Q.x"))
  return(FQ.mat=FQ.mat)
}
```

necessary.R

```
# 1. Unconstrained k-means estimator
KM.est =function(sp,K,pp.init=NULL)
 # sp: sample
 # K: number of principal points to be estimated
{
  if(K==1) pp.est = c(mean(sp))
  else
  {
    if(is.null(pp.init)) pp.init = K
    pp.est = c(sort(kmeans(sp,centers=pp.init)$centers))
  }
  names(pp.est) = c(paste("pp.est",1:K,sep=""))
```

```
  return(pp.est)
}

# 2. Quantile estimation - Tarpey (1997).
QN.est = function(sp,K,distr="norm")
 # Must know the distributions:
  ## "norm" == normal(0,1)
  ## "expo" == exp(1) and
  ## "dbexpo" == dbexp(1,0)
 # sp: sample.
 # K: number of principal points to be estimated.
{
  if(distr=="norm")
  {
    props = pnorm(q=find.sc.points(scpts.init=seq(-3,3,len=K),distr="norm",
                                   mean=0,sd=1,distr.lims=c(-30,30))$scpts,mean=0,sd=1)
  }
  if(distr=="expo")
  {
    props = pexp(q=find.sc.points(scpts.init=seq(0,3,len=K),distr="expo",
                                  rate=1,distr.lims=c(-30,30))$scpts,rate=1)
  }

  if(distr=="dbexpo")
  {
    props = pdbexp(q=find.sc.points(scpts.init=seq(-3,3,len=K),distr="dbexpo",
                                    rate=1,mu=0,distr.lims=c(-30,30))$scpts,rate=1,mu=0)
  }

  pp = c(sort(sp)[round(props*length(sp))])
  names(pp) = c(paste("pp",1:K,sep=""))
  return(pp)
}


# 3. Smooth k-means Estimator
SKM.est = function(sp,K,dens.num=1000,tol=1e-007)
  # sp: sample
  # K: number of principal points to be estimated
{
  ## density estimate from sp
  f.hat = density(sp,n=dens.num)
  ## numericaly calculate the distribution function F(x) and the
  ## cond exp function Q(x) of f.hat
  FQ.mats.hat = get.FQ.funcs(dens.list=f.hat)
  F.hat = cbind(f.hat$x,FQ.mats.hat[,1])
  Q.hat = cbind(f.hat$x,FQ.mats.hat[,2])
  ## initialise pp.est
  props = 1:K/(K+1)
  pp.est = NULL
  for(p in props) pp.est = c(pp.est,F.hat[order(abs(F.hat[,2]-p))[1],1])
  cuts = (pp.est[-1]+pp.est[-K])/2
  bound.index = NULL
  for(pnt in cuts) bound.index = c(bound.index,order(abs(pnt-f.hat$x))[1])
  bound.index = c(1,bound.index,dens.num)
  old.pp.est = pp.est
  repeat
  {
    pp.est = NULL
    pp.est = (Q.hat[bound.index[-1],2]-Q.hat[bound.index[-(K+1)],2])/(F.hat[bound.index
        [-1],2]-F.hat[bound.index[-(K+1)],2])
    cuts = (pp.est[-1]+pp.est[-K])/2
    bound.index = NULL
    for(pnt in cuts) bound.index = c(bound.index,order(abs(pnt-f.hat$x))[1])
    bound.index = c(1,bound.index,dens.num)
    if(max(abs(pp.est-old.pp.est)<tol))
      break
    else old.pp.est = pp.est
  }
  pp = c(pp.est)
  names(pp) = c(paste("pp",1:K,sep=""))
  return(pp)
}
```

```
# 4. KMS1 estimator
KMS1.est = function(sp,K,mean.true=F)
  ## KMS1 estimation: under assumption of distribution symmetric around mean
  # sp : sample
  # K : number of principal points to be estimated
  # if 'mean.true'=T then the center value is estimated by the sample mean
  # else by the sample median
{
  if(K==1) pp.est = c(mean(sp),proc.time()-tm.begin)
  else
  {
    mu.hat = ifelse(mean.true, mean(sp), median(sp))
    aug.sp = c(sp,2*mu.hat-sp)
    pp.est = c(sort(kmeans(aug.sp,centers=K)$centers))
  }
  pp = c(pp.est)
  names(pp) = c(paste("pp",1:K,sep=""))
  return(c(mu.hat,pp))
}

# 5. KMS2 Estimator
KMS2.est = function(sp,K)
  ## KMS2 estimation: under assumption of principal points symmetric around mean
  # sp : sample
  # K : number of principal points to be estimated
{
  if(K==1) pp.est = c(mean(sp))
  else
  {
    K.star = floor(K/2)
    even = T
    if(K.star!=K/2) even = F
    mu.hat = mean(sp)
    sp.star = abs(sp-mu.hat)
    pp.est.star = KM.est(sp.star,K=K.star)[1:K.star]
    if(even) pp.est = c(mu.hat+c(-rev(pp.est.star),pp.est.star))
    else
    {
      for(i in 1:20)
      {
        select = (sp.star > pp.est.star[1]/2)
        pp.est.star = KM.est(sp.star[select],K=K.star,pp.init=pp.est.star)[1:K.star]
      }
      pp.est = c(mu.hat+c(-rev(pp.est.star),0,pp.est.star))
    }
  }
  names(pp.est) = c(paste("pp.est",1:K,sep=""))
  return(pp.est)
}

# 6 & 7. Parametric k-means estimator & Quick Estimator
QE.est = function(sp,K,QE1=T,distr="norm")
  ## Quick estimates based on a result from Felsenstein (1994)
  # sp : sample
  # K : number of principal points to be estimated
  # QE1 = T => Parametric k-means Estimator
  # QE1 = F => Quick Estimator
  # 'distr': "norm","expo","dbexpo"
{
  props = (1:K)/(K+1)
  if(!QE1)
  {
    label = "Non parametric density"
    ## nonparametric density estimation
    f.hat = density(sp,n=500)
    f.third.hat = f.hat
    f.third.hat$y = f.third.hat$y^(1/3)
    const = get.FQ.funcs(dens.list=f.third.hat)[500,1]
    #const = get.FQ.funcs(dens.list=f.third.hat)$FQ.mat[500,1]
    g.hat = f.third.hat
    g.hat$y = f.third.hat$y/const
    G.hat = get.FQ.funcs(dens.list=g.hat)[,1]
```

```
    G.hat.quantiles = NULL
    for(p in props) G.hat.quantiles = c(G.hat.quantiles,g.hat$x[order(abs(G.hat-p))[1]])
  }
  else
  {
    ## ML estimation
    label = paste("Maximum likelihood",distr)
    distr.opts = c("norm","expo","dbexpo")
    if(distr=="norm")
    {
      mean.hat = mean(sp)
      sd.hat = sqrt(var(sp,na.rm=F))
      # return(c(mean.hat,sd.hat))
      G.hat.quantiles = qnorm(props,mean=mean.hat,sd=sd.hat*sqrt(3))
    }
    else if(distr=="expo")
    {
      rate.hat = 1/mean(sp)
      # return(rate.hat)
      G.hat.quantiles = qexp(props,rate=rate.hat/3)
    }
    else if(distr=="dbexpo")
    {
      mu.hat = median(sp)
      rate.hat = 1/mean(abs(sp-mu.hat))
      # return(c(mu.hat,rate.hat))
      G.hat.quantiles = NULL
      if(any(props<=0.5)) G.hat.quantiles = log(2*props[props<=0.5])/(rate.hat/3)
      if(any(props>0.5)) G.hat.quantiles = mu.hat+c(G.hat.quantiles,-log(2-2*props[props>0.5])
          /(rate.hat/3))
    }
    else
    {
      print(distr.opts)
      stop(message="distr can only be one of the above options")
    }
  }
  pp = c(G.hat.quantiles)
  names(pp) = c(paste("pp",1:K,sep=""))
  return(pp)
}
```

EstimatorsUni.R

B.2 Multivariate Estimators

```
# 1. Unconstrained k-means Estimator
unparametrickmeans <- function(x,k)
# x : nxp data matrix
# k : number of principal points
{
  x.mean = apply(x,2,mean)
  if(k==1) pp = mean(x)
  else
  {
    pp = kmeans(x,k,iter.max=50)$centers
  }
  return(pp)

}


# 2. Parametric k-means
parametrickmeans <- function(x, k, ns)
# x : nxp data matrix
# k : number of principal points
# ns : Sample size
{
```

```
  n = dim(x)[1]
  p = dim(x)[2]
  xbar = apply(x,2,mean)
  S = cov(x)
  e = eigen(S)
  # Simulate N(xbar, S) data with sample size ns and put the data in xsim:
  xsim = sweep(matrix(rnorm(ns*p), ns, p)%*%diag(sqrt(e$value))%*%t(e$vector), 2, -xbar)
  pp = kmeans(xsim,k,iter.max=50)$centers
  return(pp)
}


# 3. Subspace Constraint Estimator
subconstr <- function(x,k,q)
# x : nxp data matrix
# k : number of principal points
# q : Number of principal components
  {
  p = dim(x)[2]
  u = svd(x)$u
  d = diag(svd(x)$d)
  v = svd(x)$v
  scores = u%*%d
  pp = kmeans(scores[,1:q],centers=k,iter.max=50)$centers%*%t(v[,1:q])
  return(pp)
}

# 4. Pattern Constraints k = 4
# a. Line pattern
# b. Cross pattern
# c. Rectangular pattern


# Function that calculates the MSE(X,Y)
k.means.min <- function(y,X,k)
  # y : initial principal points
  # X : data
  # k : number of principal points
{
  n = nrow(X)
  p = ncol(X)
  v = svd(X)$v
  Y = matrix(y,nrow=k,ncol=2,byrow=F)%*%t(v[,1:2])

  distance = matrix(nrow=n,ncol=k)
  g = c()
  for(i in 1:n)
  {
    for(K in 1:k)
    {
      distance[i,K] = dist(rbind(X[i,],Y[K,]))
    }
    g[i] = which.min(distance[i,])
  }

  N = table(g)
  N.vec = c()
  Ymat = matrix(nrow=n,ncol=ncol(X))
  for(i in 1:n)
  {
    Ymat[i,] = Y[g[i],]
    N.vec[i] = N[g[i]]
  }

  ClusVar = apply( ((X-Ymat)^2), 1, sum) * N.vec
  sClus = sum(ClusVar)
  return(sClus)
}

# a) Line Pattern
linepattern <- function(X)
# X : nxp data matrix
{
```

```
  x.mean = matrix(1,nrow=4,ncol=1) %*% apply(X,2,mean)
  X = scale(X,center=T,scale=F)
  u = svd(X)$u
  d = diag(svd(X)$d)
  v = svd(X)$v
  ud = u%*%d

  a = range(ud[,1])[1]
  b = range(ud[,1])[2]
  dis = (b-a)/4
  zh = (b-a)

  MSE = c()
  Y = matrix(NA,nrow=10*zh,ncol=4)
  r = a
  for(i in 1:nrow(Y))
  {

    for (j in 1:4)
    {
      Y[i,j] = r+(dis*(j-1))
    }
    r = Y[i,1]+0.1
    yy = c(Y[i,],0,0,0,0)
    MSE[i] = k.means.min(yy,X,4)
  }
  num = which.min(MSE)
  yopt = c(Y[num,],0,0,0,0)
  Yopt = matrix(yopt,nrow=4,ncol=2,byrow=F)
   pp = Yopt %*% t(v[,1:2]) + x.mean
  return(pp)
}

# Cross Pattern
crosspattern <- function(X)
# X : nxp data matrix
{
  x.mean = matrix(1,nrow=4,ncol=1) %*% apply(X,2,mean)
  X =scale(X,center=T,scale=F)
  u = svd(X)$u
  d = diag(svd(X)$d)
  v = svd(X)$v
  ud = u%*%d

  x = c()
  y  = c()
  x[1] = max(-range(ud[,2])[1],range(ud[,2])[2])
  y[1] = x
  Y = matrix(NA,nrow=x[1]*10,ncol=8)
  Y[1,] = c(0,x,0,-x,y,0,-y,0)
  MSE = c()
  MSE[1] = k.means.min(c(Y[1,]),X,4)

  for(i in 2:nrow(Y))
  {
    x[i] = x[i-1]-0.1
    y[i] = y[i-1]-0.1
    Y[i,] = c(0,x[i],0,-x[i],y[i],0,-y[i],0)
    MSE[i] = k.means.min(c(Y[i,]),X,4)
  }
  num = which.min(MSE)
  yopt = Y[num,]
  Yopt = matrix(yopt,nrow=4,ncol=2,byrow=F)
   pp = Yopt %*% t(v[,1:2]) + x.mean
  return(pp)
}

# Rectangular Pattern
rectanglepattern <- function(X)
# X : nxp data matrix
{
  x.mean = matrix(1,nrow=4,ncol=1) %*% apply(X,2,mean)
  X = scale(X,center=T,scale=F)
```

```
  u = svd(X)$u
  d = diag(svd(X)$d)
  v = svd(X)$v
  ud = u%*%d

  x= c()
  y = c()
  x[1] = ceiling(max(-range(ud[,1])[1],range(ud[,1])[2]))
  y[1] = x
  YY = matrix(NA,ncol=8,nrow=250)
  YY[1,] = c(-x,x,-x,x,y,y,-y,-y)

  MSE = c()
  MSE[1] = k.means.min(c(YY[1,]),X,4)
  for(i in 2:250)
  {
    x[i] = x[i-1]-0.1
    y[i] = y[i-1]-0.1
    YY[i,] = c(-x[i],x[i],-x[i],x[i],y[i],y[i],-y[i],-y[i])
    MSE[i] = k.means.min(c(YY[i,]),X,4)
  }

  num = which.min(MSE)
  yopt = YY[num,]

  Yopt = matrix(yopt,nrow=4,ncol=2,byrow=F)
  pp = Yopt  %*% t(v[,1:2]) + x.mean
  return(pp)
}
```

EstimatorMV.R

B.3 Performance Measures

```
aver = function(func,X,k,...)
## Function to compute the average estimates of principal points
{
  count = c(rep(0,k))
  for (i in 1:100)
  {
    PP = func(X,k,...)
    count = PP + count
  }
  PPf = count/100
  return(PPf)
}

SMSD.Uni = function(x,k,m)
## SMSD for Univariate Distributions
#  m = method used to estimate principal points
#      1 : Unparametric k-means
#      2 : Parametric k-means
#      3 : Quantile Estimator
#      4 : Quick Estimator
#      5 : Smooth k-means Estimator
#      6 : KMS1 Estimator
#      7 : KMS2 Estimator
{
  N = length(x)
  if(m==1) PP = aver(KM.est,x,k)
  else if (m==2) PP = aver(QE.est,x,k)
  else if (m==3) PP = aver(QN.est,x,k)
  else if (m==4) PP = aver(QE.est,x,k)
  else if (m==5) PP = aver(SKM.est,x,k)
  else if (m==6) PP = aver(KMS1.est,x,k)
  else if (m==7) PP = aver(KMS2.est,x,k)

  distance = matrix(nrow=N,ncol=k)
```

```
  for (n in 1:N)
  {
    for (K in 1:k)
    {
      distance[n,K] = dist(rbind(x[n],PP[K]))
    }
  }

  mindist = c()
  for(i in 1:N)
  {
    mindist[i] = min(distance[i,])^2
  }
  smsd = (1/N)*sum(mindist)
  return(c(m,smsd))
}

SMSD.Mv = function(x,k,m,q=3)
## SMSD for Multivariate Distributions
#  m = method used to estimate principal points
#      1 : Parametric k-means
#      2 : Unparametric k-means
#      3 : Subspace constraints
#      4 : Pattern Constraint (Line Pattern)
#      5 : Pattern Constraint (Cross Pattern)
#      6 : Pattern Constraint (Rectangular Pattern)
{
  N = nrow(x)
  if(m==1) PP = parametrickmeans(x,k,100000)
  else if (m==2) PP = unparametrickmeans(x,k)
  else if (m==3) PP = subconstr(x,k,q)
  else if (m==4) PP = linepattern(x)
  else if (m==5) PP = crosspattern(x)
  else if (m==6) PP = rectanglepattern(x)

  distance = matrix(nrow=N,ncol=k)
  for (n in 1:N)
  {
    for (K in 1:k)
    {
      distance[n,K] = dist(rbind(x[n,],PP[K,]))
    }
  }

  mindist = c()
  for(i in 1:N)
  {
    mindist[i] = min(distance[i,])^2
  }
  smsd = (1/N)*sum(mindist)
  return(c(m,smsd))
}
```

SMSD.R

```
PMSD.Uni = function(x,k,m)
## PMSD for Univariate Distributions
#  m = method used to estimate principal points
#      1 : Unparametric k-means
#      2 : Parametric k-means
#      3 : Quantile Estimator
#      4 : Quick Estimator
#      5 : Smooth k-means Estimator
#      6 : KMS1 Estimator
#      7 : KMS2 Estimator
{
  n = length(x)
  distance = matrix(nrow=n,ncol=k)
  for(j in 1:n)
  {
    if(m==1) pp = aver(KM.est,x[-j],k)
    else if (m==2) pp = aver(QE.est,x[-j],k)
    else if (m==3) pp = aver(QN.est,x[-j],k)
```

```
    else if (m==4) pp = aver(QE.est,x[-j],k)
    else if (m==5) pp = aver(SKM.est,x[-j],k)
    else if (m==6) pp = aver(KMS1.est,x[-j],k)
    else if (m==7) pp = aver(KMS2.est,x[-j],k)

    for (K in 1:k)
    {
      distance[j,K]=dist(rbind(x[j],pp[K]))
      print(c(j,K))
    }
  }

  mindist = c()
  for ( i in 1:n)
  {
    mindist[i] = min(distance[i,])^2
  }
  pmsd = (1/n)*sum(mindist)
  return(c(m,pmsd))
}

PMSD.Mv = function(x,k,m,q=2)
## PMSD for Multivariate Distributions
#  m = method used to estimate principal points
#      1 : Parametric k-means
#      2 : Unparametric k-means
#      3 : Subspace constraints
#      4 : Pattern Constraint (Line Pattern)
#      5 : Pattern Constraint (Cross Pattern)
#      6 : Pattern Constraint (Rectangular Pattern)
{
  n = nrow(x)
  distance = matrix(nrow=n,ncol=k)
  for(j in 1:n)
  {
    if(m==1) pp = unparametrickmeans(x[-j,],k)
    else if(m==2) pp = parametrickmeans(x[-j,],k,1000)
    else if(m==3) pp = subconstr(x[-j,],k,q)
    else if (m==4) pp = linepattern(x[-j,])
    else if (m==5) pp = crosspattern(x[-j,])
    else if (m==6) pp = rectanglepattern(x[-j,])

    for (K in 1:k)
    {
      distance[j,K]=dist(rbind(x[j,],pp[K,]))
      print(c(j,K))
    }
  }

  mindist = c()
  for ( i in 1:n)
  {
    mindist[i] = min(distance[i,])^2
  }
  pmsd = (1/n)*sum(mindist)
  return(c(m,pmsd))
}
```

PMSD.R

B.4 Principal curve classifier

```
# TRAIN DATA

elem <- sample(10000,8000)
Digits <- read.csv ("Digit-trainassign.csv")
digits <- Digits[elem,]
NN <- nrow(digits)

# Convert each digit to a 28x28 matrix
digit.mat <- lapply (1:NN, function(i) matrix (unlist(digits[i,-1]),byrow=T,ncol=28,dimnames=
    list(28:1,1:28)))
names(digit.mat) <- digits[1:NN,1]

# (1) Averages of digits
digit.vals <- levels(factor(digits[,1]))
num.digits <- length(digit.vals)
ave.list <- vector("list",num.digits)
for (i in 1:num.digits)
{  list.pos <- (1:NN)[names(digit.mat)==digit.vals[i]]
    mat <- matrix (0,nrow=28,ncol=28,dimnames=list(28:1,1:28))
    for (j in list.pos) mat <- mat + digit.mat[[j]]
    ave.list[[i]] <- mat/length(list.pos)   # mean of grey-scale values for each of 784 pixels
         for specific digit
}

# (2) select pixels with grey-scale larger than 64 (1/4 of 256) as pixels representing average
     digit
digit.list <- lapply (ave.list, function(x) { mat <- matrix (0,nrow=28,ncol=28,dimnames=list
    (28:1,1:28))
                                               mat[x>64] <- 1
                                              return(mat)})
# (3) Gives a 2 column matrix of coordinates
digit.points <- lapply (digit.list, function(x) {  mat <- NULL
                                                    for (i in 1:nrow(x))
                                                      for (j in 1:ncol(x))
if (x[i,j]==1) mat <- rbind (mat, c(as.numeric(colnames(x)[j]),as.numeric(rownames(x)[i])))
                                                    return (mat)
})


# Use digit.points to fit principal curves

# (4) Specify starting curve for each digit
plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"0",cex=40)
zero <- rbind(c(14,27),c(15,27),c(16,27),c(20,23),c(21,20),c(20,6),c(16,2),c(13,2),c(9,6),c
    (8,14.5),c(9,22),c(12,26),c(14,27))
lines (zero, col="red")

plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"1",cex=40)
one <- rbind(c(16.5,27),c(16.5,1))
lines (one, col="red")

plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"2",cex=40)
two <- rbind(c(8,20),c(9,24),c(14,27),c(16,27),c(20,23),c(21,20),c(19,16),c(10.5,7),c(8,2),c
    (22,2))
lines (two, col="red")

plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"3",cex=40)
three <- rbind(c(8,21),c(9,24),c(14,27),c(16,27),c(20,23),c(20,20),c(16,16),c(13,15.5),c
    (18,15),c(21,11),c(20,5),c(16,2),c(13,2),c(10,4),c(8,8))
lines (three, col="red")

plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"4",cex=40)
four <- rbind(c(22,9),c(6,9),c(18,27),c(18,1))
lines (four, col="red")
```

```
plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"5",cex=40)
five <- rbind(c(21,26.5),c(10.5,26.5),c(8.5,15),c(14,18),c(17,18),c(21,13),c(21,8),c(18,3),c
    (15,2),c(11,3),c(9,5),c(8,8))
lines (five, col="red")

plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"6",cex=40)
six <- rbind(c(20.5,22),c(19,25),c(16,27),c(13,27),c(9,23),c(8,14),c(8.5,8),c(9.5,5),c(12,2),c
    (15,1.5),c(18,2.5),c(21,8),c(21,11),c(19,16),c(16,17),c(14,17),c(10,14))
lines (six, col="red")

plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"7",cex=40)
seven <- rbind(c(7,26),c(21,26),c(17,20),c(12.5,9),c(11.5,1))
lines (seven, col="red")

plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"8",cex=40)
eight <- rbind(c(13,27),c(16,27),c(19,25),c(20,22),c(20,20),c(18,17),c(15,16),c(20,13),c(21,8)
    ,c(19,4),c(16,2),c(13,2),c(10,4),c(8,8),c(9,13),c(14,16),c(11,17),c(9,20),c(9,22),c(10,25)
    ,c(13,27))
lines (eight, col="red")

plot (expand.grid(1:28,1:28),pch=3)
text (x=14.5,y=14.5,"9",cex=40)
nine <- rbind(c(8,7),c(10,3),c(13,2),c(16,2),c(19,5),c(21,14),c(21,20),c(19,25),c(16,27),c
    (13,27),c(10,25),c(8,20),c(8,17),c(10,13),c(14,11.5),c(18,13),c(20,16))
lines (nine, col="red")

given.digits <- list (zero,one,two,three,four,five,six,seven,eight,nine)

# (5) Fit principal curves
pcurveA.out <- vector("list",length(digit.vals))
for (k in 1:length(digit.vals))
{
  # dev.new()
  pcurveA.out[[k]] <- principal.curve(digit.points[[k]], plot=T, df=10, start=given.digits[[k
      ]])
  #title (main=digit.vals[k])
  print(k)
}
pcurveA.s <- lapply(pcurveA.out, function(x)x$s[x$tag,])

## CLASSIFICATION OF NEW DIGITS

# TEST DATA
elem = as.numeric(row.names(digits))
digits.test <- Digits[-elem,]
N <- nrow(digits.test)

test.mat <- lapply (1:N, function(i) matrix (unlist(digits.test[i,-1]),byrow=T,ncol=28,
    dimnames=list(28:1,1:28)))
true.digit = as.factor(digits.test[,1])

# Distance function
dist.D12 <- function(Y1, Y2)
{
  E1 <- apply(Y1,1,function(x)sum(x^2))
  E2 <- apply(Y2,1,function(x)sum(x^2))
  E1mat <- matrix(rep(E1,nrow(Y2)),nrow=nrow(Y1))
  E2mat <- matrix(rep(E2,each=nrow(Y1)),ncol=nrow(Y2))
  D <- E1mat + E2mat - 2*Y1%*%t(Y2)
  sqrt(D)
}

# Calculate distance between observed values for the digit to classify and each of the
    principal curves
predictA.vec1 <- rep(NA,2000)
for (i in 1:2000)
{
  mat <- NULL
  current.digit <- test.mat[[i]]
```

```
  for (x in 1:nrow(current.digit))
    for (y in 1:ncol(current.digit))
      if (current.digit[x,y]>64) mat <- rbind (mat, c(as.numeric(colnames(current.digit)[y]),
          as.numeric(rownames(current.digit)[x])))
  num.points <- nrow(mat)
  d.sq <- rep(NA,length(digit.vals))
  for (k in 1:length(digit.vals))
  {
    D <- dist.D12(pcurveA.s[[k]],mat)
    d.sq[k] <- sum(apply(D,2,min)^2)
  }
  predictA.vec1[i] <- digit.vals[order(d.sq)[1]]
}

table(true.digit,predictA.vec1)
accuracyA1 = sum(predictA.vec1==true.digit)/2000

# Other versions

# Fit principal curve to new digit
predictA.vec2 <- rep(NA,2000)
for (i in 1:2000)
{
  mat <- NULL
  current.digit <- test.mat[[i]]
  for (x in 1:nrow(current.digit))
    for (y in 1:ncol(current.digit))
      if (current.digit[x,y]>64) mat <- rbind (mat, c(as.numeric(colnames(current.digit)[y]),
          as.numeric(rownames(current.digit)[x])))
  digit.pcurve <- principal.curve(mat,plot=F)
  digit.s <- digit.pcurve$s[digit.pcurve$tag,]
  num.points <- nrow(digit.s)
  d.sq <- rep(NA,length(digit.vals))
  for (k in 1:length(digit.vals))
  {
    D <- dist.D12(pcurveA.s[[k]],digit.s)
    d.sq[k] <- sum(apply(D,2,min)^2)
  }
  predictA.vec2[i] <- digit.vals[order(d.sq)[1]]
}

table(true.digit,predictA.vec2)
accuracyA3 = sum(true.digit==predictA.vec2)

# Center pcurves and digit to classify; stretch\shrink digit coordinate matrix
# and use procrustes to rotate to optimally fit.
library(vegan)
predictA.vec3 <- rep(NA,2000)
for (i in 1:2000)
{
  mat <- NULL
  current.digit <- test.mat[[i]]
  for (x in 1:nrow(current.digit))
    for (y in 1:ncol(current.digit))
      if (current.digit[x,y]>64)
      {
        mat <- rbind (mat, c(as.numeric(colnames(current.digit)[y]),as.numeric(rownames(
            current.digit)[x])))
      }

  num.points <- nrow(mat)
  d.sq <- rep(NA,length(digit.vals))
  for (k in 1:length(digit.vals))
  {
    # Center
    pcurve = scale(pcurveA.s[[k]],center=T)
    mat = scale(mat,center=T)

    # Make number of rows the same
    if( nrow(pcurve) > nrow(mat))
    {
      random = sample(nrow(pcurve),nrow(pcurve)-nrow(mat))
      pcurve = pcurve[-random,]
```

```
    } else if(nrow(pcurve) < nrow(mat)) {
      random = sample(nrow(mat),nrow(mat)-nrow(pcurve))
      mat = mat[-random,]
    } else if (nrow(pcurve) == nrow(mat)) {
      mat = mat
    }

    # Shrink / Stretch
    # x
    x1 = abs(min(pcurve[,1]) - min(mat[,1]))
    x2 = abs(max(pcurve[,1]) - max(mat[,1]))

    # y
    y1 = abs(min(pcurve[,2]) - min(mat[,2]))
    y2 = abs(max(pcurve[,2]) - max(mat[,2]))

    if( min(pcurve[,1]) > min(mat[,1]) | max(pcurve[,1]) < max(mat[,1]))
    {
      mat[,1] = mat[,1]*(1-min(x1,x2))
    } else mat[,1] = mat[,1]*(1+max(x1,x2))

    if( min(pcurve[,2]) > min(mat[,2]) | max(pcurve[,2]) < max(mat[,2]))
    {
      mat[,2] = mat[,2]*(1-min(x1,x2))
    } else mat[,2] = mat[,2]*(1+max(x1,x2))

    # Rotate to fit
    pro.mat <- procrustes(pcurve,mat,scale=T)$Yrot

    # Distances
    D <- dist.D12(pcurve,pro.mat)
    d.sq[k] <- sum(apply(D,2,min)^2)
  }
  predictA.vec3[i] <- digit.vals[order(d.sq)[1]]
}

table(true.digit,predictA.vec3)
accuracyA3 = sum(true.digit==predictA.vec3)/2000

# SVM classification
trainDigit = factor(digits[,1])
ptm = proc.time()[3]
svm.results <- predict(svm(trainDigit ~ .,kernel="radial",gamma=0.0001,cost=10, data=digits
    [,-1]),
                       newdata=digits.test[,-1])
time.svm = proc.time()[3] - ptm
table (true.digit,svm.results)
svm.accuracy <- sum(true.digit==svm.results)/2000

# Knn classification
ptm = proc.time()[3]
knn.results <- knn(digits[,-1],digits.test[,-1], trainDigit)
time.knn = proc.time()[3] - ptm
table (true.digit,knn.results)
knn.accuracy <- sum(true.digit==knn.results)/2000
```

mydigit.R